Abstraction Sampling in Graphical Models

F. Broka, R. Dechter, A. Ihler and **K. Kask**
University of California, Irvine
Irvine, CA 92697, USA
{fbroka,dechter,ihler,kkask}@ics.uci.edu

Abstract

We present a new sampling scheme for approximating hard to compute queries over graphical models, such as computing the partition function. The scheme builds upon exact algorithms that traverse a weighted directed state-space graph representing a global function over a graphical model (e.g., probability distribution). With the aid of an abstraction function and randomization, the state space can be compacted (or trimmed) to facilitate tractable computation, yielding a Monte Carlo Estimate that is unbiased. We present the general scheme and analyze its properties analytically and empirically, investigating two specific ideas for picking abstractions - targeting reduction of variance or search space size.

1 INTRODUCTION

Imagine that we want to compute a function over a weighted directed graph where the graph is given implicitly, e.g., using a generative state-space search model whose explicit state graph is enormous. We therefore need to resort to approximations such as Monte-Carlo schemes. We focus on weighted search trees over probabilistic graphical models where nodes represent partial variable assignments (or configurations) and arcs are associated with weights describing probabilistic quantities from the model. The task is to compute the partition function - sum cost of all paths in the tree. Some Monte Carlo methods draw independent samples of full configurations (full paths) one variable at a time (e.g., Forward sampling). The key idea of the scheme we propose is that, guided by an abstraction function, each sample (or a "probe") is a sampled subtree representing *multiple configurations*. We argue that, under some conditions over the abstraction, such a sampled tree representing k configurations can be a more accurate estimator than k independent samples.

Our sampling scheme uses an *abstraction function* that partitions the nodes *at each level in the search tree* into subsets of *abstract states* under the intuition that nodes in the same abstract state root similar subtrees and therefore a single member from each can represent all others. Our *Abstraction Sampling (AS)* scheme brings together ideas from statistics and search to exploit their respective strengths. Search is a systematic process that can explore all configurations in a structured manner, *once*. "It does not leave any stone unturned and does not turn any stone more than once" [Pearl, 1984]. Sampling on the other hand explores only a subset of the paths, that can stochastically cover the whole search space. Abstraction Sampling allows a transition between search and sampling by generating and searching a subtree, and therefore a corresponding subset of configurations, in a coordinated manner that can overcome redundancy and some of the variance associated with random independent samples.

From a search perspective abstraction can be viewed as a license to merge nodes that root similar subtrees, sampling one of the subtrees, thus creating a more compact graph that can be searched more efficiently. From a sampling perspective, an abstract state can be viewed as a form of a "strata" within a stratified sampling scheme [Rubinstein and Kroese, 2007, Rizzo, 2007] where the process of stratified sampling is applied layer by layer. The variance reduction we hope to get rests on similar principles of variance reduction in stratified sampling, as we will elaborate more later.

Abstraction Sampling is inspired by the early work of Knuth and Chen [Knuth, 1975, Chen, 1992] who proposed a method for estimating quantities that can be expressed as aggregates (e.g., sum) of functions defined over the nodes in the graph. Our work extends this work to more general queries over graphical models such as

the partition function and to weighted AND/OR trees.

Our Contributions. In this paper we present a new algorithmic framework, Abstraction Sampling, that combines search with stratified importance sampling for answering summation queries (e.g., partition function) over graphical models. Using the notion of abstraction function we can find a cost-effective balance betwen seach and sampling, that is tuned to the problem instance and to time and memory resources, using OR or AND/OR search algorithms. We prove unbiasedness and discuss variance reduction properties, and carry out an extensive empirical evaluation over multiple challenging benchmarks. Our results show that two classes of context-based abstractions (deterministic and randomized) can often significantly improve performance over the baseline of importance sampling, for both OR and AND/OR trees. We also show that our scheme is competitive by comparing it with two state-of-the-art importance sampling schemes.

2 BACKGROUND

A graphical model, such as a Bayesian or a Markov network [Pearl, 1988, Darwiche, 2009, Dechter, 2013] can be defined by a 3-tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, 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 $\psi_\alpha \in \mathbf{F}$ 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** of a graphical model associates each variable with a node, while arcs connect nodes whose variables appear in the scope of the same local function. A graphical model represents 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 $Z = \sum_X \prod_\alpha \psi_\alpha(X_\alpha)$.

AND/OR search spaces. A graphical model can be expressed via a weighted state space graph. In a simple OR search space, the states (or nodes) are partial assignments relative to a variable ordering, where each layer corresponds to a new assigned variable. A graphical model can be transformed into a more compact AND/OR search space [Dechter and Mateescu, 2007] by capturing conditional independences, thus facilitating more effective algorithms [Marinescu and Dechter, 2009]. The AND/OR search space is defined relative to a *pseudo tree* of the primal graph.

DEFINITION **1 (pseudo tree)** *A* pseudo tree *of an undirected graph $G = (V, E)$ is a directed rooted tree $\mathcal{T} = (V, E')$ such that every arc of G not in E' is a back-arc in $\mathcal{T}$ connecting a node in $\mathcal{T}$ to one of its ancestors. The arcs in E' may not all be included in E.*

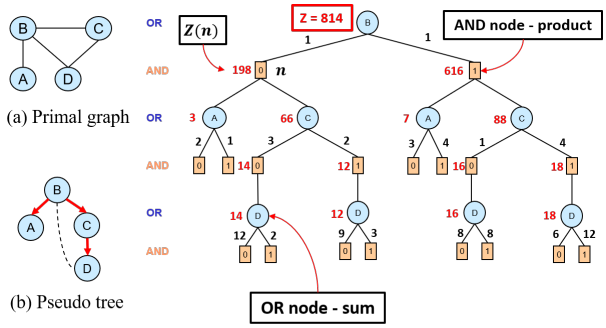


Figure 1: A Simple Graphical Model.

Given a pseudo tree $\mathcal{T}$ of a primal graph G, the *AND/OR search tree* $T_\mathcal{T}$ guided by $\mathcal{T}$ has alternating levels of OR nodes corresponding to the variables, and AND nodes corresponding to an assignment from its domain with edge costs extracted from the original functions $\mathbf{F}$ [Dechter and Mateescu, 2007]. Let s be a node in T_τ. We denote by $var(s)$ the last variable of the partial value assignment associated with s. So if s stands for $\bar{x}_{1..p} = (x_1, x_2, ..., x_p)$, then $var(s) = X_p$. Each AND node s has a cost $c(s)$ defined to be the product of all factors ψ_α that are instantiated at s but not before.

DEFINITION **2 (solution subtree)** *A* solution subtree *$\hat{x}_M$ is a subtree of $T_\mathcal{T}$ satisfying: (1) it contains the root of $T_\mathcal{T}$; (2) if an OR node is in $\hat{x}_M$, exactly one of its AND child nodes is in $\hat{x}_M$; (3) if an AND node is in $\hat{x}_M$ then all its OR children are in $\hat{x}_M$. The product of arc-costs on any full solution tree equals the cost of a full configuration of the model $\mathcal{M}$.*

Example 1 *Figure 1a is a primal graph of 4 bi-valued variables and 4 binary factors of a graphical model. Figure 1b is a pseudo tree. Figure 1c displays the AND/OR search tree guided by the pseudo tree. A solution subtree is $(B = 1, A = 0, C = 1, D = 0)$ having a cost of 72.*

Each node s in $T_\mathcal{T}$ can be associated with a *value*, $Z(s)$ expressing the conditioned partition function rooted at s. Clearly, $Z(s)$ can be computed recursively based on the values of the children of s: OR nodes by summation and AND nodes by multiplication. The value of $T_\mathcal{T}$, is the value of its root node, which is the partition function of the underlying model, $\mathcal{M}$.

The size of the AND/OR search tree, $T_\mathcal{T}$ is exponential in the height of the pseudo tree. It is possible to merge some subtrees using a concept known as *con-*

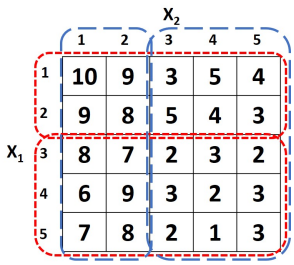


Figure 2: Motivating Example; Z=126.

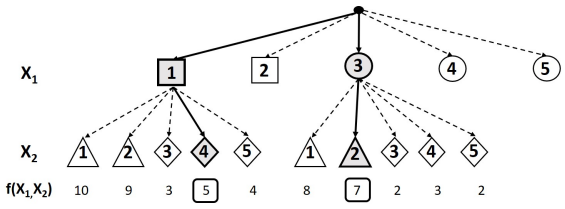


Figure 3: Motivating Example Tree

text (defined later), yielding an AND/OR graph that is exponential in the tree-width of the primal graph [Dechter and Mateescu, 2007]. As noted, $Z(s)$ can be computed by a depth-first search scheme from leaves to root of the AND/OR tree (See Figure 1 where the values attached to each node). When the pseudo tree is a chain we get back the regular OR tree, where each path corresponds to a full variable configuration.

Stratified sampling is a variance reduction technique that can be used with **importance sampling** [Rubinstein and Kroese, 2007] [Liu *et al.*, 2015] [Gogate and Dechter, 2011] to achieve further reduction in variance. Let f be a non-negative function defined on a sample space $\mathbf{X}$. We want to estimate the value $Z = \sum_{\mathbf{x}\in\mathbf{X}} f(\mathbf{x})$. In importance sampling, we can estimate Z by drawing samples from $\mathbf{X}$ according to a proposal distribution q and estimating the equivalent expression $Z = \sum_{\mathbf{x}\in\mathbf{X}} \frac{f(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x})$, leading to an importance sampling estimator $\hat{Z}^I$. In stratified importance sampling, we first divide the sample space $\mathbf{X}$ into k strata of equal area under the distribution q. Let J be a random variable with k values $\{1, .., k\}$ assigning to each configuration in the sample space the index of its strata. For each $j \in J$, we compute importance sampling estimators $\hat{Z}^I_j$ of $Z_j = \sum_{\mathbf{x}\in\mathbf{X}} \frac{f_j(\mathbf{x})}{q_j(\mathbf{x})} q_j(\mathbf{x})$ from samples drawn from the conditional distributions $q_j = q|(J = j)$ and $f_j(\mathbf{x})$ be defined as $f(\mathbf{x})$ if $J(\mathbf{x}) = j$ and 0 otherwise. The stratified importance sampling estimator is $\hat{Z}^{SI} = \frac{1}{k}\sum_{j=1}^{k} \hat{Z}^I_j$. It can be shown

THEOREM **1 ([Rizzo, 2007])** *Let Z_J be a uniform random variable defined on $\{1, ..., k\}$ assigning value Z_j to j. Let $\hat{Z}^{SI}$ be the stratified importance sampling estimator computed using m samples drawn in each of the k strata. Let $\hat{Z}^I$ be the importance sampling estimator computed with $M = mk$ samples. The variance reduction achieved by moving from $\hat{Z}^I$ to $\hat{Z}^{SI}$ is $\frac{k}{m} Var(Z_J)$.*

3 ABSTRACTION SAMPLING

A Motivating Example. Our proposed Abstraction Sampling (AS) algorithm emulates stratified importance sampling on partial configurations (nodes in the search tree) layer by layer (each layer corresponding to a variable). To illustrate the idea consider a small two dimensional function over variables X_1 and X_2 with domains $D_1 = D_2 = \{1, 2, 3, 4, 5\}$ in Figure 2. Partial configurations of length 1 (corresponding to X_1) are partitioned into two abstract states: one where $X_1 \in \{1, 2\}$ and the other where $X_1 \in \{3, 4, 5\}$. Note that this abstraction function tries to put into the same abstract states rows which have roughly the same total mass (sum of entries). At the level of X_2 we have abstract states defined just by variable X_2 - one where $X_2 \in \{1, 2\}$ and the other by $X_2 \in \{3, 4, 5\}$. We assume a uniform proposal distribution; note that the exact answer is $Z = 126$ (summing all entries in the table).

The sampling process is illustrated in Figure 3. Assume we pick $X_1 = 1$ and $X_1 = 3$ as representatives of their respective abstract states, and give them weights 2 and 3, to account for the number of states they represent. All extensions of these 2 assignments to X_2 are generated, with abstract states indicated as diamonds and triangles. There are now 4 candidates for a representative of triangle abstraction with values $\{10, 9, 8, 7\}$ and corresponding weights $\{2, 2, 3, 3\}$, and 6 diamond candidates with values $\{3, 5, 4, 2, 3, 2\}$ and corresponding weights $\{2, 2, 2, 3, 3, 3\}$. We select a representative at random in proportion to their weights. Assume this yields configurations $(X_1 = 3, X_2 = 2)$ and $(X_1 = 1, X_2 = 4)$. We next update the weights of the selected representatives to account for the mass they represent yielding: $w(X_1 = 3, X_2 = 2) \leftarrow w(X_1 = 3, X_2 = 2)/(3/10) = 3/(3/10) = 10$ and $w(X_1 = 1, X_2 = 4) \leftarrow w(X_1 = 1, X_2 = 4)/(2/15) = 2/(2/15) = 15$, yielding an estimate for the partition function $\hat{Z} = 7 \cdot 10 + 5 \cdot 15 = 145$.

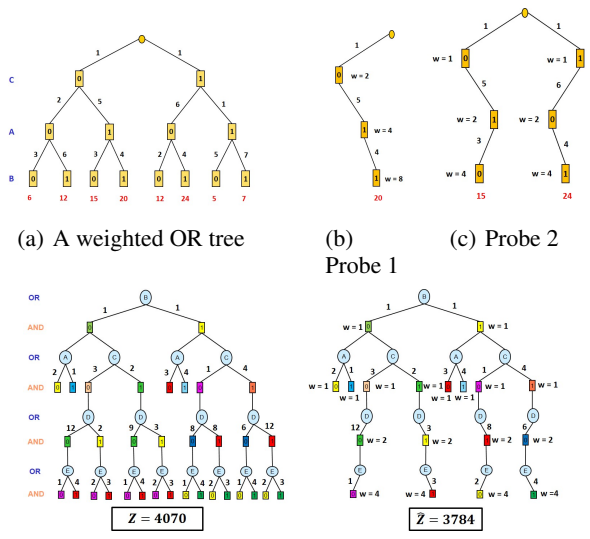
We also compared the empirical variance of our estimator with a simple importance sampler, using 100000 trials and observed variance reduction from 2337 to 398, an almost 6 fold decrease. While in this example the abstract states were not equal size, the abstractions were selected to yield a large variance between the different abstract states.

Algorithm 1: OR Abstraction Sampling(AS) one probe

Require: An implicit OR search tree T of a model $\mathcal{M}$. $c(s, s')$ is the cost of arc (s, s') extracted from $\mathcal{M}$. $g(s)$ is the product of arc-costs from root to s and $h(s)$ (heuristic function) is an upper bound on the partition function defined on nodes based on graphical model $\mathcal{M}$. Abstraction a.

Ensure: Sampled tree $\tilde{T}$. Estimate $\hat{Z} = Z(\tilde{T})$.

1: initialize $\tilde{T} \leftarrow \emptyset$, $\tilde{Z} \leftarrow 0$, $OPEN \leftarrow \{< s_0, 1 >\}$,
2: **while** OPEN is not empty **do**
3: $< s, w(s) > \leftarrow$ remove node in OPEN with smallest a
4: **if** s is a leaf node in T **then**
5: $\hat{Z} \leftarrow \hat{Z} + w(s) \cdot g(s)$
6: **else**
7: **for** each child s' of s **do**
8: $w(s') \leftarrow w(s)$
9: **if** $a(s') = i$ and OPEN contains a node $< s_{\{i\}}, w_{\{i\}} >$ of abstraction $\{i\}$ **then**
10: $p \leftarrow \frac{w(s')g(s')h(s')}{w(s')g(s')h(s')+w_{\{i\}}g(s_{\{i\}})h(s_{\{i\}})}$
11: $w_{\{i\}} \leftarrow \frac{w_{\{i\}}}{(1-p)}$
12: **with** probability p **do**:
13: remove $s_{\{i\}}$ from OPEN
14: $OPEN \leftarrow OPEN \cup \{< s', \frac{w(s')}{p} >\}$
15: add s' to $\tilde{T}$
16: **else**
17: $OPEN \leftarrow OPEN \cup \{< s', w(s') >\}$
18: $\tilde{T} \leftarrow \tilde{T} \cup \{< s, w(s) >\}$



(a) A weighted OR tree (b) Probe 1 (c) Probe 2

(d) Full AND/OR search tree (e) Probe - AND/OR

Figure 4: Example of Probes

3.1 THE ALGORITHM

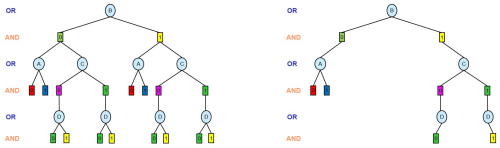
Our proposed Abstraction Sampling algorithm is a Monte Carlo process that generates compact representatives $\tilde{T}$ of T, guided by an *abstraction function*. Abstraction Sampling for OR trees (Algorithm 1) builds a subtree $\tilde{T}$ of T, level-by-level, in a breadth-first manner. Starting from the root of T, at each step, it picks a leaf node having the smallest (as explained next) abstract value, and expands it to its child nodes. Each node s is associated with a weight $w(s)$ representing the mass the abstract state stands for (root weight is 1). For each newly generated node, the algorithm checks if there already exists a node having the same abstraction. If this is the case, (line 9), it decides with some probability p, which of the representative nodes to keep and which to discard. The weight of the selected representative is adjusted to account for the discarded one (step 11). Otherwise, it adds the new node to the frontier of nodes called $OPEN$ with the weight of its parent node (step 17).

Layered Abstractions. Given a weighted directed AND/OR tree T, an abstraction function, $a : T \rightarrow I^+$, where I^+ are integers, partitions the nodes in T, layer by layer, with the requirements that i) if $var(s_1) \neq var(s_2) \rightarrow a(s_1) \neq a(s_2)$, and ii) if $s' \in ch(s)$ then $a(s) < a(s')$ (this enforces breadth-first exploration; $ch(s)$ denotes the children of s in T). Abstraction states are denoted by $\{i\}$ for an integer i.

The Sampling Proposal. We use p proportional to $w(s) \cdot g(s) \cdot h(s)$, where h is a heuristic that provides an upper bound on the partition function (of the subproblem rooted at s). While the algorithm is unbiased for any sampling probability, the heuristic yields a proposal function whose accuracy can significantly impact its convergence, as in any importance sampling scheme.

Example 2 *Consider the (OR) search tree T in Fig. 4a. The cost of each solution is obtained by a product on its solution arcs. In Figure 4b we show a probe generated via an abstraction function that puts all nodes that represent a single variable in a single abstract state. In 4c, we see a probe where each domain value of a variable corresponds to a different abstract state, yielding 2 states per variable, and thus 2 nodes per level of the generated tree. The estimate for (b) is $\hat{Z} = 180$ and for (c) $\hat{Z} = 156$.*

Abstraction Sampling for AND/OR trees requires several modifications. The underlying search tree is an AND/OR tree $T_{\mathcal{T}}$ along a pseudo-tree $\mathcal{T}$, and the abstraction function is defined on AND nodes. The algorithm builds a subtree $\tilde{T}_{\mathcal{T}}$ of $T_{\mathcal{T}}$, level-by-level, breadth first. At each step, it picks a leaf AND node having the smallest abstraction, and expands two levels down - child OR nodes, and their child AND nodes. For newly gen-



(a) A full AND/OR tree with improper abstraction (colors)

(b) AND/OR sample tree generated by improper abstraction

Figure 5: Improper Abstraction for AND/OR

erated AND nodes, the algorithm proceeds as in the OR case to select stochastically the representative node. The estimate is not accumulated during the sampling process. Rather, once a probe $\tilde{T}_{\mathcal{T}}$ is generated, its partition function estimate can be computed in a depth-first manner. Figures 4d and 4e provide an example AND/OR search tree and a possible probe. [1]

3.2 PROPER ABSTRACTIONS

To guarantee the validity and unbiasedness of our sampling scheme for general AND/OR trees, we need to ensure that the sampled AND/OR probe would include only full solution subtrees of the underlying AND/OR tree. For example, the sampled subtree in Figure 5b could have been generated from the tree in Figure 5a with the colors as abstraction, yet it contains a solution tree that corresponds just to a partial configurations (e.g. (B=0, A=0)) and therefore may lead to unbiased estimates. This situation can be avoided if we require abstractions to be *proper*, i.e. any two AND nodes of the same variable, X, can have the same abstract state, only if they are descendant of a common AND node in the sampled AND/OR tree.

DEFINITION **3 (Branching variable, Proper abstraction)** *A variable Y in a pseudo-tree $\mathcal{T}$ is a* branching variable of X, *if Y is X's closest ancestor with at least 2 child nodes. An abstraction function a over AND nodes in $T_{\mathcal{T}}$ is proper if for any two AND nodes n_1 and n_2 having $var(n_1) = var(n_2) = X$, if $a(n_1) = a(n_2)$, then n_1 and n_2 have a common ancestor AND node n_3 s.t. $var(n_3) = Y$ where Y is the branching variable of X.*

Clearly, any OR tree abstraction is proper as there are no branching variables. Abstraction Sampling for AND/OR trees enforces the generation of *proper* probes during the sampling process (for details see AND/OR algorithm in supplementary material). For the remainder of the paper we assume that AS is the general algorithm extended to AND/OR spaces that enforces the proper condition.

[1] Note that while the AND/OR-AS algorithm is defined here as a *breath-first* (BF) traversal of the search space, other formulations are possible. Our AND/OR-AS implementation uses BF traversal on non-branching variables (chains) and *depth-first* (DF) traversal on branching variables, due to superior time/space complexity of DFS algorithms.

4 PROPERTIES

Complexity The *proper* restriction limits compactness of the sampled AND/OR trees. More branchings in the pseudo tree yield more abstract states and consequently larger probes. We can show that:

THEOREM **2 (size and complexity)** *Given a pseudo-tree, the number of states in a probe by AS is $O(n \cdot m^{b+1})$, where n is the number of variables, b bounds the number of branchings along any path of the pseudo-tree and m bounds the maximum number of abstract states in the input abstraction function, a, per variable. Clearly for OR trees, $b = 0$ yielding size bounded by $O(nm)$.*

Notice that when we have many branchings in the pseudo tree $\mathcal{T}$, the underlying AND/OR tree from which we sample is far more compact than the underlying OR tree.

Unbiasedness We can show that our sampling scheme generates an unbiased estimate of the partition function. Detailed proof is in the supplementary materials.

THEOREM **3 (unbiasedness)** *Given a weighted directed AND/OR search tree T derived from a graphical model, the estimate $\hat{Z}$ generated by AS is unbiased.*

5 ON VARIANCE AND ABSTRACTION SELECTION

The proof of unbiasedness works for any sampling distribution p. The reason for choosing our specific proposal probabilities is to reduce the variance. We can show that

THEOREM **4 (exact proposal)** *If the proposal function p in AS uses an exact heuristic $h(n) = Z(n)$, then $\hat{Z}$ has zero variance (single probe is exact), for any abstraction.*

Theorem 4 addresses the extreme case when the proposal is exact. Next we talk about the other extreme when the abstraction is exact.

THEOREM **5 (exact abstraction)** *When the abstraction function satisfies that $a(n) = a(n') \Rightarrow Z(n) = Z(n')$ then, $\hat{Z}$ is exact (i.e. $\hat{Z} = Z$) with one probe, if h satisfies $a(n) = a(n') \Rightarrow h(n) = h(n')$.*

Since AS is a type of stratified importance sampling our **sampling perspective**, (e.g., Theorem 1), suggests several variance reduction ways. The first advises to use

abstractions that at each layer partition nodes into equal area abstract states under the proposal. The second is advising towards increasing the variance of the estimators *between abstract states*. The third encourages having more refined abstractions with more abstract states per layer, as long as the condition of equal size abstract states can be maintained. At the same time, our **search perspective** suggests to always unify nodes that root the same subtrees, whenever such information is available. One possibility is to use the notion of *context*.

The **context** of a variable X in a pseudo-tree $\mathcal{T}$ identifies a subset $C(X)$ of its ancestor variables, whose assignment uniquely determines the AND/OR subtree below the node ([Dechter and Mateescu, 2007]). Therefore, two nodes in the search tree having the same context, (namely, the same assignment along their contexts) root identical subtrees. So, if we use abstractions that are context-isomorph and if h is the mini-bucket heuristic, the two conditions of Theorem 5 hold, yielding:

Corollary 1 *When the abstraction function is context-isomorph, namely, $a(n) = a(n') \leftrightarrow C(n) = C(n')$ and if h is a mini-bucket heuristic, then the partition function estimate, $\hat{Z}$, is exact.*

In the following we construct two abstraction function families based on the notion of context: relaxed context-based and randomized context-based abstractions.

DEFINITION **4 (relaxed context-based abstractions)** *An abstraction a at X is context-based relative to a subset S, $S \subseteq C(X)$, iff for every n_1 and n_2 having $var(n_1) = var(n_2) = X$, we have: $a(n_1) = a(n_2) \leftrightarrow \pi_S C(n_1) = \pi_S C(n_2)$. If $|S| = j$ we say that we use a j-level context-based abstraction. In particular, 0-level abstractions puts all the nodes of a variable in a single abstract state.*

This family of abstract functions will lead to abstract states in each level having the same number of nodes, which can be viewed as a first approximation to abstract states having the same area under the proposal.

Our second family of abstractions introduces randomness into the actual way the abstraction depends on the context. This can facilitate the generation of many abstractions in an automated manner and potentially lead to tighter estimates. We randomly assign nodes into abstract states based on the value of a random hash function, taking into consideration only the context of a node.

DEFINITION **5 (randomized context-based abstraction)** *Let $k \in \mathbb{I}^+$ and $d \in \mathbb{I}^+$ be parameters. Let N be the number of variables in the model and $K = \{1, 2, .., k\}$. We assume that all domains D_i of the variables are subsets of the positive integers. We construct an abstraction a, by first sampling a vector $\mathbf{c} = (c_1, c_2, ..., c_N)$ uniformly at random from K^N. Given a node n_j at level j in the search tree, corresponding to the partial assignment $\bar{\mathbf{x}}_\mathbf{j} = (x_1, x_2, x_3, ..., x_j)$, we compute its hash value $hash(n_j) = \sum_{i=1}^{j} c_i x_i I_{C(X_j)}(X_i)$, where I is the indicator function. We define its abstraction function value $a(n_j) = hash(n_j) \mod d$. The parameter d determines the number of abstract states for each layer of the tree.*

6 EMPIRICAL EVALUATION

6.1 METHODOLOGY

Implementation and Configuration. We evaluate our algorithm on 4 benchmark sets using several configurations of the abstraction sampling algorithm. We implemented the algorithm in C++ and ran experiments on a 2.66 GHz processor with 2GB of memory. For our heuristic we use Weighted Mini-Bucket Elimination (WMBE) [Dechter and Rish, 2003, Liu and Ihler, 2011], whose strength is controlled by a parameter called the i-bound. Higher i-bounds lead to stronger heuristics at the expense of higher computation and memory cost. We use i-bound 10 in our experiments. While there is an interplay between the heuristic strength and the abstraction level, we defer such investigation to future work.

Abstraction Functions. We use the two types of context-based abstractions introduced in the previous section: relaxed context-based (RelCB) and randomized context-based (RandCB). RelCB is parametrized by its level j, while RandCB by its parameter d. We compare the more refined abstractions (higher j or d) to the 0-level abstraction, that combines all nodes in a layer into a single abstract state, corresponding to baseline regular importance sampling. For OR trees, using a fixed variable order, we experiment with abstractions having $j \in \{4, 8\}$ for RelCB, and $d \in \{16, 256\}$ for RandCB.

For AND/OR trees we introduce hybrid abstractions that allow a better control of probe size. A hybrid abstraction with parameter j_k (d_k for RandCB) performs a j-level abstraction (parameter d for RandCB) for nodes having no more than k branching variables in the path to the root, and 0-level abstraction below it. For RelCB abstraction family, we experiment with a pure 1-level abstraction ($j = 1$) and a hybrid abstraction with parameter 2_5. For RndCB family, we experiment with a pure abstraction with $d = 2$ and a hybrid abstraction with parameter 4_5. For randomized abstractions, we tested three different random seeds and observed overall similar results; we present results from one seed selected randomly.

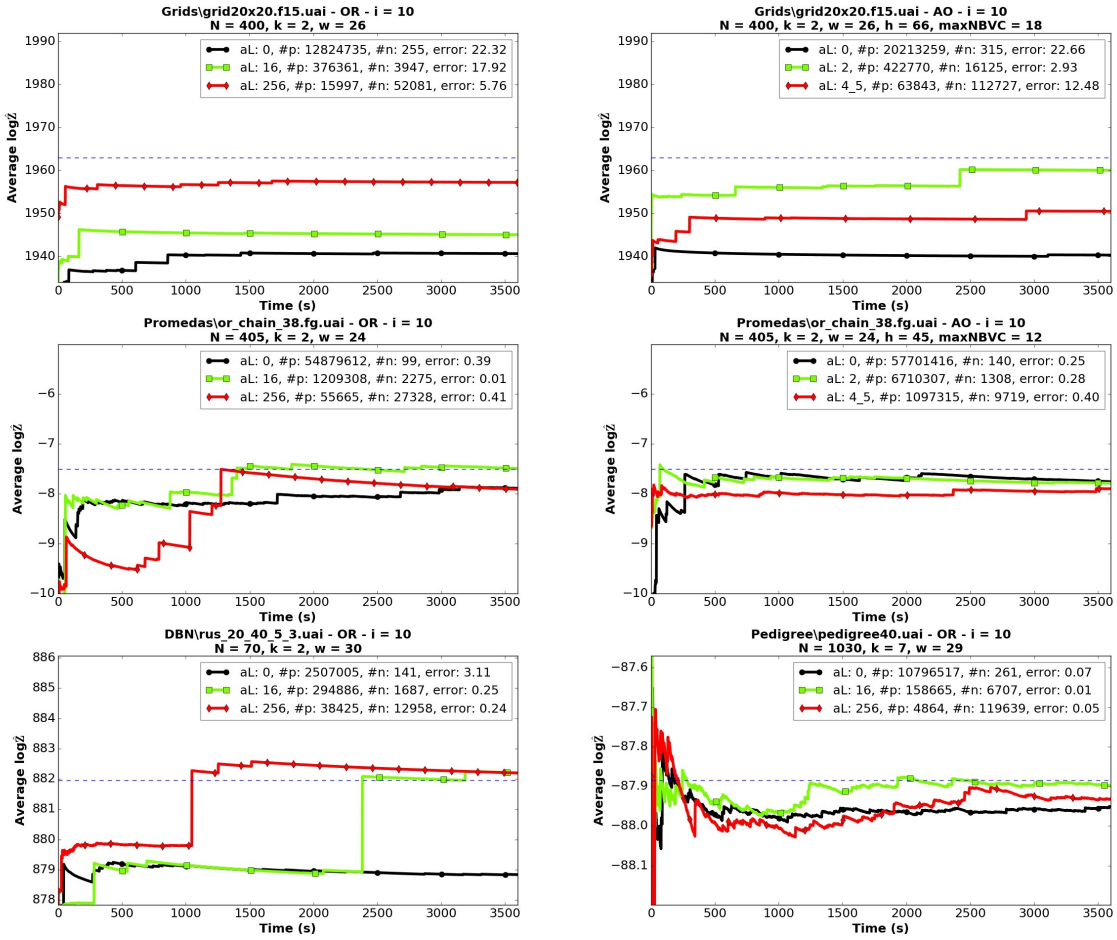


Figure 6: Convergence Plots for Different Abstraction Levels (aL) for Selected Problem Instances. $\#p$ – number of probes, $\#n$ – average number of expanded nodes per probe, h – height of pseudo tree, maxNBVC – max number of branching variables in any path of pseudo tree, error – distance from true value

Benchmarks. We tested on instances from 4 benchmarks (BN, DBN, Pedigree, Promedas). We classify instances as small (we do have the exact value of the partition function) or large (we don't, due to high induced width). We use 130 small and 155 large instances. Summary statistics are in the first column of the results table. Since DBN instances have chain-like pseudo-trees (similar to OR), we test them only on OR trees.

Performance Measure. For each instance and configuration (abstraction family, tree type, abstraction level), we ran the AS algorithm for 1 hour and recorded the partition function estimate $\hat{Z}$ at different times. For small instances (exact Z known), we compute the log partition function absolute error $|\log_{10}\hat{Z} - \log_{10} Z|$. For large instances (exact Z unknown), we use as a proxy the distance of the log of the estimate from the log of an upper bound of the partition Z_{UB} (computed using heuristic function) $|\log_{10} Z_{UB} - \log_{10}\hat{Z}|$. We present the mean of these errors aggregated by benchmark in Table 1.

Comparison with other schemes. We compare our estimates with two state-of-the-art importance sampling algorithms: Weighted Mini-Bucket Importance Sampling (WMB-IS) [Liu *et al.*, 2015] and IJGP-SampleSearch (IJGP-SS) [Gogate and Dechter, 2011]. We use original implementations from the authors and set the common i-bound parameter to 10. For our algorithms and WMB-IS, we use the same fixed variable order, while IJGP-SS computes its own variable order.

The goal of WMB-IS algorithms [Lou *et al.*, 2017a], [Lou *et al.*, 2017b] is to provide (deterministic/probabilistic) upper/lower bounds of the estimate. For that, they use a "mixture wmb-is" proposal yielding bounded importance weights so that known concentration theorems can be applied. This does not necessarily yield optimal convergence. Our focus is to improve the estimate by aiming to reduce the variance. Our approach

(abstraction sampling) should work with any proposal function. We investigated two proposals, 1) multiplicative one (leading to IS weights that are unbounded), 2) "mixture wmb-is" proposal. We observed improved performance with abstraction sampling in both cases, yet the multiplicative proposal yielded faster convergence in most cases, and it is therefore the one that we report.

6.2 RESULTS

The questions addressed by the empirical evaluation are:

- Does Abstraction Sampling result in improved convergence and what insight do we gain on the 2 classes of abstractions?
- What is the impact of tree type (OR vs. AND/OR)?
- Performance of AS across benchmarks?
- Comparison of AS with state-of-the-art?

Individual plots. Figure 6 has convergence plots for selected problem instances for both OR and AND/OR, using the family of randomized context-based abstractions (RandCB). For each abstraction level we plot the anytime estimate of the log partition function. The dashed line represents the ground truth for the log partition function.

In the first row we show OR and AND/OR plots for the same Grids instance. This is an example where moving to higher level abstractions leads to faster performance (AND/OR error is reduced from 22.66 to 2.93). In the second row (a Promedas instance), the 0-level error is already small, so higher level abstractions have about the same performance. In the third row we present two instances, one from DBN and one from Pedigrees and see that higher level abstractions speed up convergence. In the Pedigree instance, all abstraction levels show similar convergence pattern (the error is small).

Aggregate table. Table 1 presents mean errors aggregated over each benchmark for all sampling schemes. This includes our AS algorithm with both types of trees (OR and AND/OR), and both families of abstractions (RelCB and RandCB) and the two competing schemes. For rows corresponding to our scheme, we present errors for 3 abstraction levels: a_0, a_1, a_2, where a_0 is 0-level abstraction and independent of the abstraction family (RelCB or RandCB). a_1 and a_2 correspond to higher level abstractions and their definition varies for OR and AND/OR as described earlier. We show errors at 1 min, 20 min, 60 min. In column three we also show the average number of nodes per probe for each abstraction level. For IJGP-SS, we report results only at the 1 hour mark. An "inf" in the table means that the respective algorithm generates a zero estimate for one or more instances in the benchmark, so the error would be infinity. In parentheses we show the fraction of instances where competing scheme outperforms 0-level abstraction in OR trees.

We see that in both small and large **Grids instances** and for all configurations (OR/AO, RelCB/RandCB), higher level abstractions lead to performance improvement over 0-level abstraction. For grids, randomized (RandCB) abstraction may lead to more significant improvements than RelCB, with AND/OR trees performing even better. For example, after 60 min with OR-RelCB for Grids-small we improve the average error from 4.94 to 3.39, while with OR-RandCB we can improve to 1.41, and AO-RandCB drives down error to 0.84. For both small and large **DBN instances**, only RandCB shows improvement (reducing error from 0.78 to 0.42 for DBN-small, and from 363.93 to 362.88 for DBN-large). For **Pedigree**, the 0-level abstraction is already good, so as expected, higher level abstractions yield little extra benefit. In large **Promedas** instances, RandCB abstraction improved performance in both OR and AND/OR cases, while RelCB was good in the AND/OR case. For small instances, all errors are quite small, still some benefits are gained with the randomized scheme over AND/OR space.

OR vs. AND/OR. AND/OR trees usually lead to improved performance over OR. This is expected since AND/OR search spaces are smaller. This is particularly evident for Promedas (small and large) and for large Grids. For example, in Grids-large going from OR-RandCB to AO-RandCB reduces error from 900.01 to 841.84 after 60 min, for a_1 abstraction. For Grids-small the results are mixed on OR vs AND/OR for benchmarks where errors are already small in the 0-level scheme.

Comparing with state-of-the-art Importance Sampling. In most benchmarks our 0-level importance sampling baseline is competitive and often outperforms competing sampling algorithms. The point to remember is that the abstraction scheme can be augmented on top of any importance sampling scheme.

Discussion. From the analysis of results we gain several valuable insights. **Firstly**, RandCB abstraction consistently demonstrates equal and mostly improved performance compared to the baseline scheme (0-level), while RelCB abstraction is not consistently better. We hypothesize that this difference is due to the ability of RandCB abstractions to generate more uniformly sized abstract states under the proposal. **Secondly**, we observe that in general the AND/OR schemes lead to larger improvements than OR ones. This was not obvious because of the "proper" condition. **Thirdly**, we observe that as expected larger gains are achieved when the baseline importance sampling results in large errors, yet the perfor-

Table 1: Mean Error Aggregated Over Benchmark for a Given Scheme, Time and Abstraction Level (a_0, a_1, a_2) . a_0 is 0-level abstraction, (a_1, a_2) are: OR-RelCB:$(4, 8)$, OR-RandCB:$(16, 256)$, AO-RelCB:$(1, 2\_5)$, AO-RandCB:$(2, 4\_5)$. (#inst, $\bar{n}$, $\bar{w}$, $\bar{k}$, $|\bar{F}|$, $\bar{s}$) are number of instances and averages of number of variables, induced width, max domain size, number of functions, max scope size.

Benchmark #inst, $\bar{n}$, $\bar{w}$, $\bar{k}$, $\lvert\bar{F}\rvert$, $\bar{s}$	**Scheme**	**#nodes per probe** a_0, a_1, a_2	**1 min** a_0, a_1, a_2	**20 min** a_0, a_1, a_2	**60 min** a_0, a_1, a_2
DBN-small	OR-RelCB	141, 1963, 22687	1.18, 1,93, 2.58	0.88, 1.86, 1.77	0.78, 1.43, 1.65
60, 70, 30, 2, 16950, 2	OR-RandCB	141, 1611, 13449	1.18, 1.04, **0.81**	0.88, 0.71, **0.63**	0.78, **0.42**, 0.54
	WMB-IS		9.40	5.69	3.27
	IJGP-SS				1.22
Grids-small	OR-RelCB	180, 2774, 42184	6.68, 5.19, 5.07	6.06, 4.71, 4.25	4.94, 4.31, 3.39
7, 271, 24, 2, 791, 2	OR-RandCB	180, 2755, 34101	6.68, 5.05, **1.97**	6.06, 4.10, **1.55**	4.94, 3.83, 1.41
	AO-RelCB	224, 13388, 91154	5.46, 3.84, 4.70	5.43, 3.68, 3.74	4.83, 2.97, 3.83
	AO-RandCB	224, 9418, 65423	5.46, **1.97**, 4.27	5.43, 1.72, 3.36	4.83, **0.84**, 2.77
	WMB-IS		2.94	1.94	1.21
	IJGP-SS				38.81
Pedigree-small	OR-RelCB	270, 6115, 271925	**0.17**, 0.19, 0.26	0.17, 0.17, 0.19	0.17, 0.17, **0.16**
22, 917, 26, 5, 917, 4	OR-RandCB	270, 4967, 75980	**0.17**, 0.20, 0.25	0.17, 0.17, 0.19	0.17, 0.17, 0.19
	AO-RelCB	294, 10286025, 337777	0.18, 0.47, 0.21	**0.15**, 0.36, 0.17	**0.16**, 0.20, **0.16**
	AO-RandCB	294, 1171192, 92627	0.18, 0.24, 0.18	**0.15**, 0.19, 0.16	**0.16**, 0.18, **0.16**
	WMB-IS		inf (1/22)	inf (3/22)	1.06
	IJGP-SS				11.10
Promedas-small	OR-RelCB	115, 1091, 12801	0.68, 0.77, 1.59	0.33, 0.44, 0.70	0.16, 0.34, 0.47
41, 666, 26, 2, 674, 3	OR-RandCB	115, 2174, 28712	0.69, 0.69, 0.62	0.33, 0.28, 0.38	0.16, 0.15, 0.21
	AO-RelCB	110, 825, 5818	0.56, 0.59, 0.66	0.30, 0.34, 0.40	0.15, 0.23, 0.23
	AO-RandCB	110, 753, 6162	0.56, 0.32, **0.28**	0.30, 0.19, **0.15**	0.15, **0.10**, **0.10**
	WMB-IS		inf (5/41)	1.77	1.15
	IJGP-SS				3.06
DBN-large	OR-RelCB	434, 6586, 91881	366.77, 368.29, 369.59	365.32, 366.49, 367.44	363.93, 365.04, 366.20
48, 216, 78, 2, 66116, 2	OR-RandCB	434, 4858, 71545	366.77, 365.56, **365.14**	365.32, 364.04, **363.53**	363.93, 363.14, **362.88**
	WMB-IS		inf (0/48)	inf (0/48)	inf (0/48)
	IJGP-SS				**356.91**
Grids-large	OR-RelCB	2827, 45112, 719763	966.46, 925.86, 927.60	933.64, 900.71, 909.37	928.35, 889.53, 894.59
19, 3432, 117, 2, 10244, 2	OR-RandCB	2827, 45104, 710675	966.46, 945.98, 918.19	933.64, 912.19, 907.30	928.35, 900.01, 894.15
	AO-RelCB	3326, 5485338, 2849697	949.25, 875.81, 910.60	925.85, 863.23, 892.96	918.74, 854.53, 885.18
	AO-RandCB	3326, 3896561, 2826722	949.25, **860.66**, 885.97	925.85, **845.20**, 876.74	918.74, **841.84**, 871.05
	WMB-IS		inf (6/19)	inf (6/19)	inf (7/19)
	IJGP-SS				inf (0/19)
Promedas-large	OR-RelCB	194, 2092, 25156	inf, inf, inf	30.29, inf, inf	29.54, 30.28, 31.89
88, 962, 48, 2, 974, 3	OR-RandCB	194, 3586, 54901	inf, inf, 30.24	30.29, inf, 29.27	29.54, 29.26, 28.59
	AO-RelCB	158, 1561, 10840	inf, 30.45, 30.55	30.00, 29.31, 29.32	29.06, 28.67, 28.44
	AO-RandCB	158, 1319, 12082	inf, 29.23, **28.97**	30.00, 28.47, **28.06**	29.06, 27.89, **27.66**
	WMB-IS		inf (1/88)	inf (1/88)	inf (2/88)
	IJGP-SS				35.50

mance does not deteriorate when the baseline errors are small, especially with RandCB abstractions. This suggests using abstraction sampling as a robust enhancement to importance sampling schemes, especially in scenarios when finding a good proposal is difficult or computationally prohibitive. **Fourthly**, the results show that our scheme is competitive with state-of-the art schemes. **Finally**, our 0-level scheme outperforms WMB-IS although they both rely on WMB-base proposal, but one is multiplicative and the other is mixture.

7 CONCLUSION

The paper presents Abstraction Sampling, a scheme that augments search with sampling, exploiting the strength of both. The scheme can be viewed as extending stratified importance sampling to search spaces, allowing the generation of sub-trees representing multiple configurations, rather than a set of independent samples. We proved the scheme's correctness (unbiasedness for both OR and AND/OR search spaces), analyzed it complexity, discussed convergence properties and provided an extensive empirical evaluation, showing its potential.

The key question is how to design effective abstraction families. In particular, can we devise abstractions yielding equal partitions under the given proposal function, as suggested by theory. Should we aim at domain dependent abstractions? What level of abstraction refinement is cost-effective? Theory suggests that more refined abstractions are superior. We observed that (higher level) abstractions get more effective as the proposals gets less effective. We would like to investigate the interaction between these two functions. Finally, since AND/OR search spaces are superior overall, can we overcome the proper restriction to allow more flexible exploration of abstraction functions.

Acknowledgement. This work was supported in part by NSF grants IIS-1526842 and IIS-1254071, the US Air Force (Contract FA9453-16-C-0508) and DARPA (Contract W911NF-18-C-0015).

References

[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.

[L. H.S. Lelis and Dechter, 2013] L. Otten L. H.S. Lelis and R. Dechter. Predicting the size of depth-first branch and bound search trees. In *Proceedings of IJCAI-2013*, Bejin, China, 2013.

[Lelis *et al.*, 2014] Levi H. S. Lelis, Lars Otten, and Rina Dechter. Memory-efficient tree size prediction for depth-first search in graphical models. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 2014.

[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.*, 2017a] Qi Lou, Rina Dechter, and Alexander T. Ihler. Anytime anyspace AND/OR search for bounding the partition function. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 860–867, 2017.

[Lou *et al.*, 2017b] 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.

[Marinescu and Dechter, 2009] R. Marinescu and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1457–1491, 2009.

[Pearl, 1984] J. Pearl. *Heuristics: Intelligent Search Strategies*. Addison-Wesley, 1984.

[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.

SUPPLEMENTARY MATERIALS - AND/OR ALGORITHM

Algorithm 2: AOAS, a single probe

Require: A graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$ over $X = \{X_1, ..., X_n\}$, a pseudo-tree $\mathcal{T}$. An implicit AND/OR tree $T_{\mathcal{T}}$ of $\mathcal{M}$. $g(s)$ is the product of arc-costs from root to s and $h(s)$ (heuristic function) An abstraction a. s_0 is the root of the tree.

Ensure: A sampled subtree $\tilde{T}_{\mathcal{T}} = (\tilde{N}, E, C)$ of $T_{\mathcal{T}}$. Each $n \in \tilde{N}$ is a pair $n =< s, w(s) >$ where $w(s)$ is a weight. Note that OR node weight is always 1.

1: **initialize** $\tilde{T}_{\mathcal{T}} \leftarrow \{< s_0, 1 >\}$,
2: **while** $OPEN$ is not empty **do**
3: $< s, w(s) > \leftarrow$ remove smallest a node in OPEN
4: Expand s, generating all its child nodes variables in the pseudo-tree $\{X_1, ...X_r\}$, each yielding OR nodes denoted $s_1, ..., s_r$ ($var(s_j) = X_j$) and add them to $\tilde{T}_{\mathcal{T}}$.
5: **for** each OR child node s_j **do**
6: expand s_j, generating all its AND child nodes $s_{j_i} =< X_j, x_{j_i} >$, $x_{j_i} \in D_{X_j}$ with $w(s_{j_i}) = w(s)$.
7: **for** each child s_{j_i} **do**
8: **if** $\tilde{T}_{\mathcal{T}}$ contains a representative $< s_{\{k\}}, w_{\{k\}} >$ of abstraction $\{k\}$, $a(s_{j_i}) = k$ that shares the same configuration up to its branching variable (i.e., obeys properness) **then**
9: $p \leftarrow \frac{w(s_{j_i})g(s_{j_i})h(s_{j_i})}{w(s_{j_i})g(s_{j_i})h(s_{j_i}) + w_{\{k\}}g(s_{\{k\}})h(s_{\{k\}})}$
10: with probability p do:
11: remove $s_{\{k\}}$ from $\tilde{T}_{\mathcal{T}}$ and OPEN
12: add $< s_{j_i}, \frac{w(s_{j_i})}{p} >$ as a child of s_j in $\tilde{T}_{\mathcal{T}}$ representing $\{k\}$ and add it to OPEN
13: else
14: $w_{\{k\}} \leftarrow \frac{w_{\{k\}}}{1-p}$
15: **else**
16: add $< s_{j_i}, w(s_{j_i}) >$ as a child of s_j in $\tilde{T}_{\mathcal{T}}$ representing $\{k\}$ and add it to OPEN.
17: $\tilde{T}_{\mathcal{T}}$ is the final tree generated.
18: **return** $\hat{Z} \leftarrow$ compute Z of $\tilde{T}$ AOAS-Z-estimator

SUPPLEMENTARY MATERIALS - EXTENDED UNBIASEDNESS PROOF

THEOREM **6 (unbiasedness)** *Given a weighted directed AND/OR search tree T derived from a graphical model, the estimate $\hat{Z}$ generated by AS is unbiased.*

Proof. (sketch) Clearly, for any node in the AND/OR tree the partition function it roots can be expressed recursively by: $Z(n) = \prod_{n' \in ch(n)} \sum_{n'' \in ch(n')} c(n', n'')Z(n')$, $Z(n) = 1$ if n is a leaf AND node. At each step, the algorithm maintains the current, partially generated, AND/OR tree denoted $\tilde{T}^{(t)}$ where t index the algorithm's steps. The partial tree $\tilde{T}^{(t)}$ is a stochastic subtree of T whose nodes are assigned weights by the algorithm.

Algorithm 3: AOAS-Z-estimator

Require: A graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$ over $X = \{X_1, ..., X_n\}$, a pseudo-tree $\mathcal{T}$. An AND/OR tree $T_{\mathcal{T}}$ of $\mathcal{M}$; its subtree $\tilde{T}_{\mathcal{T}} = (\tilde{N}, E, C)$ of $T_{\mathcal{T}}$. $c(s)$ is the cost of an OR-to-AND arc $(parent(s), s)$ in $T_{\mathcal{T}}$.

Ensure: An estimate $\hat{Z}$ of the partition function Z.

1: Compute an estimate for each node in $\tilde{T}_{\mathcal{T}}$, bottom up, with the following rules
2: For leaf node $< s, w(s) >$, its value is $\hat{Z}(s) = w(s)c(s)$.
3: For internal OR node s, its value is $\hat{Z}(s) = \sum_{c \in ch(s)} \hat{Z}(c)$.
4: For internal AND node $< s, w(s) >$, its value is $\hat{Z}(s) = \frac{w(s)}{w(parent(s))} c(s) \prod_{c \in ch(s)} \hat{Z}(c)$.
5: **return** Value of the root node $\hat{Z}(r)$.

Let $OPEN$ be the set of AND leaf nodes of the partial tree $\tilde{T}^{(t)}$ and let $CLOSED$ be the rest of the nodes in $\tilde{T}^{(t)}$. We define an intermediate estimator of Z at step t denoted $\hat{Z}^{(t)}(n)$, over $\tilde{T}^{(t)}$ recursively as follows. For an AND node $n \in \tilde{T}^{(t)}$.

$$\hat{Z}^{(t)}(n) = \begin{cases} Z(n) & \text{if } n \in OPEN \\ \prod_{n' \in ch(n)} \sum_{n'' \in ch(n')} w(n'')c(n', n'')\hat{Z}^{(t)}(n'') & \\ & \text{if } n \in CLOSED \end{cases} \quad (1)$$

This recursive estimate combines information from the sampled nodes and estimated weights in $\tilde{T}^{(t)}$ with exact values of Z for the nodes in OPEN at time t. We can show that at any step t, $E(\hat{Z}^{(t+1)}(r) - \hat{Z}^{(t)}(r)|\tilde{T}^t) = 0$, where r is the root. Consequently the expected value of our successive approximation at the end of sampling is equal to its initial value: $\hat{Z}^{(0)}(r) = Z(r) = Z$. (For more details see supplement.) □

DEFINITION **6 (recursive function on AND/OR trees)** *Given a weighted directed AND/OR tree, having costs, c, labeling its OR to AND arcs. We define a recursive value function denoted Z(n) for an AND node by:*

$$Z(n) = \prod_{n' \in ch(n)} \sum_{n'' \in ch(n')} c(n', n'')Z(n') \quad (2)$$

The initial value for leaves: $Z(n) = 1$ if n is a leaf AND node.

THEOREM **7** *Given a weighted directed AND/OR search tree T derived from a graphical model, and a value function $Z(n)$ defined recursively over T and given a proper abstraction function over T, the estimate generated by AOAS and AOAS-Z-estimator, $\hat{Z}(r)$, is unbiased. Namely $E\hat{Z}(r) = Z(r)$, when r is the dummy root AND node of T.*

Proof. At each step, the algorithm maintains the current, partially generated, AND/OR tree denoted $\tilde{T}^{(t)}$ (we drop the subscript of the pseudo-tree for simplicity), where t index the algorithm's steps in generating the sampled tree. The partial tree $\tilde{T}^{(t)}$ is a stochastic subgraph of T whose nodes are assigned weights by the algorithm. Let $OPEN$ be the set of AND leaf nodes of the partial tree $\tilde{T}^{(t)}$ and let $CLOSED$ be the rest of the nodes in $\tilde{T}^{(t)}$.

We define an intermediate estimator of Z at step t denoted $\hat{Z}^{(t)}(n)$, over $\tilde{T}^{(t)}$ recursively as follows. For an AND node $n \in \tilde{T}^{(t)}$.

$$\hat{Z}^{(t)}(n) = \begin{cases} Z(n) & \text{if } n \in OPEN \\ \prod_{n' \in ch(n)} \sum_{n'' \in ch(n')} w(n'')c(n', n'')\hat{Z}^{(t)}(n'') & \\ & \text{if } n \in CLOSED \end{cases} \quad (3)$$

This recursive estimate combines information from the sampled nodes and estimated weights in $\tilde{T}^{(t)}$ with exact values of Z for the nodes in OPEN at time t.

We will show that at any step t, $E(\hat{Z}^{(t+1)}(r) - \hat{Z}^{(t)}(r)|\tilde{T}^{t}) = 0$. Consequently the expected value of our successive approximation at the end of sampling is equal to its initial value: $\hat{Z}^{(0)}(r) = Z(r) = Z$.

Deterministic changes. The algorithm performs deterministic steps of node expansions. These operations grow $\tilde{T}^{(t)}$ but do not change the value of the estimator at all. According to EQ. (3), when the algorithm performs node expansion, namely expanding an AND node whose current estimate is $Z(n)$ to its children and grandchildren and re-evaluate the resulting estimate at n, we will get back $Z(n)$ because the recursion obeys the recursive definition of $Z(n)$ (see EQ. (2) when $w = 1$, which are the initial weights). So, since the estimate does not change at the leaves of $\tilde{T}^{(t)}$, no change will be propagated up the tree, to the root. In other words in thos cases we need no expectation. We have that: $(\hat{Z}^{(t+1)}(r) - \hat{Z}^{(t)}(r)|\tilde{T}^{t}) = 0$.

Stochastic changes. The only stochastic change occurs when an AND node, u, is examined (step 9) and the algorithm identifies a representative AND node v having the same abstraction in OPEN. We denote by s the first common ancestor of u and v in $\tilde{T}^{(t)}$ through an OR tree. *Since the abstraction is proper*, the subtree of $\tilde{T}^{(t)}$ rooted at s, denoted by $\tilde{T}_s^{(t)}$, is an OR tree. Therefore, there would be no product in the second expression of EQ. (3) and we can see that the estimate at node s can be expressed by a sum over all paths from s to each leaf node in $\tilde{T}_s^{(t)}$. Noting explicitly the leaf nodes u and v we get, from recursing EQ. (3),

$$\hat{Z}^{(t)}(s) = \sum_{\{n \neq u,v | leafs\ in\ \tilde{T}_s^{(t)}\}} \hat{Z}^{(t)}(n) \cdot \prod_{q \in path(s..n)} w(q)c(q) + Z(u)\prod_{q \in path(s..u)} w(q)c(q) + Z(v)\prod_{q \in path(s..v)} w(q)c(q) \quad (4)$$

The first term in EQ. (4) is not affected by the stochastic process. We denote this term by B:

Once node u is processed, the resulting graph $\tilde{T}^{(t+1)}$ depends on the stochastic choice made. If u is selected, (which occurs with probability $1 - p$) we get

$$\hat{Z}^{(t+1)}(s) = B + \frac{w(u)}{1-p} Z(u)c(u) \prod_{q \in path(s..par(u))} w(q)c(q)$$

else, v is selected with probability p then we get

$$\hat{Z}^{(t+1)}(s) = B + \frac{w(u)}{p} Z(v)c(v) \prod_{q \in path(s..par(v))} w(q)c(q)$$

By simple algebraic manipulation it is possible to show that for node s we get: $E(\hat{Z}^{(t+1)}(s) - \hat{Z}^{(t)}(s)|\tilde{T}^{(t)}) = 0$. Since at all the leaf nodes of $\tilde{T}^{(t+1)}$, excluding s and its subtree, $\hat{Z}^{(t+1)}(n) - \hat{Z}^{(t)}(n) = 0$, and since at s, we proved no change in expectation between the successive approximations. We get also at the root $E(\hat{Z}^{(t+1)}(r) - \hat{Z}^{(t)}(r)|\tilde{T}^{(t)}) = 0$. □

SUPPLEMENTARY MATERIALS - FULL EXPERIMENTAL RESULTS

Table 2: Mean Error Aggregated Over Benchmark for a Given Scheme, Time and Abstraction Level (a_0, a_1, a_2) . a_0 is 0-level abstraction, (a_1, a_2) are: OR-RelCB:$(4, 8)$, OR-RandCB:$(16, 256)$, AO-RelCB:$(1, 2\_5)$, AO-RandCB:$(2, 4\_5)$. (#inst, $\bar{n}$, $\bar{w}$, $\bar{k}$, $|\bar{F}|$, $\bar{s}$) are number of instances and averages of number of variables, induced width, max domain size, number of functions, max scope size.

Benchmark #inst, $\bar{n}$, $\bar{w}$, $\bar{k}$, $\|\bar{F}\|$, $\bar{s}$	scheme	#nodes per probe a_0, a_1, a_2	1 min a_0, a_1, a_2	20 min a_0, a_1, a_2	60 min a_0, a_1, a_2
DBN-small	OR-RelCB	141, 1963, 22687	1.18, 1,93, 2.58	0.88, 1.86, 1.77	0.78, 1.43, 1.65
60, 70, 30, 2, 16950, 2	OR-RandCB-1	141, 1611, 13449	1.18, 1.04, 0.81	0.88, 0.71, 0.63	0.78, 0.42, 0.54
	OR-RandCB-2	141, 1624, 12656	1.18, 2.15, 1.77	0.88, 1.42, 1.23	0.78, 1.17, 1.07
	OR-RandCB-3	141, 1684, 14579	1.18, 1.34, 0.84	0.88, 1.05, 0.77	0.78, 0.78, 0.61
	WMB-IS		9.40	5.69	3.27
	IJGP-SS				1.22
Grids-small	OR-RelCB	180, 2774, 42184	6.68, 5.19, 5.07	6.06, 4.71, 4.25	4.94, 4.31, 3.39
7, 271, 24, 2, 791, 2	OR-RandCB-1	180, 2755, 34101	6.68, 5.05, 1.97	6.06, 4.10, 1.55	4.94, 3.83, 1.41
	OR-RandCB-2	180, 2746, 33650	6.68, 4.29, 2.77	6.06, 3.98, 1.93	4.94, 3.27, 2.02
	OR-RandCB-3	180, 2748, 33898	6.68, 4.23, 3.27	6.06, 4.04, 3.38	4.94, 3.34, 2.24
	AO-RelCB	224, 13388, 91154	5.46, 3.84, 4.70	5.43, 3.68, 3.74	4.83, 2.97, 3.83
	AO-RandCB-1	224, 9418, 65423	5.46, 1.97, 4.27	5.43, 1.72, 3.36	4.83, 0.84, 2.77
	AO-RandCB-2	224, 8938, 84428	5.46, 3.16, 3.87	5.43, 3.10, 3.81	4.83, 2.82, 3.48
	AO-RandCB-3	224, 11291, 82649	5.46, 4.28, 3.77	5.43, 3.43, 3.41	4.83, 3.23, 3.50
	WMB-IS		2.94	1.94	1.21
	IJGP-SS				38.81
Pedigree-small	OR-RelCB	270, 6115, 271925	0.17, 0.19, 0.26	0.17, 0.17, 0.19	0.17, 0.17, 0.16
22, 917, 26, 5, 917, 4	OR-RandCB-1	270, 4967, 75980	0.17, 0.20, 0.25	0.17, 0.17, 0.19	0.17, 0.17, 0.19
	OR-RandCB-2	270, 4967, 75841	0.17, 0.20, 0.25	0.17, 0.18, 0.18	0.17, 0.16, 0.16
	OR-RandCB-3	270, 4975, 76055	0.17, 0.19, 0.20	0.17, 0.17, 0.18	0.17, 0.17, 0.16
	AO-RelCB	294, 10286025, 337777	0.18, 0.47, 0.21	0.15, 0.36, 0.17	0.16, 0.20, 0.16
	AO-RandCB-1	294, 1171192, 92627	0.18, 0.24, 0.18	0.15, 0.19, 0.16	0.16, 0.18, 0.16
	AO-RandCB-2	294, 725005, 93194	0.18, 0.20, 0.18	0.15, 0.20, 0.17	0.16, 0.17, 0.16
	AO-RandCB-3	294, 2292328, 82475	0.18, 0.21, 0.18	0.15, 0.18, 0.16	0.16, 0.18, 0.16
	WMB-IS		-	-	1.06
	IJGP-SS				11.10
Promedas-small	OR-RelCB	115, 1091, 12801	0.68, 0.77, 1.59	0.33, 0.44, 0.70	0.16, 0.34, 0.47
41, 666, 26, 2, 674, 3	OR-RandCB-1	115, 2174, 28712	0.69, 0.69, 0.62	0.33, 0.28, 0.38	0.16, 0.15, 0.21
	OR-RandCB-2	115, 2172, 28850	0.68, 0. 64, 0.65	0.33, 0.28, 0.30	0.16, 0.13, 0.21
	OR-RandCB-3	115, 2172, 29017	0.68, 0.59, 0.73	0.33, 0.28, 0.36	0.16, 0.15, 0.19
	AO-RelCB	110, 825, 5818	0.56, 0.59, 0.66	0.30, 0.34, 0.40	0.15, 0.23, 0.23
	AO-RandCB-1	110, 753, 6162	0.56, 0.32, 0.28	0.30, 0.19, 0.15	0.15, 0.10, 0.10
	AO-RandCB-2	110, 769, 6453	0.56, 0.43, 0.39	0.30, 0.17, 0.20	0.15, 0.12, 0.15
	AO-RandCB-3	110, 753, 6218	0.56, 0.36, 0.29	0.30, 0.19, 0.16	0.15, 0.11, 0.10
	WMB-IS		-	1.77	1.15
	IJGP-SS				3.06
DBN-large	OR-RelCB	434, 6586, 91881	366.77, 368.29, 369.59	365.32, 366.49, 367.44	363.93, 365.04, 366.20
48, 216, 78, 2, 66116, 2	OR-RandCB-1	434, 4858, 71545	366.77, 365.56, 365.14	365.32, 364.04, 363.53	363.93, 363.14, 362.88
	OR-RandCB-2	434, 4804, 71036	366.77, 365.58, 364.49	365.32, 364.19, 363.02	363.93, 363.17, 362.53
	OR-RandCB-3	434, 4774, 70421	366.77, 365.70, 364.04	365.32, 363.84, 362.97	363.93, 363.20, 362.36
	WMB-IS		-	-	-
	IJGP-SS				356.91
Grids-large	OR-RelCB	2827, 45112, 719763	966.46, 925.86, 927.60	933.64, 900.71, 909.37	928.35, 889.53, 894.59
19, 3432, 117, 2, 10244, 2	OR-RandCB-1	2827, 45104, 710675	966.46, 945.98, 918.19	933.64, 912.19, 907.30	928.35, 900.01, 894.15
	OR-RandCB-2	2827, 45097, 711566	966.46, 938.20, 917.92	933.64, 904.34, 910.19	928.35, 897.03, 895.12
	OR-RandCB-3	2827, 45100, 709978	966.46, 937.50, 923.23	933.64, 909.52, 915.99	928.35, 898.47, 890.60
	AO-RelCB	3326, 5485338, 2849697	949.25, 875.81, 910.60	925.85, 863.23, 892.96	918.74, 854.53, 885.18
	AO-RandCB-1	3326, 3896561, 2826722	949.25, 860.66, 885.97	925.85, 845.20, 876.74	918.74, 841.84, 871.05
	AO-RandCB-2	3326, 3846042, 2820388	949.25, 853.83, 880.27	925.85, 843.66, 874.03	918.74, 840.39, 868.61
	AO-RandCB-3	3326, 4276589, 2818713	949.25, 865.29, 882.50	925.85, 846.33, 871.89	918.74, 842.33, 865.49
	WMB-IS		-	-	-
	IJGP-SS				-
Promedas-large	OR-RelCB	194, 2092, 25156	-, -, -	30.29, -, -	29.54, 30.28, 31.89
88, 962, 48, 2, 974, 3	OR-RandCB-1	194, 3586, 54901	-, -, 30.24	30.29, -, 29.27	29.54, 29.26, 28.59
	OR-RandCB-2	194, 3587, 54904	-, -, -	30.29, -, 29.36	29.54, 29.47, 28.47
	OR-RandCB-3	194, 3585, 54859	-, -, 30.21	30.29, 30.50, 29.20	29.54, 29.35, 28.55
	AO-RelCB	158, 1561, 10840	-, 30.45, 30.55	30.00, 29.31, 29.32	29.06, 28.67, 28.44
	AO-RandCB-1	158, 1319, 12082	-, 29.23, 28.97	30.00, 28.47, 28.06	29.06, 27.89, 27.66
	AO-RandCB-2	158, 1259, 11381	-, 29.24, 28.81	30.00, 28.56, 28.11	29.06, 27.96, 27.66
	AO-RandCB-3	158, 1377, 11704	-, 29.50, 28.82	30.00, 28.45, 28.07	29.06, 27.83, 27.68
	WMB-IS		-	-	-
	IJGP-SS				35.50