Value-Based Abstraction Functions for Abstraction Sampling

Bobak Pezeshki[1] **Kalev Kask**[1] **Alexander Ihler**[1] **Rina Dechter**[1]

[1]University of California, Irvine

Abstract

Monte Carlo methods are powerful tools for solving problems involving complex probability distributions. Despite their versatility, these methods often suffer from inefficiencies, especially when dealing with rare events. As such, importance sampling emerged as a prominent technique for alleviating these challenges. Recently, a new scheme called Abstraction Sampling was developed that incorporated stratification to importance sampling over graphical models. However, existing work only explored a limited set of abstraction functions that guide stratification. This study introduces three new classes of abstraction functions combined with seven distinct partitioning schemes, resulting in twenty-one new abstraction functions, each motivated by theory and intuition from both search and sampling domains. An extensive empirical analysis on over 400 problems compares these new schemes highlighting several well-performing candidates.

1 INTRODUCTION

The partition function (Z) is an important quantity in probabilistic graphical model inference and is often estimated using Monte Carlo methods such as Importance Sampling (IS) [Rubinstein and Kroese, 2016, Liu et al., 2015, Gogate and Dechter, 2011]. Inspired by the works of Knuth [1975] and Chen [1992], a framework called Abstraction Sampling (AS) [Broka et al., 2018] was introduced extending IS by enabling samples to represent multiple configurations. AS uses concepts from Stratified Sampling [Rubinstein and Kroese, 2016, Rizzo, 2007] and Compact Search [Dechter and Mateescu, 2007, Marinescu and Dechter, 2009] to build a sampled subtree called a *probe* which is then used to compute an estimate. Probes are built level-by-level according to a variable ordering where, at each level, an *abstraction function* groups nodes into *abstract states* from which representative nodes are selected to extend paths in the probe.

Using what are referred to as context-based abstraction functions, Broka et al. [2018] showed competitive performance of AS against IS, Weighted Mini-Bucket Importance Sampling (wMBIS) [Liu et al., 2015, Ihler et al., 2012], and IJGP-SampleSearch (IJGP-ss) [Gogate and Dechter, 2011]. Kask et al. [2020] improved Abstraction Sampling scalability with the AOAS algorithm that more efficiently applied AS to AND/OR search spaces. AOAS showed improved performance, additionally comparing to state-of-the-art scheme Dynamic Importance Sampling (DIS) [Lou et al., 2019].

However, AS development has lacked exploration of diverse and potentially more effective abstraction functions. While Hsiao et al. [2023] proposed using graph neural networks to learn abstraction functions, such methods require learning on a corpus of similar problems before use.

Contributions. This work provides a detailed study of new abstraction schemes for AS. We present a new class of abstractions defined by real-valued functions aimed at capturing relevant similarity features between nodes. Three classes of this new framework are introduced and augmented by seven partitioning strategies. A purely randomized scheme is also introduced. An extensive empirical evaluation is performed on over 400 problems, comparing our novel schemes against: each other, the previous relCB and randCB abstraction functions [Broka et al., 2018, Kask et al., 2020], and implicitly against IS, wMBIS, IJGP-ss, and DIS.

Our experiments identify three schemes in particular that perform significantly better than any previous scheme. Our results demonstrate a significant improvement for one of the most competitive sampling schemes, thus also yielding a substantial computational advance for one of the most challenging tasks in probabilistic inference.

2 GENERAL BACKGROUND

Graphical Models. A *graphical model*, such as a Bayesian or Markov network [Pearl, 1988, Darwiche, 2009, Dechter, 2013], can be defined by a 3-tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$,

where $\mathbf{X}$ is a set of variables, and $\mathbf{D}$ is the set of variable domains, and $\mathbf{F}$ is a set of functions such that each function $f_{\boldsymbol{\alpha}} \in \mathbf{F}$ is defined over a subset of variables $\boldsymbol{\alpha} \subseteq \boldsymbol{X}$ (called the function's scope) capturing local interactions. $\mathcal{M}$ defines a global function, often a factorized probability distribution on $\mathbf{X}$, $P(\mathbf{X}) = \frac{1}{Z}\prod_{\alpha} f_{\alpha}(X_{\alpha})$, where $Z = \sum_{X}\prod_{\alpha} f_{\alpha}(X_{\alpha})$, known as the partition function, is a normalization factor. A *primal graph* $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ of $\mathcal{M}$ associates each variable with a node ($\mathbf{V} = \mathbf{X}$) with edges $e \in \mathbf{E}$ connecting nodes whose variables interact locally, appearing in the scope of the same functions.

Search Spaces of Graphical Models. A graphical model can be transformed into a compact AND/OR search space to leverage conditional independence and facilitate efficient search algorithms [Dechter and Mateescu, 2007].

Given a primal graph $\mathcal{G}$, an AND/OR search space is defined relative to a *pseudo tree* $\mathcal{T} = (\mathbf{V}, \mathbf{E}')$, a directed rooted tree that captures conditional independence encoded in the model. A pseudo tree $\mathcal{T}$ is constructed according to a variable ordering such that every arc of $\mathcal{G}$ not in $\mathbf{E}'$ is a back-arc in $\mathcal{T}$. This construction ensures conditional independence of any variable and its descendants from variables found in the other branches of $\mathcal{T}$ given assignments to their common ancestors. The pseudo tree in Figure 1a was constructed using a variable ordering $o = [A, B, C, D]$. The dashed line shows an edge in the primal graph that is missing from $\mathcal{T}$, but that would be a back-arc if it were present. From its structure we see that variables C and D are independent of B given assignment to A. Here A is referred to as a *branching variable* since it branches to multiple children.

Guided by a pseudo tree $\mathcal{T}$, an *AND/OR search tree* T has alternating levels of OR nodes corresponding to variables and AND nodes corresponding to possible assignments to those variables. Figure 1 shows an AND/OR search tree and its guiding pseudo tree. Note that in the pseudo tree, variables B and C extend to different branches from A. Similarly, in the AND/OR search tree, we see OR nodes B and C extending to different branches under each possible assignment of A. An arc into an AND node n_X of variable X has a cost $c(n_X)$ equal to the product of functions $f_{\boldsymbol{\alpha}} \in \boldsymbol{F}$ such that the path to n_X fully instantiates all $X' \in \boldsymbol{\alpha}$ and such that $X \in \boldsymbol{\alpha}$ [Dechter and Mateescu, 2007].

Notation. Capital letters (X) represent variables and small letters (x) their values. Boldfaced letters represent a collection. Boldfaced capital letters ($\mathbf{X}$) denote a collection of variables, $|\mathbf{X}|$ its cardinality, $D_{\boldsymbol{X}}$ their joint domains (all possible configurations of $\boldsymbol{X}$), and bolded $\boldsymbol{x}$ a particular realization in that joint domain (a particular configuration of $\boldsymbol{X}$).

In the context of search, n is used generally to represent nodes in a search tree. For AND/OR search trees, n_X is used to specifically refer to an AND node associated with variable X, and Y_{n_X} the OR node associated with variable

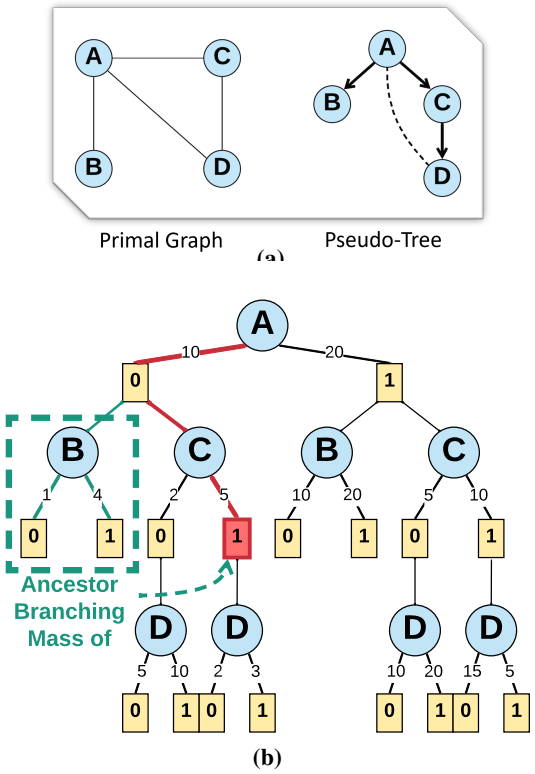

Figure 1: A full AND/OR tree representing 16 possible full configurations of binary variables A, B, C, and D guided by the pseudo tree shown in subfigure (a) above. The path cost for the highlighted node $n_{A=0,C=1}$ at the end of the path $\rightarrow(A{=}0)\rightarrow(C{=}1)$ is $g(n_{A=0,C=1}) = 10{\cdot}5$. The value of the subtree under $n_{A=0,C=1}$ is $Z(n_{A=0,C=1}) = 2{\cdot}1 + 3{\cdot}1$. Boxed in green is the ancestor branching subtree for $n_{A=0,C=1}$ and it has the value $R(n_{A=0,C=1}) = 1{\cdot}1 + 4{\cdot}1$. Thus, $Q(n_{A=0,C=1}) = (10{\cdot}5){\cdot}(1{\cdot}1 + 4{\cdot}1){\cdot}(2{\cdot}1 + 3{\cdot}1)$.

Y that is the child of n_X. $ch(n)$ are the children of node n. $path(n)$ is the configuration of the variables along the path from the root of a search tree T to node n according to assignments corresponding to that path. For the highlighted node n in Figure 1b, $path(n) = \{A{=}0, C{=}1\}$. $varpath(n)$ is the set of variables that $path(n)$ provides a configuration for. In Figure 1b $varpath(n) = \{A, C\}$. The cost of the arc to an AND node n_X is

$$c(n_X) = \prod_{f \in \{f_{\boldsymbol{\alpha}} \in \boldsymbol{F} \mid \boldsymbol{\alpha} \subseteq varpath(n_X),\ X \in \boldsymbol{\alpha}\}} f(path(n_X)). \tag{1}$$

or 1, vacuously. Letting $anc(n)$ be the AND node ancestors of n in the search tree, the cost of $path(n)$ is $g(n) = \prod_{n' \in anc(n)} c(n')$. In Figure 1b, $g(n) = 10 \cdot 5$.

We now define some important quantities involved in evaluating AND/OR search spaces.

***Z*(*n*).** The total cost of the subtree rooted at n. For an AND node n_X with children OR nodes $Y_{n_X} \in ch(n_X)$, $Z(n_X)$ satisfies

$$Z(n_X) = \prod_{Y_{n_X} \in ch(n_X)} Z(Y_{n_X}) \tag{2}$$

such that for OR nodes Y_{n_X}

$$Z(Y_{n_X}) = \sum_{n_Y \in ch(Y_{n_X})} c(n_Y) \cdot Z(n_Y) \tag{3}$$

with $Z(n_X) = 1$ in the case n_X has no children.

Note that given $n_\varnothing$ as the dummy root node of AND/OR tree T, $Z(n_\varnothing) = Z$ of the underlying model $\mathcal{M}$. We denote estimation of $Z(n)$ as $\hat{Z}(n)$. Heuristic estimates of $Z(n)$ are more specifically denoted as $h(n)$.

$R(n)$. On the path from the root of an AND/OR tree T to some node n_X, there may be an intermediate node n_Y associated with branching variable Y in the guiding pseudo tree $\mathcal{T}$. (In Figure 1b, on the path to the highlighted node $n_{A=0,C=1}$, node $n_{A=0}$ is traversed where A is a branching variable in $\mathcal{T}$ of Figure 1a). When this happens, the remaining variables of the model are split between different branches. Thus, the $Z(n)$ of any node down one of the branches will necessarily omit the costs from the configurations of the variables included in the other branch(es). $R(n_X)$, or the *ancestor branching mass*, captures these omitted costs. (In Figure 1b, the green box shows the portion of T corresponding to $R(n_{A=0,C=1})$).

More formally, let $br(n_X)$ be the set of ancestor nodes n_{Y_i} of n_X such that each Y_i is a branching variable ancestor of X in $\mathcal{T}$. We then define $R(n_X)$ simply as:

$$R(n_X) = \prod_{n_Y \in br(n_X)} \prod_{\substack{W_{n_Y} \in ch(n_Y) \\ W_{n_Y} \notin path(n_X)}} Z(W_{n_Y}), \tag{4}$$

(In Figure 1b, $br(n_{A=0,C=1}) = \{n_{A=0}\}$, A being the only branching variable ancestor of C in $\mathcal{T}$, and $B_{n_{A=0}}$ the only respective child OR node ***not*** not on the path to $n_{A=0,C=1}$. Thus, $R(n_{A=0,C=1}) = Z(B_{n_{A=0}})$). We denote approximations to $R(n)$ as $r(n)$.

$Q(n)$. We can now concisely define a quantity $Q(n)$ as the contribution to Z from all full configurations consistent with $path(n)$. In other words, $Q(n)$ is the unnormalized measure of the configuration $path(n)$, with $P(path(n)) = \frac{Q(n)}{Z}$. The quantity $Q(n)$ obeys:

$$Q(n) = g(n) \cdot R(n) \cdot Z(n). \tag{5}$$

Example. In Figure 1b, consider the path from the root to the red node $n_{A=0,C=1}$. Following $n_{A=0}$ to our node, we see OR node $B_{n_{A=0}}$ branches off of the path. So,

$$\begin{aligned} Q(n_{A=0,C=1}) &= g(n_{A=0,C=1}) \cdot R(n_{A=0,C=1}) \cdot Z(n_{A=0,C=1}) \\ &= g(n_{A=0,C=1}) \cdot Z(B_{n_{A=0}}) \quad \cdot Z(n_{A=0,C=1}) \\ &= (10 \cdot 5) \quad \cdot (1 \cdot 1 + 4 \cdot 1) \quad \cdot (2 \cdot 1 + 3 \cdot 1) \end{aligned}$$

Stratified Importance Sampling. Abstraction Sampling builds on Stratified Importance Sampling, which in turn builds on Importance Sampling and Stratified Sampling. *Importance Sampling* is a Monte Carlo scheme used for approximating likelihood queries [Rubinstein and Kroese, 2016, Liu et al., 2015, Gogate and Dechter, 2011]. *Stratified Sampling* is a variance reduction technique for sampling a search space by first dividing it into disjoint strata [Rubinstein and Kroese, 2016]. In *Stratified Importance Sampling*, the sample space is first divided into k strata, then representatives from each strata chosen and re-weighted to represent the omitted members of their respective strata. Rizzo [2007] shows that to reduce overall variance given strata of equal mass under the proposal, the sum of the variances within the strata should be minimized.

Algorithm 1: AOAS Overview

1. **Initialization:** Begin with a dummy root node r.
2. **Probe Generation:** Proceeding in a DFS manner according to a pseudo tree $\mathcal{T}$...
   (a) **Expansion:** Generate children nodes n corresponding to the next variable in the DFS ordering of $\mathcal{T}$. Inherit $w(n)$ from parents and assign appropriate $g(n), h(n)$, and $r(n)$ values.
   (b) **Abstraction:**
      i. **Form Abstract States:** Using $a(\cdot)$, partition newly expanded nodes into abstract states.
      ii. **Select Representative:** Using proposal $p(n) \propto q(n)$, stochastically select a representative from each abstract state and reweigh it such that $w(n) \leftarrow \frac{w(n)}{p(n)}$
   (c) **Backtrack:** After reaching a leaf in $\mathcal{T}$, recursively backtrack until reaching the node that extends to the next unexplored branch of $\mathcal{T}$. While backtracking, update parent node n''s $\hat{Z}(n')$ estimates based on its children's $w(n), g(n)$, and $\hat{Z}(n)$ values.
   (d) **Repeat:** Repeat steps 2a-2c until backtracking to the root node.
3. **Return:** $\hat{Z} = w(r)\,\hat{Z}(r)$ for the root node r.

3 ABSTRACTION SAMPLING

Abstraction Sampling (AS) [Broka et al., 2018] applies concepts of Stratified Importance Sampling to sampling over probabilistic graphical models. AS is guided by an abstraction function $a(\cdot)$ that dictates how nodes are partitioned into *abstract states* (abstract states being analogous to strata in stratified sampling). A search tree is iteratively expanded along a variable ordering. After each expansion, $a(\cdot)$ is used to group nodes into abstract states. Then AS uses an importance-sampling-like process to select a representative from each abstract state and reweights it using importance sampling weights to account for the unselected nodes it represents. The selected nodes are then further expanded and the process iterates. This process yields a weighted sampled subtree of the full search tree T as a sample, referred to as a *probe*. It is important to note that AS probes can contain multiple full configurations, whereas samples from importance sampling are each only a single full configuration.

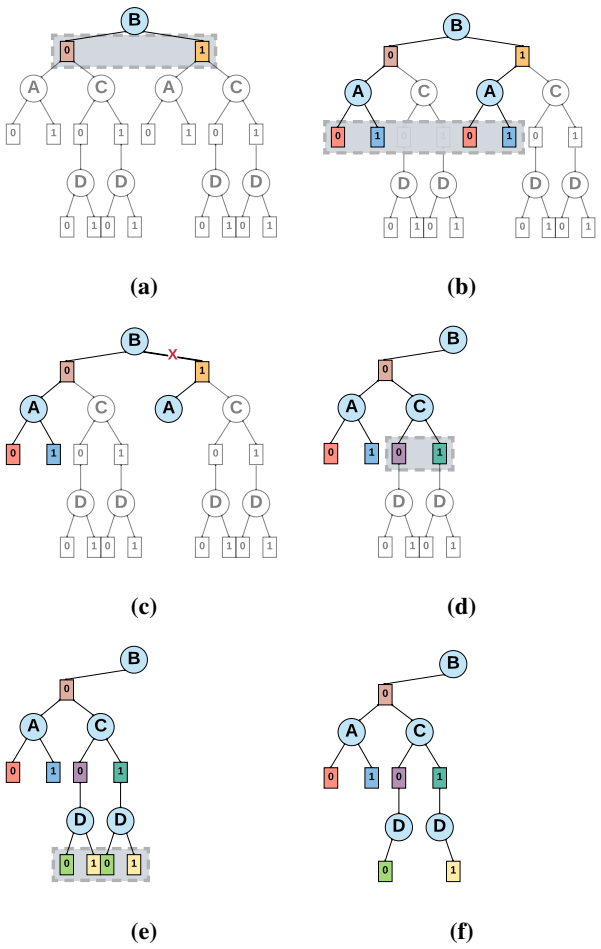

Figure 2: From Kask et al. [2020], a sample trace of AOAS following ordering $B \rightarrow A \rightarrow C \rightarrow D$. Transparent nodes indicate portions of the reachable search space yet to be explored. Gray boxes indicate nodes considered for abstraction. Nodes with the same domain values (also indicated by the same color) are abstracted into the same abstract state. Only one node of each color is stochastically selected as a representative for its respective abstract state. Step (c) shows an optional pruning step. Step (f) shows the final probe capturing four full configurations: B=0, A=0, C=0, D=0, B=0, A=1, C=0, D=0, B=0, A=0, C=1, D=1, $B = 0, A = 1, C = 1, D = 1$.

AOAS. Taking Abstraction Sampling further, Kask et al. [2020] introduced algorithm AOAS that more effectively applied Abstraction Sampling to AND/OR search spaces and significantly improved its performance. AOAS uses a proposal function $p(n) \propto w(n)q(n) = w(n)g(n)h(n)r(n)$ where a weight $w(n)$ accounts for the nodes previously abstracted into the path to n, $g(n)$ is the cost of the path to n, $h(n)$ is a heuristic estimate of $Z(n)$, and $r(n)$ is an estimate of $R(n)$ (see Figure 3). Algorithm 1 provides a high level description of the AOAS procedure. Figure 2 shows a sample trace of AOAS from Kask et al. [2020]. A more detailed version of the algorithm and detailed description of the sample trace can be found in the Supplemental Materials.

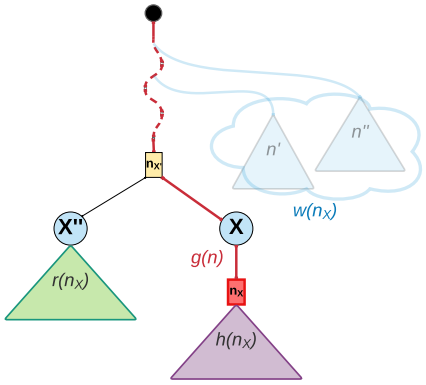

Figure 3: The unnormalized proposal distribution $w(n)q(n)$ visualized to show it considering nodes previously abstracted (via $w(n)$), the ancestor branching mass (via $r(n)$), current path cost (via $g(n)$), and subtree mass (via $h(n)$).

4 VALUE-BASED ABSTRACTIONS

The choice of abstraction function is a crucial aspect of Abstraction Sampling but has only received limited attention so far. The main focus of this work is to identify new abstraction functions that significantly improve AS performance.

Existing State-of-the-Art: Context-Based Abstraction Functions. Broka et al. [2018] designed abstractions based on assignments to a variable's context $C(X)$ - a subset of its ancestors in $\mathcal{T}$ whose assignments uniquely determine the AND/OR subtree below it [Dechter and Mateescu, 2007]. However, the number of configurations to a context is exponential in the context's size. Thus, Broka et al. [2018] and Kask et al. [2020] used *relaxed* context-based (*RelCB*) and *randomized* context-based (*RandCB*) abstractions to control the number of abstract states. RelCB uses a parameter $nCtx$ that groups nodes with the same configuration over the most recent $nCtx-1$ context variables (the relaxed context) into the same abstract state. With a domain size of k, this yields at most k^{nCtx} abstract states at each level. RandCB considers the entire context but bounds the number of abstract states per level based on an $nAbs$ parameter and by using a randomized hashing scheme to associate each full context assignment to one of the $nAbs$ abstract states.

Value-Based Abstractions. We now introduce a new way to form abstractions that we call Value-Based Abstractions. They are defined by (1) a positive real-valued function $\mu : D_{\boldsymbol{X}} \rightarrow \mathbb{R}^+$, where $D_{\boldsymbol{X}}$ is a set of configurations for the variables $\boldsymbol{X}$, and by (2) a partitioning scheme ψ_μ that assigns nodes to abstract states based on their μ value and in an order-consistent manner as defined next.

Definition 4.1 (Value-Ordered Partitioning)
Given $nAbs$ and a function $\mu : D_{\boldsymbol{X}} \rightarrow \mathbb{R}^+$, a partitioning function $\psi_\mu : D_{\boldsymbol{X}} \rightarrow \{A_1, A_2, ...A_{nAbs}\}$, is order-consistent with μ relative to the $nAbs$ abstract states if for any $n_1 \in \boldsymbol{A_i}$ and $n_2 \in \boldsymbol{A_j}$, $i < j \Leftrightarrow \mu(n_1) \leq \mu(n_2)$.

4.1 VALUE-BASED ABSTRACTION CLASSES

We introduce three Value-Based Abstraction classes, each characterized by a unique value function μ that signifies a notion of similarity between nodes. We will subsequently provide partitioning schemes that, together with μ, will yield a set of full abstraction functions.

1. Heuristic-Based Abstractions. Heuristic-Based (HB) abstractions use $\mu(n) = h(n)$, where $h(n)$ is a heuristic estimate of $Z(n)$. Unlike partial or hashed contexts as used by Broka et al. [2018], heuristic estimates of $Z(n)$ can often provide *quantitative* insight into potential similarities of $Z(n)$ values. In particular, this intuition holds when using heuristics that provide bounds on $Z(n)$ such as those via Weighted Mini-Bucket Elimination (wMBE) [Dechter and Rish, 2003, Liu and Ihler, 2011].

2. Heuristic and Ancestral Branching-Based Abstractions. Recall that $r(n)$ is an estimate of n's ancestor branching mass $R(n)$. We can show that:

Theorem 4.1 (AOAS Exact Abstractions)
If an abstraction function $a(\cdot)$ forms abstract states $\boldsymbol{A_i} \in \boldsymbol{A}$ such that $\exists c_i \in \mathbb{R}^+, \forall n \in \boldsymbol{A_i}, \frac{h(n)r(n)}{Z(n)R(n)} = c_i$ whenever $Z(n)R(n) > 0$ (or $h(n)r(n) = 0$ otherwise), then AOAS is exact with its estimates having zero variance.

This observation suggests to use $\mathfrak{hr}(n) = \frac{h(n)r(n)}{Z(n)R(n)}$ as a similarity measure. When nodes having close $\mathfrak{hr}$ values are placed in the same abstract state it can lead to a reduction in variance of the resulting estimate. However, without access to $Z(n)$ or $R(n)$ we cannot evaluate this ratio directly. Instead we use the intuition that grouping based on $h(n)r(n)$ may result in sets of nodes also with similar $Z(n)R(n)$, and thus result in similar $\mathfrak{hr}(n)$. We call such schemes that use $\mu(n) = h(n)r(n)$ HR-Based (HRB) abstractions.

3. Q-Based Abstractions. Another intuition for generating abstractions comes from statistics theory. In his work on stratified Importance Sampling, Rizzo [2007] showed the potential of overall variance reduction by forming strata (abstract states) having equal mass under the proposal distribution and that minimizes the variance within each strata. Thus, since our proposal p is proportional to $w(n)q(n)$, we use $\mu(n) = w(n)q(n) = w(n)g(n)h(n)r(n)$ in what are called Q-based (QB) abstractions.

4.2 ORDERED PARTITIONING SCHEMES

Next we describe seven partitioning schemes ψ to be used with μ to partition the nodes $\boldsymbol{n}$ into abstract states. Together, μ and ψ define a value-based abstraction function.

Running Example. We will use a running example to illustrate the result of using various partitioning schemes.

Assume we have eight nodes with the following $\mu(n)$:

$$1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100 \tag{6}$$

Algorithm 2: $\Psi_{simpleVB}$

$baseCardinality \leftarrow \lfloor \frac{|\boldsymbol{n}|}{nAbs} \rfloor$
$extras \leftarrow |\boldsymbol{n}| \mod nAbs$
$\boldsymbol{n}^* \leftarrow SORT(\boldsymbol{n}, \mu, \text{low-to-high})$
$j_{begin} \leftarrow 1$
foreach $i \leftarrow 1, ..., nAbs$ **do**
  if $extras > 0$ **then**
    $j_{end} \leftarrow j_{begin} + baseCardinality$
    $extras \leftarrow extras - 1$
  else
    $j_{end} \leftarrow j_{begin} + baseCardinality - 1$
  $\boldsymbol{A_i} \leftarrow \{n^*_{j_{begin}}, ..., n^*_{j_{end}}\}$
  $j_{begin} \leftarrow j_{end} + 1$
end
$\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
return $\boldsymbol{A}$

and want to partition the nodes into $nAbs = 4$ abstract states. As we describe each partitioning scheme, we also demonstrate how the scheme would partition these nodes.

1. SimpleVB. The *simpleVB* (simple value-based) scheme groups nodes having similar $\mu(n)$ into the same state by a simple 2-step process: 1) nodes are ordered by $\mu(n)$ (low to high), and 2) nodes are partitioned into abstract states with [approximately] equal cardinality.

Running Example: {1.0, 1.1}, {1.2, 1.3}, {1.4, 1.5}, {10, 100}.

This method leverages speed while still aiming to roughly group nodes with similar $\mu(n)$ together.

2. minVarVB. *minVarVB* uses Ward's Minimum Variance Hierarchical Clustering, also known as Ward's Method [Ward, 1963] (Algorithm 3), to cluster nodes into $nAbs$ abstract states. Use of Ward's method minimizes total within variance of $\mu(\cdot)$ across all abstract states. Ward's Method can be combined with Lance-Williams linear distance updates [Lance and Williams, 1967] to increase efficiency. More details on Ward's Method and Lance-Williams linear distance updates are found in the Supplemental Materials.

Running Example: {1.0, 1.1, 1.2}, {1.3, 1.4, 1.5}, {10}, {100}.

In contrast to *simpleVB*, *minVarVB* places considerable computational resources into computing abstractions by using Ward's Method. Thus *minVarVB* leads to fewer probes being generated but provably forms abstractions that minimize the total within variance of $\mu(n)$ among the abstract states.

3. equalDistVB. In attempt to combine the intuition from *minVarVB* and the speed of *simpleVB*, *equalDistVB* greedily adds nodes in order of μ (low to high) into an abstract state $\boldsymbol{A_i}$ until

$$\mu(\boldsymbol{A_{1,...,i}}) = \sum_{j=1}^{i} \sum_{n \in \boldsymbol{A_j}} \mu(n) \geq \mathcal{Q}_i = \frac{i \cdot \sum_{n' \in \boldsymbol{n}} \mu(n')}{nAbs}, \tag{7}$$

i.e., until the total sum of node values from $\boldsymbol{A_1}, ..., \boldsymbol{A_i}$ reaches or exceeds $\frac{i}{nAbs}$ of the total across all of the nodes

Algorithm 3: Ward's Method

1. **Initialization:** Treat each data point as an individual cluster. Assign each cluster a label.
2. **Compute Pairwise Distances:** Calculate the pairwise distances between all clusters. Various distance metrics can be used, such as Euclidean distance.
3. **Cluster Merging Iteration:**
   (a) Identify the pair of clusters $\boldsymbol{C_i}$ and $\boldsymbol{C_j}$ that, when merged into a new cluster $\boldsymbol{C_{ij}}$, results in the smallest increase in the overall within-cluster variance. This is determined using the formula:
   $$\Delta Var = Var(\boldsymbol{C_{ij}}) - (Var(\boldsymbol{C_i}) + Var(\boldsymbol{C_j}))$$
   where $Var(\boldsymbol{C_{ij}})$ is the variance of the merged cluster, and $Var(\boldsymbol{C_i})$ and $Var(\boldsymbol{C_j})$ are the variances of clusters $\boldsymbol{C_i}$ and $\boldsymbol{C_j}$, respectively.
   (b) Update distance measures between the newly merged cluster and all other clusters.
4. **Repeat:** Repeat steps 2-3 until the desired number of clusters is achieved.

being partitioned. When paired with Q-based abstractions, *equalDistVB* aims to partition nodes into equal mass states under the proposal, motivated by Rizzo [2007].

Running Example: {1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100},{},{},{}.

Although *equalDistVB* hopes to strike a balance between efficiency and low variance of $\mu(n)$ within each abstract state, from the running example we can see it may yield undesirable partitionings for skewed distributions of $\mu(\cdot)$ values. In the example, all of the nodes need to be placed into the first of four abstract states before the sum of their values reaches/exceeds $\frac{1}{4}$ of the total of all nodes being partitioned. Thus, the remaining abstract states end up empty.

4. equalDistVB2. A second version of the equalDist scheme, *equalDistVB2*, follows the same general strategy as *equalDistVB* but uses a reversed sort ordering in attempt to mitigate overfilling of abstract states. Modifying the sort order from low-to-high to high-to-low in Line 1 of Algorithm 4 converts *equalDistVB* to *equalDistVB2*.

Running Example: {100}, {}, {}, {10, 1.5, 1.4, 1.3, 1.2, 1.1, 1.0}

We see that *equalDistVB2* can still over-pack abstract states. The next two variants aim to mitigate this issue further.

5. equalDistVB3. In order to further lessen over-packing and ensure abstract states are not left empty, *equalDistVB3* modifies *equalDistVB2* so that, after processing each abstract state, the next state always has a node added to it by default before checking the abstract state fill condition. Modifying the sort order from low-to-high to high-to-low in Line 1 and $\boldsymbol{A_i} \leftarrow \{\}$ to $\boldsymbol{A_i} \leftarrow \{n^*_j\}; j \leftarrow j+1;$ in Line 4 of Algorithm 4 converts *equalDistVB* to *equalDistVB3*.

Running Example: {100}, {10}, {1.5}, {1.4, 1.3, 1.2, 1.1, 1.0}.

Algorithm 4: $\Psi_{equalDistVB}$

1 $\boldsymbol{n}^* \leftarrow SORT(\boldsymbol{n}, \mu,$ low-to-high$)$
2 $j \leftarrow 1$
3 **foreach** $i \leftarrow 1, ..., nAbs$ **do**
4 $\boldsymbol{A_i} \leftarrow \{\}$
5 **while** $\mu(\boldsymbol{A_{1,...,i}}) < \mathcal{Q}_i$ **do**
6 $\boldsymbol{A_i} \leftarrow A_i \cup \{n^*_j\}$
7 $j \leftarrow j+1$
8 **end**
9 **end**
10 $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
11 **return** $\boldsymbol{A}$

While still very efficient, *equalDistVB3* ensures that the provided $nAbs$ granularity is honored, allowing users better control of the search vs. sampling interpolation possible with Abstraction Sampling.

6. equalDistVB4. The final equalDist variant, *equalDistVB4*, aims for more even partitioning. Before processing each abstract state $\boldsymbol{A_i}$, a new cut-off is determined based the remaining nodes $\boldsymbol{n^*_{rm}}$ and remaining abstract states:

$$\widehat{\mathcal{Q}}_i = \frac{\sum_{n \in \boldsymbol{n^*_{rm}}} \mu(n)}{nAbs - i + 1}. \tag{8}$$

Nodes are added to abstract state $\boldsymbol{A_i}$ while $\mu(\boldsymbol{A_i}) < \widehat{\mathcal{Q}}_i$. Modifying the sort order from low-to-high to high-to-low in Line 1 and $\mu(\boldsymbol{A_{1,...,i}}) < \mathcal{Q}_i$ to $\mu(\boldsymbol{A_i}) < \widehat{\mathcal{Q}}_i$ in Line 5 of Algorithm 4 converts *equalDistVB* to *equalDistVB4*.

Running Example: {100}, {10}, {1.5, 1.4, 1.3}, {1.2, 1.1, 1.0}.

Still computationally efficient, *equalDistVB4* spreads nodes with small values more evenly across abstract states.

7. randVB. It can be beneficial to rely on randomness to ensure a diverse sampling of abstractions. *randVB* does this by sampling $nAbs-1$ partition points uniformly at random and without replacement from between nodes sorted according to $\mu(\cdot)$, and then partitions the nodes accordingly. The resulting abstract states ensure that nodes are still grouped according to $\mu(\cdot)$, but the sizes of those groups vary.

Algorithm 5: Ψ_{randVB}

1 $\boldsymbol{s} \sim Unif(\{\boldsymbol{M} \subseteq \{1, ..., |\boldsymbol{n}| - 1\} \mid |\boldsymbol{M}| = nAbs - 1\})$
2 $\boldsymbol{s^*} \leftarrow SORT(\boldsymbol{s})$
3 $\boldsymbol{n}^* \leftarrow SORT(\boldsymbol{n}, \mu,$ high-to-low$)$
4 $j \leftarrow 1$
5 **foreach** $i \leftarrow 1, ..., nAbs-1$ **do**
6 $\boldsymbol{A_i} \leftarrow \{n^*_j, ..., n^*_{s^*_i}\}$
7 $j \leftarrow s^*_i + 1$
8 **end**
9 $\boldsymbol{A_{nAbs}} = \{n^*_j, ..., n^*_{|n^*|}\}$
10 $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
11 **return** $\boldsymbol{A}$

Running Example: ex1: {100, 10}, {1.5}, {1.4, 1.3, 1.2}, {1.1, 1.0}; ex2: {100}, {10, 1.5, 1.4, 1.3}, {1.2, 1.1}, {1.0}; etc.

Complexity. Assuming $\mu(\cdot)$ is $\mathcal{O}(1)$, each of the proposed partitioning schemes have time complexity $\mathcal{O}(|\boldsymbol{n}| \log |\boldsymbol{n}|)$ and space complexity $\mathcal{O}(|\boldsymbol{n}|)$, with the exception of *minVarVB*, which requires $\mathcal{O}(|\boldsymbol{n}|^2)$ for both. More details can be found in the Supplemental Materials.

5 RANDOM-ONLY ABSTRACTIONS

Another unexplored approach was to use purely randomized abstraction schemes. At first glance, one may not expect such schemes to perform well, but randomization in concert with an informative heuristic and proposal can be beneficial.

Intuition. First, given an informative heuristic, the stochastic selection of a representative node *within* each abstract state using a good proposal function will typically opt for nodes that represent greater mass, which is generally beneficial in importance sampling. Second, the randomness of node assignments to the abstract states enables nodes that may otherwise have little chance of being selected to occasionally have a greater chance of selection, leading to a more diverse distribution of probes.

The simpleRand Scheme. More concisely referred to as RAND, the simpleRand scheme partitions nodes via a 2-step process: 1) nodes first are shuffled to create a uniformly random permutation, and then 2) the nodes are partitioned into (approximately) equal cardinality $nAbs$ abstract states.

Algorithm 6: $\Psi_{simpleRand}$

$baseCardinality \leftarrow \lfloor \frac{|\boldsymbol{n}|}{nAbs} \rfloor$
$extras \leftarrow |\boldsymbol{n}| \mod nAbs$
$\boldsymbol{n^*} \leftarrow RANDOM\_SHUFFLE(\boldsymbol{n})$
$j_{begin} \leftarrow 1$
foreach $i \leftarrow 1, ..., nAbs$ **do**
  if $extras > 0$ **then**
    $j_{end} \leftarrow j_{begin} + baseCardinality$
    $extras \leftarrow extras - 1$
  else
    $j_{end} \leftarrow j_{begin} + baseCardinality - 1$
  $\boldsymbol{A_i} \leftarrow \{n^*_{j_{begin}}, ..., n^*_{j_{end}}\}$
  $j_{begin} \leftarrow j_{end} + 1$
end
$\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
return $\boldsymbol{A}$

Running Example: {1.4, 1.1}, {1.2, 10}, {1.0, 1.3}, {100, 1.5}.

Complexity. Both time and space are $\mathcal{O}(|\boldsymbol{n}|)$.

6 EMPIRICAL EVALUATION

Overview. All combinations of Value-Based Abstraction Classes: Heuristic-Based (**HB**), HR-Based (**HRB**), and Q-Based (**QB**); with each of the Ordered Partitioning Schemes: ***simpleVB***, ***minVarVB***, ***equalDistVB1-4***, and ***randVB***; were tested, resulting in twenty-one value-based abstraction functions. The formerly evaluated context-based (**CTX**) abstraction functions: randCB and relCB were compared against. In addition, the purely random abstraction function, **RAND**, was also included. With the exception of RelCB, each abstraction function uses a hyper parameter, $nAbs$, which bounds the number of abstract states at any level. RelCB instead uses an $nCtx$ parameter that limits the number of context variables used in assigning abstract states. To facilitate comparison, we report RelCB's $nCtx$ parameter instead as an equivalent $nAbs$ parameter assuming a domain size of 2. (For example, if RelCB was run using $nCtx = 6$, we report it with $nAbs = 2^6$). All abstraction functions were tested using the AOAS algorithm [Kask et al., 2020]. All algorithms were implemented in C++. All experiments were run on a 2.66 GHz processor and allotted 8 GB of memory.

Heuristics. To inform the sampling proposal, Weighted Mini-Bucket Elimination (wMBE) [Dechter and Rish, 2003, Liu and Ihler, 2011] – which pairs well with AND/OR search [Mateescu and Dechter, 2005] – is used as a heuristic. The i-bound (**iB**) parameter controls the strength of wMBE, where higher i-bounds generally lead to stronger heuristics, and thus better proposals, at the expense of more computation and memory. We standardize our experiments by using the same i-bound when comparing across algorithms.

Benchmarks. In line with previous work on Abstraction Sampling, we perform experiments on the same set of over 400 problems from five benchmarks: DBN, Grids, Linkage-Type4, Pedigree, and Promedas used by Kask et al. [2020]. We refer to problem instances with known Z values as *Exact*. Larger problems without exact solutions are called *LARGE*. For LARGE problems, estimates from 10hr of AOAS using the RAND - RAND being well performing - are used as the reference Z value. When experimenting

Table 1: Exact Benchmark Statistics. Average statistics for Exact problems. **N**: number of instances, **|X|**: average number of variables, **k**: average of problems' largest domain sizes, **w***: average induced tree-width, **d**: average $\mathcal{T}$ depth.

Benchmark	N	|X|	k	w*	d
DBN	66	67	2	29	30
Grids	8	250	2	22	49
Pedigree	25	690	5	25	89
Promedas	65	612	2	21	62

Table 2: LARGE Benchmark Statistics. Average statistics for LARGE problems. **N**: number of instances, **|X|**: average number of variables, **k**: average of problems' largest domain sizes, **w***: average induced tree-width, **d**: average $\mathcal{T}$ depth.

Benchmark	N	|X|	k	w*	d
DBN	48	216	2	78	78
Grids	19	3432	2	117	220
Linkage-Type4	82	6550	5	45	761
Promedas	173	1194	2	72	114

on Exact problems, algorithms use a small i-bound of 5 (weakening the heuristic estimates) and were given a limited time of 300sec to increase difficulty. For LARGE problems, an i-bound of 10 and time limit of 1200sec are used.

Performance Measure. To evaluate performance, we define error as: $Error = |\log_{10} \hat{Z} - \log_{10} Z^*|$, where $\hat{Z}$ is the estimate obtained from AS and Z^* is the reference Z value. For Exact problems, $Z^* = Z$.

6.1 RESULTS

Summary Comparison. To examine potential of the different methods, we tested each algorithm with a range of $nAbs \in \{1, 4, 16, 64, 256, 512, 1024, 2048\}$. For each $nAbs$ and benchmark, we calculated the average error across problems of the benchmark and identified the $nAbs$ that resulted in the lowest average error. Table 3a focuses on Exact problems and shows this lowest average error and corresponding $nAbs$ for each algorithm. Table 3b shows the corresponding results for LARGE problems on the better performing QB and RAND classes, and the CTX class for comparison. If an algorithm was unable to produce a positive Monte Carlo Z estimate for a problem (denoted "Fail"), the wMBE heuristic bound was used as its Z estimate and error computed accordingly. We highlight the best performing schemes.

Comparison using 100 Samples. To assess the quality of abstraction functions in an implementation-agnostic manner and irrespective of resulting probe-sizes or speed, we conducted experiments using a one-hundred sample limit (**m-100**). Table 4 shows these results on Exact problems for the better performing QB and Rand classes using $nAbs = 256$. $nAbs = 256$ was chosen as (1) it is an intermediate granularity and (2) all schemes produced 100 samples in a reasonable time. We highlight the best performing schemes.

Varying nAbs. Table 5 shows average error for $nAbs \in \{4, 64, 1024\}$ on Exact problems of each benchmark. We focus on the better performing variants of QB: minVarQB, equalDistQB3, equalDistQB4; the purely randomized scheme RAND; and the context-based schemes (CTX) for comparison. In Figure 4 and Figure 5, we also show average error across a wider array of $nAbs$ for minVarQB and equalDistQB4, respectively, the latter also acting as a representative for the profile of equalDistQB3 and RAND.

Time Series Plot. Figure 6 and Figure 7 show time-series results for the better performing QB algorithms, RAND, and CTX schemes on a representative Grids and Promedas problem. Each algorithm was plotted with the $nAbs$ that resulted in the lowest average error for the respective benchmark.

6.2 ANALYSIS

Comparison with Context-Based Schemes. Table 3a shows that there is always a partitioning scheme for HB and HRB that can outperform the best CTX scheme on Exact

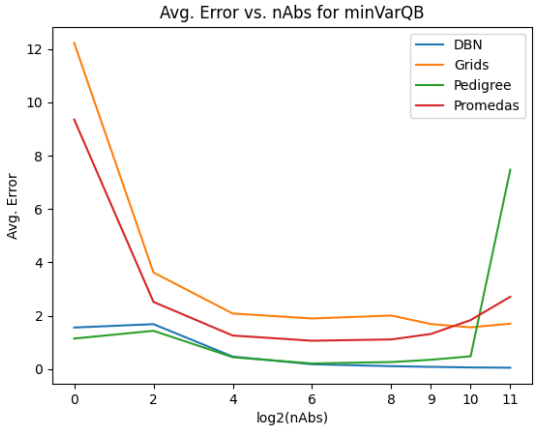

Figure 4: Varying $nAbs$ for minVarQB. Average error on Exact problems using iB-5 and time limit 300 sec for each benchmark at various abstraction granularities (in $\log_2$).

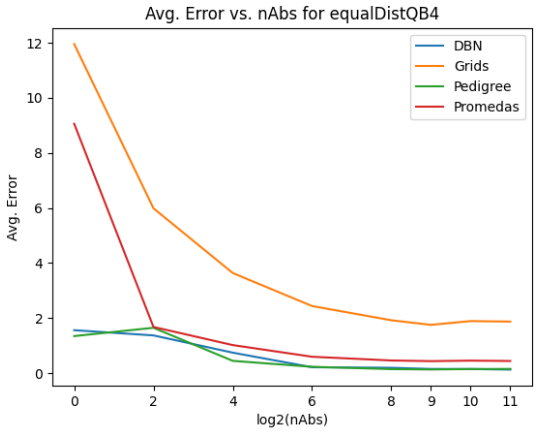

Figure 5: Varying $nAbs$ for equalDistQB4. Average error on Exact problems using iB-5 and time limit 300 sec for each benchmark at various abstraction granularities (in $\log_2$).

problems. For HB, the *simple* and *rand* partitioning schemes perform best, whereas for the HRB class it is more benchmark dependent. QB with *minVar*, *equalDist3*, and *equalDist4* partitioning outperform the CTX schemes across all benchmarks. RAND also consistently outperforms the CTX schemes. Results from Table 3b on LARGE problems agree, with the exception of QB with *minVar* and RAND, which fall slightly shy of randCB's performance on Promedas.

Comparison with Purely Randomized Abstractions. Table 3 shows RAND is a particularly well performing scheme across all benchmarks. However, the QB class using the *equalDist3* and *equalDist4* strategies is consistently comparable or better than the purely randomized scheme. No other scheme does as well.

Comparison with Non Abstraction Sampling Schemes. In prior work by Broka et al. [2018] and Kask et al. [2020], Abstraction Sampling using CTX based abstractions was shown as competitive against several powerful schemes such as Importance Sampling (IS), Weighted Mini-Bucket Importance Sampling (wMBIS) [Liu et al., 2015], IJGP-

iB-5, t-300sec, Exact		DBN			Grids			Pedigree			Promedas		
Class	**Scheme**	**nAbs**	**Fail**	**Avg. Error**	**nAbs**	**Fail**	**Avg. Error**	**nAbs**	**Fail**	**Avg. Error**	**nAbs**	**Fail**	**Avg. Error**
HB	simple	2048	0	0.440	1024	0	2.202	2048	0	0.150	1024	0	0.575
	minVar	1	0	1.361	16	0	3.251	64	0	0.422	16	2	2.509
	equalDist	1	0	1.365	2048	0	10.854	1024	0	0.303	1024	0	2.332
	equalDist2	1	0	1.570	512	0	8.050	1024	0	0.315	64	0	2.123
	equalDist3	1	0	1.489	2048	0	2.764	1024	0	0.279	256	0	2.196
	equalDist4	1024	0	2.819	64	0	6.029	512	0	0.214	2048	0	1.355
	rand	256	0	0.496	2048	0	2.248	2048	0	0.185	2048	0	0.752
HRB	simple	2048	0	0.491	4	0	9.667	256	0	0.225	2048	0	0.705
	minVar	1	0	1.500	64	0	2.319	256	0	0.309	16	1	2.801
	equalDist	1	0	1.305	256	0	10.635	1024	0	0.638	16	4	4.055
	equalDist2	1	0	1.549	2048	0	6.790	16	0	0.457	16	2	3.445
	equalDist3	1	0	1.405	1024	0	2.292	16	0	0.537	16	2	2.656
	equalDist4	1	0	1.511	512	0	1.829	64	0	0.483	2048	0	2.024
	rand	2048	0	0.451	4	0	6.122	64	0	0.666	1024	1	2.165
QB	simple	1	0	1.469	16	0	10.076	256	0	0.297	256	1	3.164
	minVar	2048	0	0.050	1024	0	1.566	64	0	0.210	64	1	1.062
	equalDist	4	0	1.174	2048	0	8.134	2048	0	0.144	2048	0	0.583
	equalDist2	2048	0	0.736	2048	0	4.405	1024	0	0.145	2048	0	0.539
	equalDist3	**2048**	**0**	**0.042**	**2048**	**0**	**1.771**	**512**	**0**	**0.148**	**2048**	**0**	**0.412**
	equalDist4	**2048**	**0**	**0.130**	**512**	**0**	**1.754**	**512**	**0**	**0.134**	**512**	**0**	**0.437**
	rand	1	0	1.295	256	0	6.048	16	0	0.740	16	2	5.988
CTX	rand	4	0	1.381	4	0	5.030	16	0	0.540	1024	1	2.442
	rel	1	0	1.472	64	0	4.021	64	0	0.424	64	6	4.349
RAND	**rand**	**2048**	**0**	**0.104**	**1024**	**0**	**1.501**	**1024**	**0**	**0.143**	**1024**	**0**	**0.513**

(a)

iB-10, t-1200sec, LARGE		DBN			Grids			Linkage-Type4			Promedas		
Class	**Scheme**	**nAbs**	**Fail**	**Avg. Error**	**nAbs**	**Fail**	**Avg. Error**	**nAbs**	**Fail**	**Avg. Error**	**nAbs**	**Fail**	**Avg. Error**
QB	simple	1	0	6.540	16	0	197.931	2048	13	48.681	4	34	11.919
	minVar	2048	0	1.837	1024	0	28.423	256	31	93.058	16	13	5.403
	equalDist	512	0	5.423	2048	0	118.547	2048	22	46.196	512	15	5.960
	equalDist2	2048	0	3.813	2048	0	91.994	1024	21	40.310	2048	12	4.982
	equalDist3	**2048**	**0**	**1.645**	**2048**	**0**	**19.277**	**1024**	**20**	**37.490**	**256**	**5**	**2.560**
	equalDist4	**2048**	**0**	**1.643**	**2048**	**0**	**18.866**	**2048**	**16**	**30.512**	**512**	**5**	**2.476**
	rand	4	0	6.292	16	0	163.973	256	17	156.992	4	28	11.532
CTX	rand	64	0	5.710	512	0	111.104	2048	53	194.741	256	0	3.222
	rel	1	0	6.267	1024	0	80.633	1024	37	129.189	16	34	11.247
RAND	**rand**	**2048**	**0**	**2.123**	**2048**	**0**	**19.053**	**1024**	**19**	**33.804**	**1024**	**10**	**3.936**

(b)

Table 3: Summary Comparison. Each table shows the Abstraction Class (*Class*), Partitioning Scheme (*Scheme*), bound on the number of abstract states per level (*nAbs*), number of problems for which a positive solution could not be estimated (*Fail*), and average $\log_{10} Z$ error (*Avg. Error*) across Exact problems (subtable (a)) and LARGE problems (subtable (b)) in each benchmark. Color bars visualize error magnitudes. We hightliht the best performing algorithms: those for which: (1) difference in total average error (summed across the benchmarks) with respect to the best such total was less than 15% of the best, and (2) within each individual benchmark, the difference in average error with respect to the best average error was less than 35% of the best. (An exception to the latter criterion was granted to Exact DBN, on which the best average error from equalDistQB3 was unusually low).

iB-5, m-100, Exact			DBN		Grids		Pedigree		Promedas	
Class	Scheme	nAbs	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error
QB	simpleQB	256	0	5.350	0	17.406	0	1.059	14	9.659
	minVarQB	**256**	**0**	**0.111**	**0**	**1.911**	**0**	**0.223**	**1**	**1.634**
	equalDist	256	0	5.619	0	15.533	1	0.858	13	5.420
	equalDist2	256	0	2.319	0	11.220	0	0.563	6	3.479
	equalDist3	**256**	**0**	**0.173**	**0**	**3.615**	**0**	**0.206**	**1**	**1.473**
	equalDist4	**256**	**0**	**0.277**	**0**	**2.305**	**0**	**0.180**	**1**	**1.373**
	randQB	256	0	4.982	0	12.653	0	3.211	13	19.441
CTX	rand	256	0	3.587	0	9.568	2	4.695	3	14.386
	rel	256	0	5.265	0	8.013	0	1.097	36	10.845
RAND	**rand**	**256**	**0**	**0.288**	**0**	**2.464**	**0**	**0.325**	**3**	**2.570**

Table 4: 100-Sample Comparison. For abstraction granularity of $nAbs = 256$, aggregated statistics (as described in Table 3) for Exact problems of each benchmark with each algorithm allotted 100 samples.

iB-5, t-300sec, Exact			DBN		Grids		Pedigree		Promedas	
Class	Scheme	nAbs	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error
QB	minVar	4	0	1.684	0	3.622	0	1.434	2	2.518
		64	0	0.180	0	1.897	0	0.210	1	1.062
		1024	0	0.060	0	1.566	0	0.479	2	1.837
	equalDist3	4	0	1.594	0	5.861	0	1.668	1	1.804
		64	0	0.236	0	2.570	0	0.221	0	0.570
		1024	0	0.051	0	1.844	0	0.155	0	0.462
	equalDist4	4	0	1.371	0	5.988	0	1.648	1	1.678
		64	0	0.215	0	2.438	0	0.231	0	0.596
		1024	0	0.150	0	1.891	0	0.150	0	0.455
CTX	rand	4	0	1.381	0	5.030	0	1.852	7	4.643
		64	0	1.763	0	5.950	0	0.598	1	2.659
		1024	0	2.007	0	5.513	0	1.114	1	2.442
	rel	4	0	1.850	0	5.933	0	1.332	10	5.729
		64	0	3.510	0	4.021	0	0.424	6	4.349
		1024	0	5.086	0	5.136	0	1.041	15	6.688
RAND	rand	4	0	1.018	0	4.329	0	1.705	2	2.947
		64	0	0.418	0	2.094	0	0.212	0	0.757
		1024	0	0.120	0	1.501	0	0.143	0	0.513

Table 5: Varying nAbs. Average error when using $nAbs \in \{4, 64, 1024\}$ for minVarQB, equalDistQB3, equalDistQB4, the CTX based algorithms, and RAND, each with iB-5 and time limit of 300 sec.

SampleSearch (IJGP-ss) [Gogate and Dechter, 2011], and Dynamic Importance Sampling [Lou et al., 2019]. Thus, superior performance against CTX schemes implicitly indicates competitiveness against the these other methods.

Abstraction Quality of the QB Schemes. When drawing an equal number of samples with the same abstraction granularity of $nAbs = 256$ (Table 4), QB with *equalDist3* and *equalDist4* and RAND are well performing as seen when

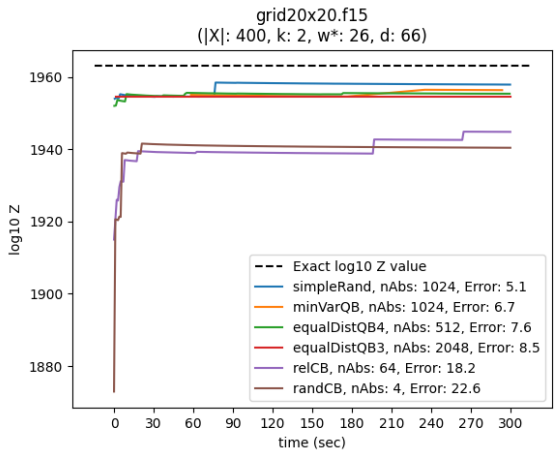

Figure 6: Z estimates from various algorithms versus time on Exact Grids problem grid20x20.f15 using $iB = 5$. The dashed black line shows the true Z value.

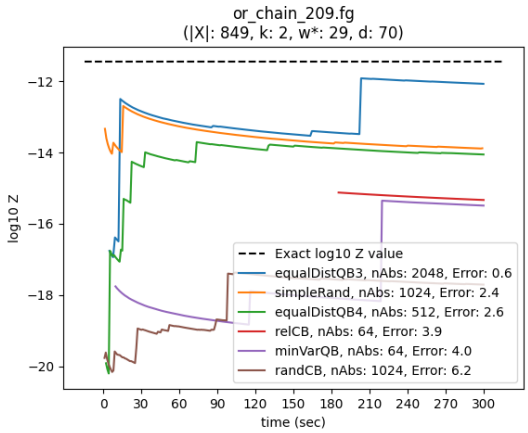

Figure 7: Z estimates from various algorithms versus time on Exact Promedas problem or_chain_209.fg using $iB = 5$. The dashed black line shows the true Z value.

using a time limit (Table 3). A key difference is that QB with *minVar*, which had showed slightly worse performance under a time limit, is now best. This in part explains the success of QB *equalDist3* and *equalDist4*, which try to emulate QB *minVar* while using faster greedy strategies.

Anytime Behavior. Figure 6 and Figure 7 show that Abstraction Sampling estimates continue to improve as time progresses. We also notice that estimates are often underestimates that increase over time, a common phenomenon of importance sampling due to the proposal distribution's tails.

Choice of Abstraction Granularity. From Table 5 we see that for the well performing QB *equalDist3* and *equalDist4* schemes and for the RAND scheme there is a trend that greater $nAbs$ improves performance. Figure 5 further supports this for QB with *equalDist4*, for which plots of QB *equalDist3* and RAND have similar profiles (omitted for brevity). However in Figure 4 and Table 5 we see that for *minVar* error begins to increase when $nAbs$ becomes too high. This can be explained by the higher computational cost of forming *minVar* abstractions (which is more time consuming), leaving less time for probe generation.

	HB	HRB	QB
simple	2.75	1.12	0.72
minVar	1.05	1.13	2.95
equalDist	0.75	0.59	1.16
equalDist2	0.84	0.75	1.82
equalDist3	1.20	1.01	4.05
equalDist4	0.87	1.14	3.90
rand	2.41	0.93	0.60

Figure 8: Performance Matrix. Relative average performance of value-based schemes vs. existing context-based abstractions. Values > 1.00 indicate superior performance.

Summary of Results. Experiments show the QB scheme with *equalDist3* or *equalDistQB4* and RAND performing the best of the newly proposed abstraction functions, significantly outperforming the former state-of-the-art (Figure 8). These schemes tend to improve as the abstraction granularity $nAbs$ increases up to a point, past which we see little difference in performance. Thus, our study suggests that these three abstraction schemes should be the first choice when using AOAS, and be used with the largest $nAbs$ feasible.

7 CONCLUSION

This exploration of abstraction functions for use with AND/OR Abstraction Sampling (AS) featured a new value-based abstraction framework, introducing three abstraction classes: HB, QB, and HRB each defined by real-valued functions that aim to capture informative elements from search and sampling to guide abstractions and improve AS performance. Each class was tested with each of seven node partitioning schemes to form twenty-one new abstraction functions. Additionally, a new purely randomized abstraction scheme, RAND, was presented that places nodes into equal cardinality abstract states completely at random.

Results from an extensive empirical evaluation on over 400 benchmark problems show two of the QB based schemes (*equalDistQB3*, and *equalDistQB4*) and the RAND scheme having superior performance consistently and throughout all benchmarks. In particular, performance was significantly improved relative to former state-of-the-art context-based abstractions, and thus also implicitly against Importance Sampling, Weighted Mini-Bucket Importance Sampling, IJGP-SampleSearch, and Dynamic Importance Sampling.

Based on this study and earlier findings, we believe that AOAS is one of the best schemes for estimating the partition function to date. Future work will explore adjusting the abstraction schemes to problem instances through learning and also the potential for applying adaptive sampling.

Acknowledgements

Thank you to the reviewers for their valuable comments and suggestions. This work was supported in part by NSF grants IIS-2008516 and CNS-2321786.

References

Filjor Broka, Rina Dechter, Alexander. 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. URL http://auai.org/uai2018/proceedings/papers/234.pdf.

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

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

Rina Dechter. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. doi: 10.2200/S00529ED1V01Y201308AIM023. URL http://dx.doi.org/10.2200/S00529ED1V01Y201308AIM023.

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

Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *J. ACM*, 50(2):107–153, 2003. doi: 10.1145/636865.636866. URL http://doi.acm.org/10.1145/636865.636866.

Vibhav Gogate and Rina Dechter. Samplesearch: Importance sampling in presence of determinism. *Artif. Intell.*, 175(2):694–729, 2011. doi: 10.1016/j.artint.2010.10.009. URL https://doi.org/10.1016/j.artint.2010.10.009.

Vincent Hsiao, Dana Nau, and Rina Dechter. Graph neural networks for dynamic abstraction sampling. In *AAAI Workshop on Graphs and More Complex Structures for Learning and Reasoning (GCLR)*, 2023.

Alexander Ihler, Natalia Flerova, Rina Dechter, and Lars Otten. Join-graph based cost-shifting schemes. In Nando de Freitas and Kevin P. Murphy, editors, *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, pages 397–406. AUAI Press, 2012. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2302&proceeding_id=28.

Kalev Kask, Bobak Pezeshki, Filjor Broka, Alexander Ihler, and Rina Dechter. Scaling up and/or abstraction sampling. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4266–4274. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/589. URL https://doi.org/10.24963/ijcai.2020/589. Main track.

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

G. N. Lance and W. T. Williams. A General Theory of Classificatory Sorting Strategies: 1. Hierarchical Systems. *The Computer Journal*, 9(4):373–380, 02 1967. ISSN 0010-4620. doi: 10.1093/comjnl/9.4.373. URL https://doi.org/10.1093/comjnl/9.4.373.

Qiang Liu and Alexander. 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.

Qiang Liu, John W Fisher III, and Alexander 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. Curran Associates, Inc., 2015.

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

R. Marinescu and Rina Dechter. Memory intensive AND/OR search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1492–1524, 2009.

Robert Mateescu and Rina Dechter. The relationship between and/or search and variable elimination. pages 380–387, 01 2005.

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

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

Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method*. Wiley Publishing, 3rd edition, 2016. ISBN 1118632168.

Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. ISSN 01621459. URL http://www.jstor.org/stable/2282967.