Value-Based Abstraction Functions for Abstraction Sampling (Supplemental Materials)

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

[1]University of California, Irvine

Abstract

For revised supplemental materials, please visit `https://ics.uci.edu/~dechter/publications.html`. This document includes supplemental background, descriptions, details, and results in extension to the main paper. Given its size, we suggest using the table of contents to navigate. For an additional background on graphical models, AND/OR search trees, and variable elimination, please view the EXTENDED BACKGROUND supplemental document.

CONTENTS

1 AOAS BACKGROUND

Taken with permission directly from Kask et al. [2020].

1.1 SAMPLE ALGORITHM TRACE

Here we show a trace of abstraction sampling using the AOAS algorithm using an abstraction function that groups AND nodes of the same domain value together in an abstract state.

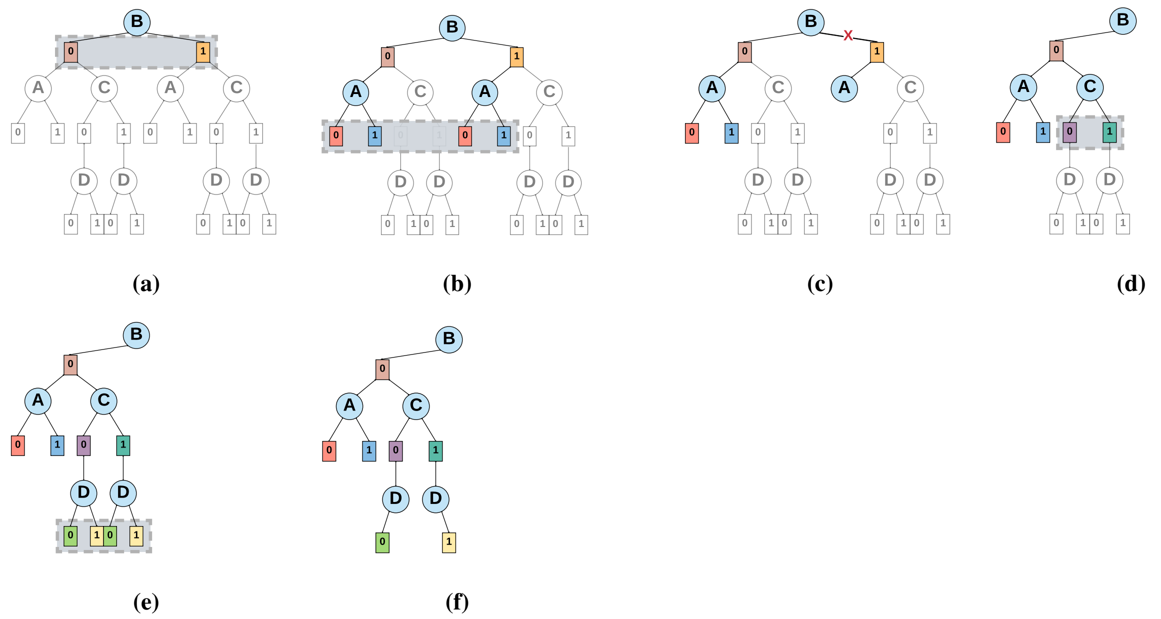


Figure 1: From Kask et al. [2020], a sample trace of AOAS following DFS ordering $B \rightarrow A \rightarrow C \rightarrow D$. Transparent nodes indicate portions of the reachable search space yet to be explored. Gray boxes indicate nodes considered for abstraction. Nodes with the same domain values (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 optional pruning step. Step (f) shows the final example probe capturing four full configurations: $B = 0, A = 0, C = 0, D = 0$, $B = 0, A = 1, C = 0, D = 0$, $B = 0, A = 0, C = 1, D = 1$, $B = 0, A = 1, C = 1, D = 1$.

Starting with variable B (Figure 1a), each node belongs to a different abstraction and is therefore kept. Next, we expand to A and abstract across its nodes (Figure 1b). Not restricted to *proper* abstractions, we partition across *all* nodes of A, regardless of whether they fall under B=0 or B=1. We see two nodes in each abstract state (denoted by the red and blue coloring). Next we calculate their respective proposals (line 21). Note that the proposal of each node n relies on $r(n)$ (line 15), which captures the values of the nodes in its $Out(path(n))$, in this case nodes of C. Since the nodes of C have not been expanded yet, we use their heuristic values as an approximation of their values. We then stochastically choose a representative from each abstract state (line 23). Suppose that both red and blue representatives are stochastically chosen from under B=0 (Figure 1c). Since A has no descendant, we backtrack to B, updating its node values (line 33) and performing a pruning step (line 31). In pruning, we remove AND nodes of B that do not extend to AND nodes of A, and thus prune B=1 (denoted by the red "X" in Figure 1c), in order to ensure formation of proper AND/OR probes. Finally, we expand and abstract C and D (Figures 1d-1f). The $r(n)$ for D's nodes is inherited from the $r(n_C)$ of its respective n_C parent. We backtrack from D to the root updating values (no further pruning was necessary). The result is a valid probe (Figure 1f) containing four solutions: (B=0, A=0, C=0, D=0), (B=0, A=0, C=1, D=1), (B=0, A=1, C=0, D=0), and (B=0, A=1, C=1, D=1). We estimate the partition function by computing $\hat{Z}(B)$.

1.2 DETAILED ALGORITHM

Algorithm 1: AOAS.

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

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

```
Function AOAS ($\mathcal{T}$, $h$, $a$)
begin
    $PROBE \leftarrow n_D, g(n_D), w(n_D), r(n_D), \hat{Z}(n_D) \leftarrow 1$
    $STACK \leftarrow push($empty stack, $D)$
    while $STACK$ is not empty do
        $X \leftarrow top(STACK)$
        if $X$ has unvisited children in $\mathcal{T}$ then
            $Y \leftarrow$ the next unvisited child of $X$
            foreach $n_X \in PROBE$ do
                $PROBE \leftarrow PROBE$ expanded from $n_X$ to $Y$
                $F'_Y \leftarrow$ newly added AND nodes of $Y \in PROBE$
                foreach $n_Y \in F'_Y$ do
                    $w(n_Y) \leftarrow w(n_X)$
                    $g(n_Y) \leftarrow g(n_X) \cdot c(n_Y)$
                    $r(n_Y) \leftarrow r(n_X) \cdot \prod_{\{S \neq Y \in ch_{\mathcal{T}}(X)\}} \hat{V}(S_{n_X})$
                end
            end
            $A \leftarrow \{A_i \mid A_i = \{n_Y \in PROBE \mid a(n) = i\}\}$
            foreach $A_i \in A$ do
                foreach $n \in A_i$ do
                    $p(n) \leftarrow \frac{w(n) \cdot g(n) \cdot h(n) \cdot r(n)}{\sum_{m \in A_i} w(m) \cdot g(m) \cdot h(m) \cdot r(m)}$
                end
                $n_{Y_i} \propto_p A_i$ ;                                  // randomly select
                $w(n_{Y_i}) \leftarrow w(n_{Y_i})/p(n_{Y_i})$
                $\hat{Z}(n_{Y_i}) \leftarrow 1$
                $PROBE \leftarrow PROBE \setminus A_i \cup \{n_{Y_i}\}$
            end
            $push(STACK, Y)$
        else
            $pop(STACK),\ W \leftarrow top(STACK)$
            $PROBE \leftarrow PROBE$ s.t. all $n_W$ without descendants are pruned
            foreach $n_W$ in $PROBE$ do
                $\hat{Z}(n_W) \leftarrow \hat{Z}(n_W) \cdot \sum_{n_X \leftarrow child(n_W)} \hat{Z}(n_X) \cdot c(n_X) \cdot \frac{w(n_X)}{w(n_W)}$
            end
            if $X = D$ then $\hat{Z}_{\mathcal{M}} = \hat{Z}(D)$;
        end
    end
    return $\hat{Z}_{\mathcal{M}}$
end
```

2 PROBE SIZE VARIABILITY

Even with the same abstraction function and granularity (ie. allowed number of abstract states per level), probe sizes can vary greatly. One reason for this is due to abstractions causing nodes from certain branches of the probe to replaced by representative from other branch, and thus the current branch will no longer be extended. We provide a paired example in Figure 3 and Figure 4 where in both cases the probes are constructed according to the pseudo tree shown in Figure 2, an abstraction function is used that groups nodes with the same domain value together (indicated by yellow coloring for grouping of nodes with a domain value of 0 and blue coloring grouping nodes together that have domain value of 1) is used, and the abstraction granularity is set to $nAbs = 2$ (meaning that nodes are abstracted into at most two abstract states).

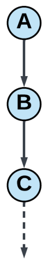
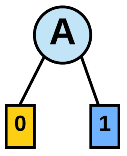
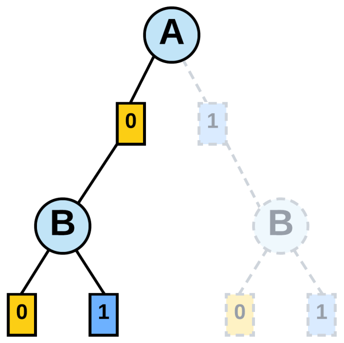


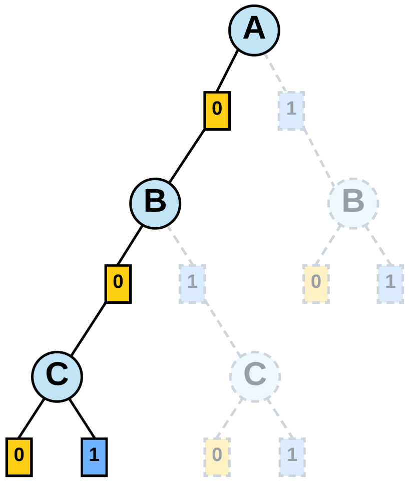
Figure 2: A linear psuedo tree.



(a) Variable A is expanded. Each node is placed into a separate abstract state and each is selected to represent their respective abstract state.

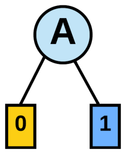


(b) Variable B is expanded from each existing node of A. B nodes with domain value 0 are joined together into an abstract state (yellow); B nodes with domain value 1 constitute a different abstract state (blue). For each resulting abstract state, the corresponding node underneath the branch of $A \leftarrow 0$ is stochastically selected as the representative. As there are no selected representatives underneath the branch of $A \leftarrow 1$, those nodes will no longer be extended (and can be pruned).

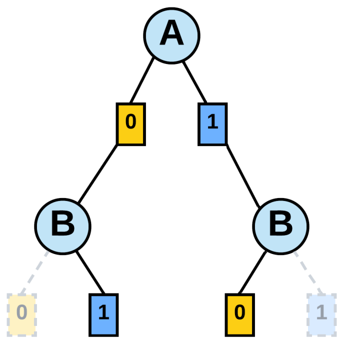


(c) Variable C is expanded from each representative node of B. C nodes with domain value 0 are joined together into an abstract state (yellow); C nodes with domain value 1 constitute a different abstract state (blue). For each resulting abstract state, the corresponding node underneath the branch of $A \leftarrow 0, B \leftarrow 0$ is stochastically selected as the representative. As there are no selected representatives underneath the branch of $A \leftarrow 0, B \leftarrow 1$, those nodes will no longer be extended (and can be pruned).

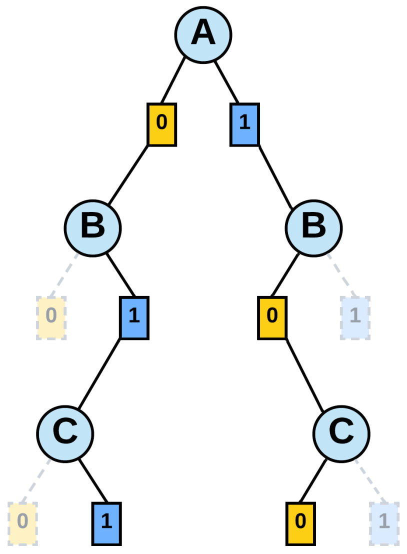
Figure 3: An example of a "skewed" probe construction following the pseudo tree in Figure 2, using an abstraction function that groups nodes of the same domain value into the same abstract state, and using a granularity of $nAbs = 2$. At each level, representatives of all abstract states are chosen under the same single branch, thus only extending only one path in the probe.



(a) Variable A is expanded. Each node is placed into a separate abstract state and each is selected to represent their respective abstract state.



(b) Variable B is expanded from each existing node of A. B nodes with domain value 0 are joined together into an abstract state (yellow); B nodes with domain value 1 constitute a different abstract state (blue). The stochastically selected representative from the $B = 1$ abstract state ends up under the $A \leftarrow 0$ branch while the representative from the $B = 0$ abstract state is selected from under $A \leftarrow 1$. As a result, both $A \leftarrow 0$ and $A \leftarrow 1$ branches have an extension to a node from B and will continue to be extended.



(c) Variable C is expanded from each existing node of B. C nodes with domain value 0 are joined together into an abstract state (yellow); C nodes with domain value 1 constitute a different abstract state (blue). The stochastically selected representative from the $C = 1$ abstract state ends up under the $A \leftarrow 0, B \leftarrow 1$ branch while the representative from the $C = 0$ abstract state is selected from under $A \leftarrow 1, B \leftarrow 0$. As a result, both $A \leftarrow 0, B \leftarrow 1$ and $A \leftarrow 1, B \leftarrow 0$ branches have an extension to a node from C and will continue to be extended.

Figure 4: An example of a "balanced" probe construction following the pseudo tree in Figure 2, using an abstraction function that groups nodes of the same domain value into the same abstract state, and using a granularity of $nAbs = 2$. At each level, representatives of all abstract states are chosen under the same single branch, thus only extending only one path in the probe.

3 EXACT ABSTRACTION PROOFS

Required Definitions.

Definition 3.0.0.1 (Abstraction Function $h(n)$ vs. $Z(n)$ Proportionality)
An abstraction function $a(n)$ maintains $h(n)$ vs. $Z(n)$ proportionality if, for every abstract state A_i formed by $a(n)$, $\forall n \in A_i, h(n) = \alpha\, Z(n)$, for some constant α specific to A_i.

Definition 3.0.0.2 (Abstraction Function $h(n)r(n)$ vs. $Z(n)R(n)$ Proportionality)
An abstraction function $a(n)$ maintains $h(n)r(n)$ vs. $Z(n)R(n)$ proportionality if, for every abstract state A_i formed by $a(n)$, $\forall n \in A_i, h(n)r(n) = \alpha\, Z(n)R(n)$, for some constant α specific to A_i.

Definition 3.0.0.3 (Exact Abstraction Function)
An abstraction function $a(.)$ is exact for an abstraction sampling algorithm, AS, if use of $a(.)$ with AS always leads to AS estimates having zero variance and $\hat{Z} = Z$ for every AS probe.

3.1 ORAS

Theorem 3.1.0.1 (ORAS Exact Abstractions from $h(n)$ vs. $Z(n)$ Proportionality)
If an abstraction function $a(.)$ maintains $h(n)$ vs. $Z(n)$ Proportionality, then it is an exact abstraction function for ORAS.

Proof. We know that if we were to use exhaustive search, we would arrive at the true Z value. We use a proof by induction that assumes that after each abstraction step we will compute the rest of the probe exactly using exhaustive search. Thus, if abstractions are performed layer by layer down from the root, after each abstraction we know that $Z(n')$ will be computed exactly for the selected node n'.

We denote the estimate that would be generated by a probe constructed after t time steps as $\hat{Z}^{(t)}(PROBE)$. (As we will describe, each time step will correspond to an abstraction step). As a base case, $\hat{Z}^{(t=0)}(PROBE) = Z$ since all values will be computed exactly via exhaustive search. In the inductive step, we will show that after each time step t, if instead of using exhaustive search immediately, we first perform an abstraction on the current level of the probe, the resulting estimate of the newly abstracted probe $\hat{Z}^{(t+1)}(PROBE)$ will remain unchanged. Namely, we will show that

$$\hat{Z}^{(t)}(PROBE) - \hat{Z}^{(t+1)}(PROBE) = 0$$

This shows that the abstractions maintain exactness of the probe's estimate.

Starting from the left hand side

$$LHS = \hat{Z}^{(t)}(PROBE) - \hat{Z}^{(t+1)}(PROBE)$$

We note the difference in the overall probe estimates during an Abstraction Sampling is due to the change in the probe estimate that results from each individual abstraction step (namely selection and reweighing of a representative node n' from an abstract state A_i). Thus for our time steps, we will focus on the difference in value resulting from a single arbitrary abstraction step.

$$= \sum_{n \in A_i} w^{(t)}(n) g(n) Z(n) - w^{(t+1)}(n') g(n') Z(n')$$

Above, the left term shows the contribution to the partition function due to nodes of abstract state A_i (still assuming we will perform exhaustive search below each one), and the right term is the contribution of a selected node n' after abstraction (note the adjustment to the selected node's weight).

Using the fact that $w^{(t+1)}(n') = \frac{w^{(t)}(n')}{p(n')}$ (from the importance weight modification), we now get

$$= \sum_{n \in A_i} w^{(t)}(n) g(n) Z(n) - \frac{w^{(t)}(n')}{p(n')} g(n') Z(n')$$

(Note that $p(n')$ cannot be zero, otherwise n' would not have been selected).

Noting that for $p(n') = \frac{w^{(t)}(n')g(n')h(n')}{\sum_{n\in A_i} w^{(t)}(n)g(n)h(n)}$ and substituting we get

$$= \sum_{n\in A_i} w^{(t)}(n)g(n)Z(n)$$

$$- w^{(t)}(n')g(n')Z(n')\frac{\sum_{n\in A_i} w^{(t)}(n)g(n)h(n)}{w^{(t)}(n')g(n')h(n')}$$

$$= \sum_{n\in A_i} w^{(t)}(n)g(n)Z(n) - \frac{Z(n')}{h(n')}\sum_{n\in A_i} w^{(t)}(n)g(n)h(n)$$

Now, per our assumption, $\forall n \in A_i$, let $h(n) = \alpha\, Z(n)$, where α is the proportionality constant by which $h(n)$ differs from $Z(n)$. Then

$$= \sum_{n\in A_i} w^{(t)}(n)g(n)Z(n) - \frac{Z(n')}{\alpha\, Z(n')}\sum_{n\in A_i} w^{(t)}(n)g(n)\,\alpha\, Z(n)$$

$$= \sum_{n\in A_i} w^{(t)}(n)g(n)Z(n) - \frac{\alpha}{\alpha}\sum_{n\in A_i} w^{(t)}(n)g(n)Z(n)$$

$$= \sum_{n\in A_i} w^{(t)}(n)g(n)Z(n) - \sum_{n\in A_i} w^{(t)}(n)g(n)Z(n)$$

$$= 0 = RHS$$

□ □

3.2 AOAS

Theorem 3.2.0.1 (AOAS Exact Abstractions from $h(n)r(n)$ vs. $Z(n)R(n)$ Proportionality)
If an abstraction function $a(.)$ maintains $h(n)r(n)$ vs. $Z(n)R(n)$ Proportionality, then it is an exact abstraction function for AOAS.

Proof. We know that if we were to use exhaustive search, we would arrive at the true Z value. We use a proof by induction that assumes that after each abstraction step we will compute the rest of the probe exactly using exhaustive search. Thus, if abstractions are performed layer by layer down from the root, after each abstraction we know that $Z(n')$ will be computed exactly for the selected node n'. We also assume that, $R(n)$ for every node will be computed exactly. This assumption holds true before we perform any abstractions (as everything is computed exactly via exhaustive search) and continues to hold if we can show that, after each abstraction step, the resulting estimates remains unchanged (and thus remains exact).

We denote the estimate that would be generated by a probe constructed after t time steps as $\hat{Z}^{(t)}(PROBE)$. (As we will describe, each time step will correspond to an abstraction step). As a base case, $\hat{Z}^{(t=0)}(PROBE) = Z$ since all values will be computed exactly via exhaustive search. In the inductive step, we will show that after each time step t, if instead of using exhaustive search immediately, we first perform an abstraction on the current level of the probe, the resulting estimate of the newly abstracted probe $\hat{Z}^{(t+1)}(PROBE)$ will remain unchanged. Namely, we will show that

$$\hat{Z}^{(t)}(PROBE) - \hat{Z}^{(t+1)}(PROBE) = 0$$

This shows that the abstractions maintain exactness of the probe's estimate.

Starting from the left hand side

$$LHS = \hat{Z}^{(t)}(PROBE) - \hat{Z}^{(t+1)}(PROBE)$$

We note the difference in the overall probe estimates during an Abstraction Sampling is due to the change in the probe estimate that results from each individual abstraction step (namely due to the selection and reweighing of a representative node n' from an abstract state A_i). Thus for our time steps, we will focus on the difference in value resulting from a single arbitrary abstraction step.

$$= \sum_{n\in A_i} w^{(t)}(n)g(n)Z(n)R(n) - w^{(t+1)}(n')g(n')Z(n')R(n')$$

Above, the left term shows the contribution to the partition function due to nodes of abstract state A_i (still assuming we will perform exhaustive search below each one), and the right term is the contribution of a selected node n' after abstraction (note the adjustment to the selected node's weight).

Using the fact that $w^{(t+1)}(n') = \frac{w^{(t)}(n')}{p(n')}$ (from the importance weight modification), we now get

$$= \sum_{n \in A_i} w^{(t)}(n)g(n)Z(n)R(n) - \frac{w^{(t)}(n')}{p(n')} g(n')Z(n')R(n')$$

(Note that $p(n')$ cannot be zero, otherwise n' would not have been selected).

Noting that for $p(n') = \frac{w^{(t)}(n')g(n')h(n')r(n')}{\sum_{n \in A_i} w^{(t)}(n)g(n)h(n)r(n')}$ and substituting we get

$$\begin{aligned}
&= \sum_{n \in A_i} w^{(t)}(n)g(n)Z(n)R(n) \\
&\quad - w^{(t)}(n')g(n')Z(n')R(n') \frac{\sum_{n \in A_i} w^{(t)}(n)g(n)h(n)r(n)}{w^{(t)}(n')g(n')h(n')r(n')} \\
&= \sum_{n \in A_i} w^{(t)}(n)g(n)Z(n)R(n) \\
&\quad - \frac{Z(n')R(n')}{h(n')r(n')} \sum_{n \in A_i} w^{(t)}(n)g(n)h(n)r(n)
\end{aligned}$$

Now, per our assumption, $\forall n \in A_i$, let $h(n)r(n) = \alpha\, Z(n)R(n)$, where α is the proportionality constant by which $h(n)r(n)$ differs from $Z(n)R(n)$. Then

$$\begin{aligned}
&= \sum_{n \in A_i} w^{(t)}(n)g(n)Z(n)R(n) \\
&\quad - \frac{Z(n')R(n')}{\alpha\, Z(n')R(n')} \sum_{n \in A_i} w^{(t)}(n)g(n)\, \alpha\, Z(n)R(n) \\
&= \sum_{n \in A_i} w^{(t)}(n)g(n)Z(n)R(n) \\
&\quad - \frac{\alpha}{\alpha} \sum_{n \in A_i} w^{(t)}(n)g(n)Z(n)R(n) \\
&= \sum_{n \in A_i} w^{(t)}(n)g(n)Z(n)R(n) - \sum_{n \in A_i} w^{(t)}(n)g(n)Z(n)R(n)
\end{aligned}$$

$$= 0 = RHS$$

$\square$ $\square$

4 PARADIGMS INTUITING ABSTRACTION STRATEGIES

Next we review concepts from search and sampling that offer paradigms from which we draw ideas for abstraction functions.

4.1 SEARCH PARADIGMS

In [tree] search, one can merge nodes that have the same value to produce a more efficient graph search Mateescu et al. [2008]. Abstraction functions by Broka et al. [2018] focused on this paradigm and approached it by using the concept of a node's context - the assignments to the smallest subset of a node's ancestor variables that dictates its value. Due to the potentially large context size for variables, and consequently the exponentially high number of combinations of assignments to the context, the full context of variables could not be used in most cases. Broka et al. [2018] resolved this by creating two context-based abstraction functions that were relaxed to allow nodes with different contexts to be grouped in the same abstract state. However, sharing the same partial context does not necessarily imply the same, nor even similar, node values. Our new Heuristic-Based abstractions hope to provide more accurate abstractions based on the same ideology.

4.2 SAMPLING PARADIGMS

Consider wanting to compute the $\mathbb{E}_{p^*}[f(x)] = \sum_x f(x)p^*(x)$ for a distribution $p^*(.)$ over a variable X that is difficult to sample from but easy to evaluate, and given a positive value function $f(x)$. Using a proposal distribution $p(.)$ that is easy to sample from, and noticing the equivalency of the target quantity with $\sum_x \frac{f(x)p^*(x)}{p(x)}p(x)$, we can be estimate the quantity by importance sampling by drawing m samples to estimate the equivalent quantity $\mathbb{E}_p[f(x)\frac{p^*(x)}{p(x)}] \approx \frac{1}{m}\sum_{j=1}^m f(x^{(j)})\frac{p^*(x^{(j)})}{p(x^{(j)})}, x^{(j)} \overset{\text{iid}}{\sim} p$. it is well known that importance sampling achieves zero variance when 1) $p(x) = 0 \implies p^*(x) = 0$, and 2) otherwise $p(x)$ is proportional to $p^*(x)f(x)$ Kahn and Marshall [1953], Owen [2013].

Lemma 4.2.0.1 (Importance Sampling Exact Proposal Based on Proportionality with Target Distribution)
Given a distribution $p^*(.)$ over a variable X that is easy to evaluate, and given a positive value function $f(x)$, importance sampling to estimate $\mathbb{E}_{p^*}[f(x)]$ achieves zero variance when using a proposal function $p(.)$ such that 1) $p(n) = 0 \implies p^*(n)f(n) = 0$, and 2) $p(n) \propto p^*(n)f(n)$, otherwise.

Note that we can also use importance sampling to simply compute $\sum_x f(x) = \sum_x \frac{f(x)}{p(x)}p(x) = \mathbb{E}_p[\frac{f(x)}{p(x)}] \approx \frac{1}{m}\sum_{j=1}^m \frac{f(n^{(j)})}{p(x^{(j)})}, x^{(j)} \overset{\text{iid}}{\sim} p$. Note that the partition function over a graphical model, $Z = \sum_{\boldsymbol{x}} \boldsymbol{F}(\boldsymbol{x}), \boldsymbol{F}(\boldsymbol{x}) = \prod_{f \in \boldsymbol{F}} f(x)$, has the form of this task.

In fact, expanding an AND/OR search tree level-by-level, the partition function Z with respect to the nodes n at any variable X can be written as $Z = \sum_n g(n)Z(n)R(n)$. Thus, using a proposal $p(.)$ to perform importance sampling at any level we could instead estimate

$$Z = \sum_n g(n)Z(n)R(n) = \sum_x \frac{g(n)Z(n)R(n)}{p(n)}p(n) \tag{1}$$

$$\approx \frac{1}{m}\sum_{j=1}^m \frac{g(n^{(j)})Z(n^{(j)})R(n^{(j)})}{p(n^{(j)})}, n^{(j)} \overset{\text{iid}}{\sim} p \tag{2}$$

Thus, sampling at any level would also allow for zero variance / exact computation if similarly $p(n) \propto g(n)Z(n)R(n)$.

Note that in Abstraction Sampling each abstract state involves a node selection procedure analogous to importance sampling and that AOAS uses a proposal $p(n) \propto g(n)h(n)r(n)$. $g(n)$ can always be evaluated exactly. Then assuming that $h(n) = 0 \implies Z(n) = 0$ and $r(n) = 0 \implies R(n) = 0$, it naturally follows that designing each abstract states $\boldsymbol{A_i}$ such that $\forall n \in \boldsymbol{A_i}, h(n)r(n) = \alpha\, g(n)Z(n)R(n)$, for some constant α, we similarly achieve zero variance.

Definition 4.2.0.1 (Abstraction Function $h(n)r(n)$ vs. $Z(n)R(n)$ Proportionality)
An abstraction function $a(n)$ maintains $h(n)r(n)$ vs. $Z(n)R(n)$ proportionality if, for every abstract state A_i formed by $a(n)$, $\forall n \in A_i, h(n)r(n) = \alpha\, Z(n)R(n)$, for some constant α specific to A_i.

Definition 4.2.0.2 (Exact Abstraction Function)
An abstraction function $a(.)$ is exact for an abstraction sampling algorithm, AS, if use of $a(.)$ with AS always leads to AS estimates having zero variance and $\hat{Z} = Z$ for every AS probe.

Thus, we can say:

Theorem 4.2.0.2 (AOAS Exact Abstractions from $h(n)r(n)$ vs. $Z(n)R(n)$ Proportionality)
If an abstraction function $a(.)$ maintains $h(n)r(n)$ vs. $Z(n)R(n)$ Proportionality, then it is an exact abstraction function for AOAS. (Proof in Supplemental Materials)

Normally we neither have access to the proportionality constant α or even know whether nodes have the same α. However one idea is to use the magnitude of $h(n)r(n)$ itself as a heuristic for similarities in α. This drives the intuition for a new HR-Based class of abstractions.

Also from a sampling perspective, Rizzo [2007] showed the following about stratified importance sampling when sampling from equal area strata under the proposal:

Proposition 4.2.0.3 (Stratified Importance Sampling Variance Reduction)
Suppose that $M = mk$ is the number of replicates for an importance sampling estimator $\hat{\theta^I}$, and $\theta^{\hat{S}I}$ is a stratified importance sampling estimator, with estimates $\hat{\theta_j}$ for θ_j on the individual strata, each with m replicates. If $Var(\hat{\theta^I}) = \sigma^2/M$ and $Var(\hat{\theta_j}) = \sigma_j^2/m$, $j = 1, ..., k$, then

$$\sigma^2 - k\sum_{j=1}^{k}\sigma_j^2 \geq 0, \tag{3}$$

with equality if and only if $\theta_1 = ... = \theta_k$. Hence stratification never increases variance, and there exists a stratification that reduces the variance except when [the proposal function] $g(x)$ is constant.

Two takeaways from this proposition are that 1) we can achieve variance reduction with respect to importance sampling (analogous to Abstraction Sampling with all nodes placed into a single abstract state) by stratifying into equal area strata under the proposal, and 2) reducing the variance of each strata σ_j^2 leads to greater variance reduction. These will help drive the intuition for a new Q-Based abstraction class, as well as motivate several new partitioning schemes.

5 ADDITIONAL INFORMATION ABOUT VALUE-BASED ABSTRACTIONS

As described in the main paper, value-based abstraction functions consist of two parts: (1) a value function $\mu : n \to \mathbb{R}$ that assigns a real value on a positive scale to nodes n that are to be abstracted, and (2) a partitioning scheme that then abstracts nodes based on $\mu(n)$. And because $\mu(n)$ are values on a positive scale (implying semantics between smaller vs. larger values), the partitioning schemes can be designed to partition the nodes in a way that maintains an ordering of $\mu(n)$. This results in what we call value-based ordered abstractions.

Algorithm 2: General Value-0Ordered Abstraction Function Scheme

```
input  : A set of nodes $\boldsymbol{n}$ to be partitioned into abstract states; an abstraction value function $\mu(\cdot)$; a parameter $nAbs$ bounding the
         number of abstract states; a partitioning function $\Psi_o(\cdot)$ that partitions $\boldsymbol{n}$ into abstract states such that nodes are ordered by
         $\mu(n)$ according to sort-order $o$
output : Nodes $\boldsymbol{n}$ partitioned into abstract states $\boldsymbol{A} = \{\boldsymbol{A_i} \mid i <= nAbs\}$ such that sort order $o$ of $\mu(n)$ is maintained across all $\boldsymbol{A_i}$.
1 begin
2     if $|\boldsymbol{n}| <= nAbs$ then
3         $\boldsymbol{A} = \{\{n\} \mid n \in \boldsymbol{n}\}$
4     else
5         $\boldsymbol{A} = \Psi_o(\boldsymbol{n}, \mu, nAbs)$
6     return $\boldsymbol{A}$
7 end
```

6 DETAILED DESCRIPTIONS OF ORDERED PARTITIONING SCHEMES FOR VALUE BASED ABSTRACTIONS

We now present seven schemes, each defined by a unique sort order o and partition strategy Ψ combination. Each scheme uses a different method to partition nodes into abstract states keeping the nodes in sort order according to o. With a provided value function $\mu(.)$, each scheme can be used to form an ordered value abstraction function. In addition to defining each scheme, we also describe the motivation behind its creation.

Running Example As we motivate and describe the schemes, we will also provide an example of abstract states that would result from partitioning the following nodes:

$$\{1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100\} \tag{4}$$

into $nAbs = 4$ abstract states by each of partitioning schemes that will be presented.

6.0.1 simpleVB

$\Psi_{simpleVB}$ (Algorithm 3)

Algorithm 3: $\Psi_{simpleVB}$

```
input  : A set of nodes $\boldsymbol{n}$ to be partitioned into $nAbs$ abstract states; a value function $\mu(.)$
output : $\boldsymbol{n}$ partitioned into abstract states[1] $\boldsymbol{A} = \{\boldsymbol{A_i} \mid i \in \{1, ..., nAbs\}\}$ such that $\forall \boldsymbol{A_i}, \boldsymbol{A_j} \in \boldsymbol{A}, -1 \leq |\boldsymbol{A_i}| - |\boldsymbol{A_j}| \leq 1$
1  begin
2      $baseCardinality \leftarrow \lfloor \frac{|\boldsymbol{n}|}{nAbs} \rfloor$
3      $extras \leftarrow |\boldsymbol{n}| \mod nAbs$
4      $\boldsymbol{n}^* \leftarrow SORT(\boldsymbol{n}, \mu,$ low-to-high$)$
5      $j_{begin} \leftarrow 1$
6      foreach $i \leftarrow 1, ..., nAbs$ do
7          if $extras > 0$ then
8              $j_{end} \leftarrow j_{begin} + baseCardinality$
9              $extras \leftarrow extras - 1$
10         else
11             $j_{end} \leftarrow j_{begin} + baseCardinality - 1$
12         $\boldsymbol{A_i} \leftarrow \{n^*_{j_{begin}}, ..., n^*_{j_{end}}\}$
13         $j_{begin} \leftarrow j_{end} + 1$
14     end
15     $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
16     return $\boldsymbol{A}$
17 end
```

The simpleVB (simple value-based) scheme follows the motivation of grouping nodes of similar value in the same abstract state by a simple 2-step method: 1) first, nodes are ordered by their heuristic value (low to high), and 2) next the ordered nodes are partitioned into [approximately] equal cardinality abstract states.

Time Complexity.
Partitioning is achieved via one pass through $|\boldsymbol{n}^*|$ leading to $\mathcal{O}(|\boldsymbol{n}|\, log|\boldsymbol{n}|)$ time complexity due to sorting.

Space Complexity.
No more than linear space is required. $\mathcal{O}(|\boldsymbol{n}|)$

Result on Running Example.
$\{1.0, 1.1\}, \{1.2, 1.3\}, \{1.4, 1.5\}, \{10, 100\}$

Through its simplicity, this method aims to leverage speed allowing for abstractions to be formed much quicker leading to greater number of samples.

[1]Such that nodes maintain sort order o across all abstract states.

6.0.2 minVarVB

$\Psi = \Psi_{minVarVB}$ (Algorithm 4)

Algorithm 4: $\Psi_{minVarVB}$

```
input  : A set of nodes $\boldsymbol{n}$ to be partitioned into $nAbs$ abstract states; a value function $\mu(.)$
output : $\boldsymbol{n}$ partitioned into abstract states[1] $\boldsymbol{A} = \{\boldsymbol{A_i} \mid i \in \{1, ..., nAbs\}\}$ satisfying min $\sum_{\boldsymbol{A_i} \in \boldsymbol{A}} Var(\boldsymbol{A_i}, v)$
1 begin
2     $\boldsymbol{A} = WardsMethod(\boldsymbol{n}, nAbs, \mu(\cdot),$ Euclidian distance$)$
3     return $\boldsymbol{A}$
4 end
```

As mentioned in Section 4.2, Proposition 4.2.0.3, Rizzo [2007] showed that in stratified importance sampling minimizing variance of the estimates within individual strata can lead to a reduction in overall variance.

The minVarVB scheme was designed based on this intuition. The scheme uses Ward's Minimum Variance Hierarchical Clustering (or Ward's Method, for short) Ward [1963] to group nodes into a $nAbs$ abstract states so as to minimize variance within each abstract state with respect to the provided value function $\mu(.)$.

Ward's Minimum Variance Hierarchical Clustering is an agglomerative hierarchical clustering algorithm designed to create a dendrogram by iteratively merging clusters. The primary objective is to minimize the total within-cluster variance. Ward's method works as outlined in Algorithm 5.

Algorithm 5: Ward's Method

1. **Initialization:** Treat each data point as an individual cluster. Assign each cluster a label or identifier.
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.

Ward's Method can be combined with Lance-Williams linear distance updates Lance and Williams [1967] to increase efficiency. Lance-Williams linear distance updates, in the context of agglomerative clustering, refer to the formula used to calculate the distance between clusters as they are merged during the hierarchical clustering process. The general form of Lance-Williams distance updates can be expressed as follows:

$$d_{(ij)k} = \alpha_i d_{ik} + \alpha_j d_{jk} + \alpha d_{ij} + \gamma |d_{ik} - d_{jk}| \tag{5}$$

where:

- d_{ij}, d_{ik}, and d_{jk} are the pair-wise distances between clusters $\boldsymbol{C_i}$, $\boldsymbol{C_j}$, and $\boldsymbol{C_k}$
- $d_{(ij)k}$ is the distance between the newly merged cluster $\boldsymbol{C_i} \cup \boldsymbol{C_j}$ and cluster $\boldsymbol{C_k}$
- $\alpha_i, \alpha_j, \alpha,$ and γ are coefficients that depend on the linkage criterion used

In the case of Ward's method, the coefficients are specific to the minimization of within-cluster variance and are calculated as follows:

$$\begin{aligned}\alpha_i &= \frac{|\boldsymbol{C_i}| + |\boldsymbol{C_k}|}{|\boldsymbol{C_i}| + |\boldsymbol{C_j}| + |\boldsymbol{C_k}|} \\ \alpha_j &= \frac{|\boldsymbol{C_j}| + |\boldsymbol{C_k}|}{|\boldsymbol{C_i}| + |\boldsymbol{C_j}| + |\boldsymbol{C_k}|} \\ \alpha &= -\frac{|\boldsymbol{C_k}|}{|\boldsymbol{C_i}| + |\boldsymbol{C_j}| + |\boldsymbol{C_k}|} \\ \gamma &= 0\end{aligned} \tag{6}$$

(The inclusion of γ provides additional flexibility in the more general case, adjusting the distance updates based on the specific clustering criterion being used).

Time Complexity.[2]
The choice of clusters to merge generally leads to having a $\mathcal{O}(|\boldsymbol{n}|^3)$ time complexity due to the need to compare pair-wise distances between all clusters at each iteration. However, in the case where nodes are distributed linearly in one dimension, use of a priority queue, and using Lance-Williams distance updates, the time complexity is can be reduced to $\mathcal{O}(|\boldsymbol{n}|^2)$.

Space Complexity.[2]
The space complexity is implementation dependent, with most time-efficient variants making use of a distance matrix leading to $\mathcal{O}(|\boldsymbol{n}|^2)$ space complexity.

Result on Running Example.
$\{1.0, 1.1, 1.2\}, \{1.3, 1.4, 1.5\}, \{10\}, \{100\}$

In contrast to simpleVB, minVarVB places considerable resources into computing abstractions, leading to fewer samples, but with potentially better estimates with an appropriate value function $\mu(.)$.

6.0.3 equalDistVB

$\Psi_{equalDistVB}$ (Algorithm 6)

Algorithm 6: $\Psi_{equalDistVB}$

```
input  : A set of nodes $\boldsymbol{n}$ to be partitioned into $nAbs$ abstract states; a value function $\mu(.)$
output : With $\mu(\boldsymbol{A_{1,...,i}}) = (\sum_{j=1}^{i}\sum_{n' \in \boldsymbol{A_j}} \mu(n')$, $n^{\text{last}}_{\boldsymbol{A_i}}$ be the last node in $\boldsymbol{A_i}$, and $\mathcal{Q}_i = \frac{i \cdot \sum_{n \in \boldsymbol{n}^*} \mu(n)}{nAbs}$, $\boldsymbol{n}$ partitioned into
         abstract states[1] $\boldsymbol{A} = \{\boldsymbol{A_i} \mid i \in \{1, ..., nAbs\}\}$ such that for $i = 1, ..., nAbs$ in order, ( $\mu(\boldsymbol{A_{1,...,i}}) \geq \mathcal{Q}_i$ ) $\wedge$
         ( ( $\boldsymbol{A_i} = \{\}$ ) $\vee$ ( $\mu(\boldsymbol{A_{1,...,i}}) - \mu(n^{\text{last}}_{\boldsymbol{A_i}}) < \mathcal{Q}_i$ ) )
begin
    $\boldsymbol{n}^* \leftarrow SORT(\boldsymbol{n}, \mu,$ low-to-high$)$
    $j \leftarrow 1$
    foreach $i \leftarrow 1, ..., nAbs$ do
        $\boldsymbol{A_i} \leftarrow \{\}$
        while $\mu(\boldsymbol{A_{1,...,i}}) < \mathcal{Q}_i$ do
            $\boldsymbol{A_i} \leftarrow A_i \cup \{n^*_j\}$
            $j \leftarrow j + 1$
        end
    end
    $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
    return $\boldsymbol{A}$
end
```

In sampling it is generally beneficial to predominantly sample high impact regions of the search/sampling space. Allowing the provided value function $\mu(.)$ to serve as a heuristic of nodes that are part of these high impact spaces, equalDistVB attempts to balance this intuition with the notion of variance reduction from minVarVB in attempts to group fewer predicted high impact nodes together in abstract states and allowing for the predicted lower impact nodes to be part of larger abstract states. Also inspired by the simplicity of simpleVB, the scheme works by greedily adding nodes in value order (low to high) into abstract state $\boldsymbol{A_i}$ until the total sum of node values from $\boldsymbol{A_1}, ..., \boldsymbol{A_i}$ reaches or exceeds the $\frac{i}{nAbs}$ quantile.

[2]Assuming $\mu(n)$ is $\mathcal{O}(1)$ in both time and space.

When paired with the QB abstraction class, the equalDistVB schemes also attempts to partition notes into abstract states of equal mass under the proposal. This in corresponds to the condition for Proposition 4.2.0.3 for stratified importance sampling variance reduction.

Time Complexity.[2]
$\mu(\boldsymbol{A_{1...i}})$ can be updated progressively in constant time, and thus computation of $\mathcal{Q}_i$ at each iteration can also be done in constant time. Partitioning is achieved via one pass through $|\boldsymbol{n}^*|$ leading to $\mathcal{O}(|\boldsymbol{n}|\,log|\boldsymbol{n}|)$ time complexity due to sorting.

Space Complexity.[2]
No more than linear space is required. $\mathcal{O}(|\boldsymbol{n}|)$

Result on Running Example.
$\{1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 10, 100\}, \{\}, \{\}, \{\}$

Although, this method hopes to find a balance between intuitions previously explored, and without compromising speed and efficiency of abstract state generation, from the running example we can see how this method yield undesirable results in the presence of certain distributions of node values. In this example, the first quantile is only reached after all the nodes have been added to the first abstract state, leaving no nodes remaining to be partitioned into the subsequent abstract states.

6.0.4 equalDistVB2

$\Psi_{\textit{equalDistVB2}}$ (Algorithm 7)

Algorithm 7: $\Psi_{\textit{equalDistVB2}}$

input : A set of nodes $\boldsymbol{n}$ to be partitioned into $nAbs$ abstract states; a value function $\mu(.)$

output : With $\mu(\boldsymbol{A_{1,...,i}}) = (\sum_{j=1}^{i}\sum_{n'\in\boldsymbol{A_j}}\mu(n')$, $n^{\text{last}}_{\boldsymbol{A_i}}$ be the last node in $\boldsymbol{A_i}$, and $\mathcal{Q}_i = \frac{i\cdot\sum_{n\in\boldsymbol{n}^*}\mu(n)}{nAbs}$, $\boldsymbol{n}$ partitioned into abstract states[1] $\boldsymbol{A} = \{\boldsymbol{A_i} \mid i \in \{1, ..., nAbs\}\}$ such that for $i = 1, ..., nAbs$ in order, $(\ \mu(\boldsymbol{A_{1,...,i}}) \geq \mathcal{Q}_i\ ) \wedge$ $(\ (\ \boldsymbol{A_i} = \{\}\ ) \vee (\ \mu(\boldsymbol{A_{1,...,i}}) - \mu(n^{\text{last}}_{\boldsymbol{A_i}}) < \mathcal{Q}_i\ )\ )$

```
begin
    $\boldsymbol{n}^* \leftarrow SORT(\boldsymbol{n}, \mu, \text{high-to-low})$
    $j \leftarrow 1$
    foreach $i \leftarrow 1, ..., nAbs$ do
        $\boldsymbol{A_i} \leftarrow \{\}$
        while $\mu(\boldsymbol{A_{1,...,i}}) < \mathcal{Q}_i$ do
            $\boldsymbol{A_i} \leftarrow A_i \cup \{n^*_j\}$
            $j \leftarrow j + 1$
        end
    end
    $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs}\boldsymbol{A_i}$
    return $\boldsymbol{A}$
end
```

By simply reversing the sort order, equalDistVB2 is able to use the same partitioning strategy $\Psi_{\textit{equalDistVB}}$ associated with equalDistVB meanwhile mitigate some of the overfilling of abstract states.

Time Complexity.[2]
$\mu(\boldsymbol{A_{1...i}})$ can be updated progressively in constant time, and thus computation of $\mathcal{Q}_i$ at each iteration can also be done in constant time. Partitioning is achieved via one pass through $|\boldsymbol{n}^*|$ leading to $\mathcal{O}(|\boldsymbol{n}|\,log|\boldsymbol{n}|)$ time complexity due to sorting.

Space Complexity.[2]
No more than linear space is required. $\mathcal{O}(|\boldsymbol{n}|)$

Result on Running Example.
$\{100\}, \{\}, \{\}, \{10, 1.5, 1.4, 1.3, 1.2, 1.1, 1.0\}$

We see that equalDistVB2 can still be subject to over packing of abstract states. Next we present two more equalDistvB variants that continue to mitigate this artifact.

6.0.5 equalDistVB3

$\Psi_{equalDistVB3}$ (Algorithm 8)

Algorithm 8: $\Psi_{equalDistVB3}$

input : A set of nodes $\boldsymbol{n}$ to be partitioned into $nAbs$ abstract states; a value function $\mu(.)$

output : With $\mu(\boldsymbol{A_{1,...,i}}) = (\sum_{j=1}^{i} \sum_{n' \in \boldsymbol{A_j}} \mu(n')$, $n_{\boldsymbol{A_i}}^{\text{last}}$ be the last node in $\boldsymbol{A_i}$, and $\mathcal{Q}_i = \frac{i \cdot \sum_{n \in \boldsymbol{n}^*} \mu(n)}{nAbs}$, $\boldsymbol{n}$ partitioned into abstract states[1] $\boldsymbol{A} = \{\boldsymbol{A_i} \mid i \in \{1, ..., nAbs\}\}$ such that for $i = 1, ..., nAbs$ in order, $(\ \mu(\boldsymbol{A_{1,...,i}}) \geq \mathcal{Q}_i\ ) \wedge$ $(\ (\ |\boldsymbol{A_i}| = 1\ ) \vee (\ \mu(\boldsymbol{A_{1,...,i}}) - \mu(n_{\boldsymbol{A_i}}^{\text{last}}) < \mathcal{Q}_i\ )\ )$

```
begin
    $\boldsymbol{n}^* \leftarrow SORT(\boldsymbol{n}, \mu, \text{high-to-low})$
    $j \leftarrow 1$
    foreach $i \leftarrow 1, ..., nAbs$ do
        $\boldsymbol{A_i} \leftarrow \{n_j^*\}$
        $j \leftarrow j + 1;$
        while $\mu(\boldsymbol{A_{1,...,i}}) < \mathcal{Q}_i$ do
            $\boldsymbol{A_i} \leftarrow A_i \cup \{n_j^*\}$
            $j \leftarrow j + 1$
        end
    end
    $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
    return $\boldsymbol{A}$
end
```

In order to lessen over packing and ensure abtract states are not left empty, equalDistVB3 modifies equalDistVB2 so that, after processing of each abstract state, the next state is forced an addition of at least a single node by default.

Time Complexity.[2]
$\mu(\boldsymbol{A_{1...i}})$ can be updated progressively in constant time, and thus computation of $\mathcal{Q}_i$ at each iteration can also be done in constant time. Partitioning is achieved via one pass through $|\boldsymbol{n}^*|$ leading to $\mathcal{O}(|\boldsymbol{n}|\ log|\boldsymbol{n}|)$ time complexity due to sorting.

Space Complexity.[2]
No more than linear space is required. $\mathcal{O}(|\boldsymbol{n}|)$

Result on Running Example.
$\{100\}, \{10\}, \{1.5\}, \{1.4, 1.3, 1.2, 1.1, 1.0\}$

Still highly efficient, equalDistVB3 manages to ensure that the provided $nAbs$ granularity is honored, allowing users better control of the search vs. sampling interpolation possible with Abstraction Sampling.

6.0.6 equalDistVB4

$\Psi_{equalDistVB4}$ (Algorithm 9)

The final varaint of the equalDist schemes, equalDistVB4 attempts to perform a more even partitioning than the previous variants by recomputing quantiles. Each time the algorithm progesses to processing a new abstract state, remaining nodes and abstract states are used to compute new quantiles which are then used to guide filling of the current abstract state in the same way previously done.

Time Complexity.[2]
$\mu(\boldsymbol{A_{1...i}})$ can be updated progressively in constant time, and thus computation of $\widehat{\mathcal{Q}}_i$ at each iteration can also be done in constant time. Partitioning is achieved via one pass through $|\boldsymbol{n}^*|$ leading to $\mathcal{O}(|\boldsymbol{n}|\ log|\boldsymbol{n}|)$ time complexity due to sorting.

Space Complexity.[2]
No more than linear space is required. $\mathcal{O}(|\boldsymbol{n}|)$

Result on Running Example.
$\{100\}, \{10\}, \{1.5, 1.4, 1.3\}, \{1.2, 1.1, 1.0\}$

Still highly efficient, equalDistVB3 manages to ensure that the provided $nAbs$ granularity is honored, allowing users better

```
Algorithm 9: $\Psi_{equalDistVB4}$
input  : A set of nodes $\boldsymbol{n}$ to be partitioned into $nAbs$ abstract states; a value function $\mu(.)$
output : With $\mu(\boldsymbol{A_{1,...,i}}) = (\sum_{j=1}^{i} \sum_{n' \in \boldsymbol{A_j}} \mu(n')$, $n^{\text{last}}_{\boldsymbol{A_i}}$ be the last node in $\boldsymbol{A_i}$, and $\widehat{\mathcal{Q}}_i = \frac{\mu(\boldsymbol{n^*}) - \mu(\boldsymbol{A_{1,...,i-1}})}{nAbs-i+1}$, $\boldsymbol{n}$ partitioned
         into abstract states[1] $\boldsymbol{A} = \{\boldsymbol{A_i} \mid i \in \{1, ..., nAbs\}\}$ such that for $i = 1, ..., nAbs$ in order, ( $\mu(\boldsymbol{A_i}) \geq \widehat{\mathcal{Q}}_i$ ) $\wedge$
         ( ( $|\boldsymbol{A_i}| = 1$ ) $\vee$ ( $\mu(\boldsymbol{A_i}) - \mu(n^{\text{last}}_{\boldsymbol{A_i}}) < \widehat{\mathcal{Q}}_i$ ) )
begin
    $\boldsymbol{n^*} \leftarrow SORT(\boldsymbol{n}, \mu,$ high-to-low$)$
    $j \leftarrow 1$
    foreach $i \leftarrow 1, ..., nAbs$ do
        $\boldsymbol{A_i} \leftarrow \{\}$
        while $\mu(\boldsymbol{A_i}) < \widehat{\mathcal{Q}}_i$ do
            $\boldsymbol{A_i} \leftarrow A_i \cup \{n^*_j\}$
            $j \leftarrow j + 1$
        end
    end
    $\boldsymbol{A} \leftarrow \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
    return $\boldsymbol{A}$
end
```

control of the search vs. sampling interpolation possible with Abstraction Sampling.

6.0.7 randVB

Ψ_{randVB} (Algorithm 10)

```
Algorithm 10: $\Psi_{randVB}$
input  : A set of nodes $\boldsymbol{n}$ to be partitioned into $nAbs$ abstract states; a value function $\mu(.)$
output : $\boldsymbol{n}$ partitioned into abstract states[1] $\boldsymbol{A} = \{\boldsymbol{A_i} \mid i \in \{1, ..., nAbs\}\}$
begin
    $\boldsymbol{n^*} \leftarrow SORT(\boldsymbol{n}, \mu,$ high-to-low$)$
    $\boldsymbol{s} \sim Unif(\{\boldsymbol{M} \subseteq \{1, ..., |\boldsymbol{n^*}| - 1\} \mid |\boldsymbol{M}| = nAbs - 1\})$
    $\boldsymbol{s^*} \leftarrow SORT(\boldsymbol{s})$
    $j \leftarrow 1$
    foreach $i \leftarrow 1, ..., nAbs-1$ do
        $\boldsymbol{A_i} = \{n^*_j, ..., n^*_{s^*_i}\}$
        $j \leftarrow s^*_i + 1$
    end
    $\boldsymbol{A_{nAbs}} = \{n^*_j, ..., n^*_{|n^*|}\}$
    $\boldsymbol{A} = \cup_{i=1}^{nAbs} \boldsymbol{A_i}$
    return $\boldsymbol{A}$
end
```

If the quality of $\mu(.)$ as a measure of similarity is unknown or poor, it could instead be beneficial to rely on randomness to ensure a diverse sampling of abstractions. randVB does this by sampling $nAbs-1$ partition points between the sorted nodes $\boldsymbol{n^*}$ uniformly at random and without replacement, and then partitions the nodes accordingly. As a result, abstract states are formed such that nodes are still grouped according to $\mu(.)$, but the size of those groups varies.

Time Complexity.[2]
$\mathcal{O}(|\boldsymbol{n}|\, log|\boldsymbol{n}|)$ time complexity due to sorting.

Space Complexity.[2]
No more than linear space is required. $\mathcal{O}(|\boldsymbol{n}|)$

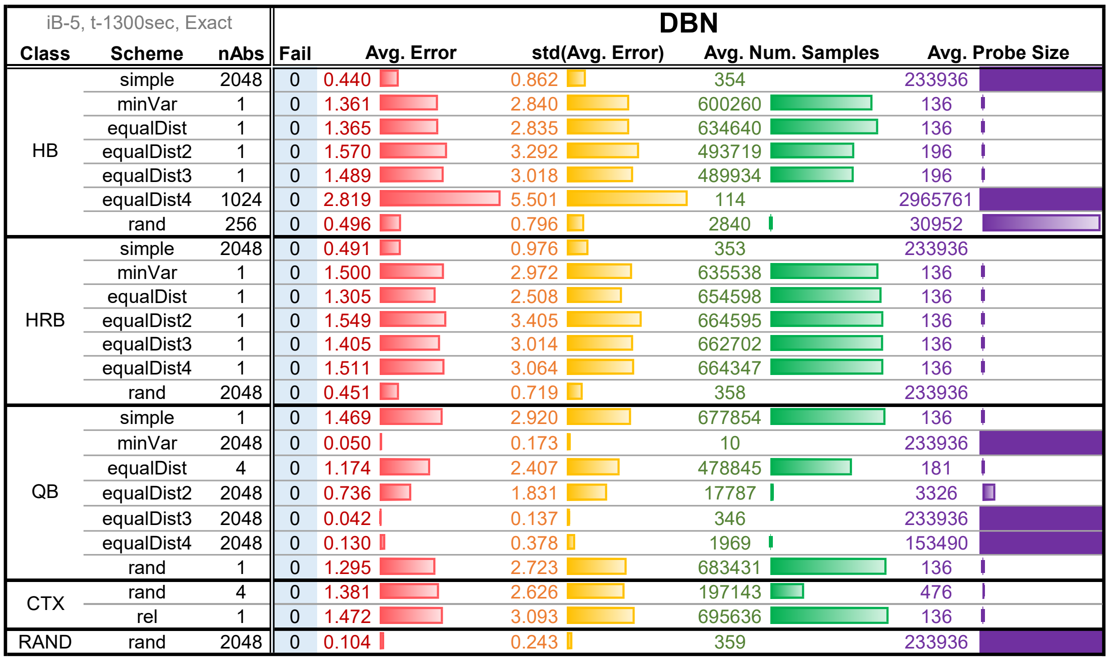
Result on Running Example.
$\{100, 10\}, \{1.5\}, \{1.4, 1.3, 1.2\}, \{1.1, 1.0\}$;
$\{100\}, \{10, 1.5, 1.4, 1.3\}, \{1.2, 1.1\}, \{1.0\}$; ...etc.

7 EXTENDED RESULTS

In extension to the main paper, here we show a more comprehensive set of aggregated data tables, now also including the standard deviation of the errors, the average number of samples drawn, and average probe sizes.

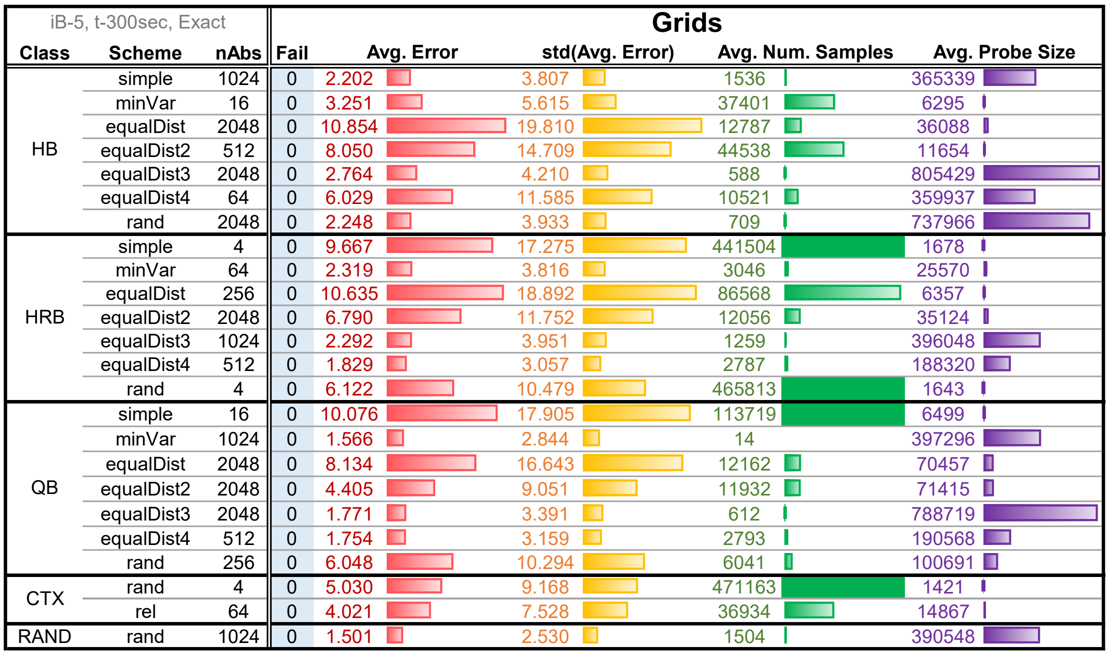
7.1 SUMMARY COMPARISON.

7.1.1 Exact Problems



iB-5, t-1300sec, Exact			DBN				
Class	Scheme	nAbs	Fail	Avg. Error	std(Avg. Error)	Avg. Num. Samples	Avg. Probe Size
HB	simple	2048	0	0.440	0.862	354	233936
	minVar	1	0	1.361	2.840	600260	136
	equalDist	1	0	1.365	2.835	634640	136
	equalDist2	1	0	1.570	3.292	493719	196
	equalDist3	1	0	1.489	3.018	489934	196
	equalDist4	1024	0	2.819	5.501	114	2965761
	rand	256	0	0.496	0.796	2840	30952
HRB	simple	2048	0	0.491	0.976	353	233936
	minVar	1	0	1.500	2.972	635538	136
	equalDist	1	0	1.305	2.508	654598	136
	equalDist2	1	0	1.549	3.405	664595	136
	equalDist3	1	0	1.405	3.014	662702	136
	equalDist4	1	0	1.511	3.064	664347	136
	rand	2048	0	0.451	0.719	358	233936
QB	simple	1	0	1.469	2.920	677854	136
	minVar	2048	0	0.050	0.173	10	233936
	equalDist	4	0	1.174	2.407	478845	181
	equalDist2	2048	0	0.736	1.831	17787	3326
	equalDist3	2048	0	0.042	0.137	346	233936
	equalDist4	2048	0	0.130	0.378	1969	153490
	rand	1	0	1.295	2.723	683431	136
CTX	rand	4	0	1.381	2.626	197143	476
	rel	1	0	1.472	3.093	695636	136
RAND	rand	2048	0	0.104	0.243	359	233936

Table 1



iB-5, t-300sec, Exact			Grids				
Class	Scheme	nAbs	Fail	Avg. Error	std(Avg. Error)	Avg. Num. Samples	Avg. Probe Size
HB	simple	1024	0	2.202	3.807	1536	365339
	minVar	16	0	3.251	5.615	37401	6295
	equalDist	2048	0	10.854	19.810	12787	36088
	equalDist2	512	0	8.050	14.709	44538	11654
	equalDist3	2048	0	2.764	4.210	588	805429
	equalDist4	64	0	6.029	11.585	10521	359937
	rand	2048	0	2.248	3.933	709	737966
HRB	simple	4	0	9.667	17.275	441504	1678
	minVar	64	0	2.319	3.816	3046	25570
	equalDist	256	0	10.635	18.892	86568	6357
	equalDist2	2048	0	6.790	11.752	12056	35124
	equalDist3	1024	0	2.292	3.951	1259	396048
	equalDist4	512	0	1.829	3.057	2787	188320
	rand	4	0	6.122	10.479	465813	1643
QB	simple	16	0	10.076	17.905	113719	6499
	minVar	1024	0	1.566	2.844	14	397296
	equalDist	2048	0	8.134	16.643	12162	70457
	equalDist2	2048	0	4.405	9.051	11932	71415
	equalDist3	2048	0	1.771	3.391	612	788719
	equalDist4	512	0	1.754	3.159	2793	190568
	rand	256	0	6.048	10.294	6041	100691
CTX	rand	4	0	5.030	9.168	471163	1421
	rel	64	0	4.021	7.528	36934	14867
RAND	rand	1024	0	1.501	2.530	1504	390548

Table 2

iB-5, t-300sec, Exact			Pedigree				
Class	Scheme	nAbs	Fail	Avg. Error	std(Avg. Error)	Avg. Num. Samples	Avg. Probe Size
HB	simple	2048	0	0.150	0.564	393	1208067
	minVar	64	0	0.422	0.894	1904	34760
	equalDist	1024	0	0.303	0.626	1104	406884
	equalDist2	1024	0	0.315	0.536	1090	410306
	equalDist3	1024	0	0.279	0.539	727	606552
	equalDist4	512	0	0.214	0.622	1526	305759
	rand	2048	0	0.185	0.473	406	1170793
HRB	simple	256	0	0.225	0.378	3637	155656
	minVar	256	0	0.309	0.543	131	149534
	equalDist	1024	0	0.638	0.921	1653	247759
	equalDist2	16	0	0.457	0.646	83869	5396
	equalDist3	16	0	0.537	0.843	63832	8067
	equalDist4	64	0	0.483	0.836	14789	34813
	rand	64	0	0.666	0.983	17216	36226
QB	simple	256	0	0.297	0.510	3672	153687
	minVar	64	0	0.210	0.561	1939	36977
	equalDist	2048	0	0.144	0.646	524	808760
	equalDist2	1024	0	0.145	0.637	1067	410631
	equalDist3	512	0	0.148	0.643	1403	324983
	equalDist4	512	0	0.134	0.600	1415	322792
	rand	16	0	0.740	1.021	76974	8055
CTX	rand	16	0	0.540	0.827	169911	2790
	rel	64	0	0.424	0.653	28214	29061
RAND	rand	1024	0	0.143	0.619	878	620063

Table 3

iB-5, t-300sec, Exact			Promedas				
Class	Scheme	nAbs	Fail	Avg. Error	std(Avg. Error)	Avg. Num. Samples	Avg. Probe Size
HB	simple	1024	0	0.575	1.288	7878163	215898
	minVar	16	2	2.509	5.329	4119191	2304
	equalDist	1024	0	2.332	3.857	8057221	145513
	equalDist2	64	0	2.123	4.632	8086745	8209
	equalDist3	256	0	2.196	4.354	8212578	53287
	equalDist4	2048	0	1.355	2.486	8106429	471471
	rand	2048	0	0.752	1.476	8136226	382946
HRB	simple	2048	0	0.705	1.594	8281435	444640
	minVar	16	1	2.801	5.552	8302630	2403
	equalDist	16	4	4.055	7.212	8505442	1255
	equalDist2	16	2	3.445	6.549	8445561	1667
	equalDist3	16	2	2.656	5.561	8389700	2330
	equalDist4	2048	0	2.024	3.247	8278922	429451
	rand	1024	1	2.165	4.691	8284836	184056
QB	simple	256	1	3.164	5.634	8156519	44804
	minVar	64	1	1.062	3.999	8149950	13097
	equalDist	2048	0	0.583	1.053	8159447	85975
	equalDist2	2048	0	0.539	1.098	8146812	87006
	equalDist3	2048	0	0.412	0.917	8136397	517395
	equalDist4	512	0	0.437	1.062	8155880	126503
	rand	16	2	5.988	12.148	8401169	1892
CTX	rand	1024	1	2.442	4.755	8045093	2016
	rel	64	6	4.349	7.852	8384108	3268
RAND	rand	1024	0	0.513	1.033	8047804	228960

Table 4

7.1.2 LARGE Problems

iB-10, t-1200sec, LARGE			DBN				
Class	Scheme	nAbs	Fail	Avg. Error	std(Avg. Error)	Avg. Num. Samples	Avg. Probe Size
HB	simple	512	0	3.059	5.994	466	213236
	minVar	1	0	6.372	10.410	170425	434
	equalDist	1	0	6.354	10.259	171742	434
	equalDist2	1	0	6.172	9.889	146566	598
	equalDist3	1	0	6.548	10.206	144646	598
	equalDist4	1	0	6.525	10.576	162296	434
	rand	64	0	1.855	2.986	3682	27039
HRB	simple	2048	0	3.202	6.388	116	844724
	minVar	1	0	6.102	9.811	167382	434
	equalDist	1	0	6.273	10.219	165303	434
	equalDist2	1	0	6.689	10.719	164615	434
	equalDist3	1	0	6.564	10.301	163186	434
	equalDist4	1	0	6.606	10.704	162441	434
	rand	2048	0	1.915	3.994	116	844724
QB	simple	1	0	6.540	10.583	162844	434
	minVar	2048	0	1.837	4.023	11	844724
	equalDist	512	0	5.423	9.545	28518	50129
	equalDist2	2048	0	3.813	7.105	11104	162286
	equalDist3	2048	0	1.645	3.853	115	844724
	equalDist4	2048	0	1.643	3.847	170	758313
	rand	4	0	6.292	9.781	52602	1721
CTX	rand	64	0	5.710	8.760	4947	22519
	rel	1	0	6.267	10.128	165870	434
RAND	rand	2048	0	2.123	4.214	116	844724

Table 5

iB-10, t-1200sec, LARGE			Grids				
Class	Scheme	nAbs	Fail	Avg. Error	std(Avg. Error)	Avg. Num. Samples	Avg. Probe Size
HB	simple	2048	0	73.710	117.967	585	5281698
	minVar	64	0	71.628	112.070	1948	184817
	equalDist	2048	0	149.888	252.503	6894	438547
	equalDist2	1024	0	119.823	195.442	12859	247710
	equalDist3	64	0	82.927	124.857	15758	197893
	equalDist4	1024	0	63.194	97.515	1020	2846847
	rand	2048	0	82.203	132.286	527	5492402
HRB	simple	1024	0	193.654	311.138	1042	3061184
	minVar	512	0	37.972	56.653	29	1534848
	equalDist	2048	0	127.990	216.992	6524	475696
	equalDist2	2048	0	104.502	168.754	6388	501514
	equalDist3	2048	0	38.936	52.976	429	6090687
	equalDist4	2048	0	34.676	50.051	460	5664129
	rand	16	0	160.168	262.678	78263	48729
QB	simple	16	0	197.931	331.349	73034	51032
	minVar	1024	0	28.423	44.701	7	3064517
	equalDist	2048	0	118.547	209.112	6013	932447
	equalDist2	2048	0	91.994	160.979	5935	939064
	equalDist3	2048	0	19.277	31.795	429	6135039
	equalDist4	2048	0	18.866	34.470	462	5658527
	rand	16	0	163.973	270.397	78137	48849
CTX	rand	512	0	111.104	189.309	53385	66495
	rel	1024	0	80.633	131.304	1990	1210381
RAND	rand	2048	0	19.053	30.561	517	5915471

Table 6

iB-10, t-1200sec, LARGE			Linkage-Type4				
Class	Scheme	nAbs	Fail	Avg. Error	std(Avg. Error)	Avg. Num. Samples	Avg. Probe Size
HB	simple	2048	21	47.383	124.818	215	6362535
	minVar	256	37	133.377	208.955	118	526228
	equalDist	2048	37	136.462	209.952	806	1725651
	equalDist2	2048	36	132.531	206.716	775	1763041
	equalDist3	2048	32	118.653	191.472	373	4231305
	equalDist4	2048	29	98.222	180.908	260	5348049
	rand	2048	21	52.171	117.178	258	5548243
HRB	simple	2048	17	48.474	105.528	201	7170175
	minVar	512	38	138.131	211.272	31	1203151
	equalDist	2048	32	123.253	192.089	1138	1438777
	equalDist2	2048	34	129.751	198.056	1114	1453506
	equalDist3	2048	31	118.091	185.967	405	4586845
	equalDist4	2048	26	95.895	158.305	335	5182785
	rand	1024	18	127.021	162.172	576	3170049
QB	simple	2048	13	48.681	102.256	165	7582217
	minVar	256	31	93.058	176.650	115	595380
	equalDist	2048	22	46.196	128.408	324	3606296
	equalDist2	1024	21	40.310	115.108	823	1613744
	equalDist3	1024	20	37.490	115.666	428	3151667
	equalDist4	2048	16	30.512	104.300	155	7276760
	rand	256	17	156.992	197.622	2123	786014
CTX	rand	2048	53	194.741	250.879	78237	12693
	rel	1024	37	129.189	210.249	911	2128473
RAND	rand	1024	19	33.804	107.942	531	3043774

Table 7

iB-10, t-1200sec, LARGE			Promedas				
Class	Scheme	nAbs	Fail	Avg. Error	std(Avg. Error)	Avg. Num. Samples	Avg. Probe Size
HB	simple	1024	16	5.981	14.402	5303	316842
	minVar	16	25	9.433	17.375	135360	3961
	equalDist	64	23	9.664	16.936	122333	11729
	equalDist2	16	22	9.465	17.026	438953	3209
	equalDist3	16	18	8.534	16.129	364644	3961
	equalDist4	16	19	8.011	15.663	368986	3973
	rand	64	22	8.296	16.348	129906	14836
HRB	simple	512	15	5.849	14.157	10763	158416
	minVar	16	24	9.577	17.048	130796	4001
	equalDist	16	32	11.596	19.010	546356	2629
	equalDist2	4	25	10.380	17.881	1755156	841
	equalDist3	16	22	9.779	17.253	388573	3844
	equalDist4	16	22	9.217	16.843	383539	3876
	rand	64	27	9.556	17.661	128420	15010
QB	simple	4	34	11.919	19.156	2214241	849
	minVar	16	13	5.403	13.076	127451	4261
	equalDist	512	15	5.960	13.509	21151	61005
	equalDist2	2048	12	4.982	12.955	5495	230190
	equalDist3	256	5	2.560	8.629	16078	90936
	equalDist4	512	5	2.476	8.229	7638	187975
	rand	4	28	11.532	19.413	2330332	841
CTX	rand	256	0	3.222	5.085	160087	12862
	rel	16	34	11.247	18.992	761684	2399
RAND	rand	1024	10	3.936	11.615	5010	348002

Table 8

7.2 COMPARISON USING 100 SAMPLES.

7.2.1 Exact Problems

Table 9

iB-5, m-100, Exact			DBN		Grids		Pedigree		Promedas	
Class	Scheme	nAbs	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error
HB	simpleQB	256	0	1.601	0	4.768	0	0.337	14	3.121
	minVarQB	256	0	5.028	0	5.134	0	0.615	1	5.423
	equalDist	256	0	5.269	0	15.958	1	2.145	13	6.556
	equalDist2	256	0	5.966	0	11.009	0	1.384	6	6.464
	equalDist3	256	0	6.203	0	5.804	0	0.669	1	5.480
	equalDist4	256	0	4.501	0	22.576	0	1.103	1	4.382
	randQB	256	0	0.712	0	5.515	0	0.531	13	4.988
HRB	simpleQB	256	0	1.638	0	15.757	0	0.721	14	3.014
	minVarQB	256	0	4.703	0	2.404	0	0.287	1	4.295
	equalDist	256	0	6.030	0	16.132	1	2.817	13	8.830
	equalDist2	256	0	6.361	0	10.462	0	2.546	6	8.272
	equalDist3	256	0	6.613	0	4.236	0	2.291	1	7.427
	equalDist4	256	0	6.753	0	3.179	0	1.241	1	5.552
	randQB	256	0	0.720	0	9.838	0	1.818	13	7.074
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

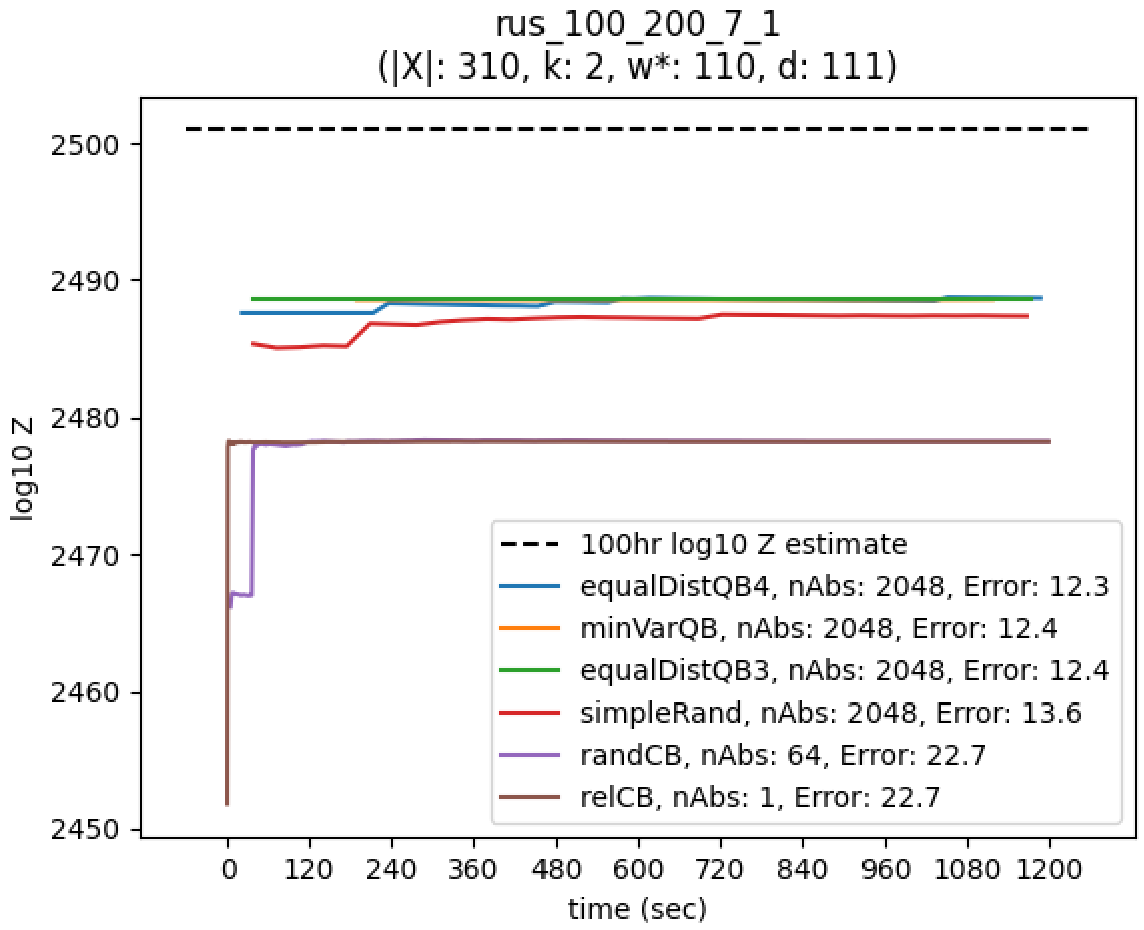
7.2.2 LARGE Problems

Table 10

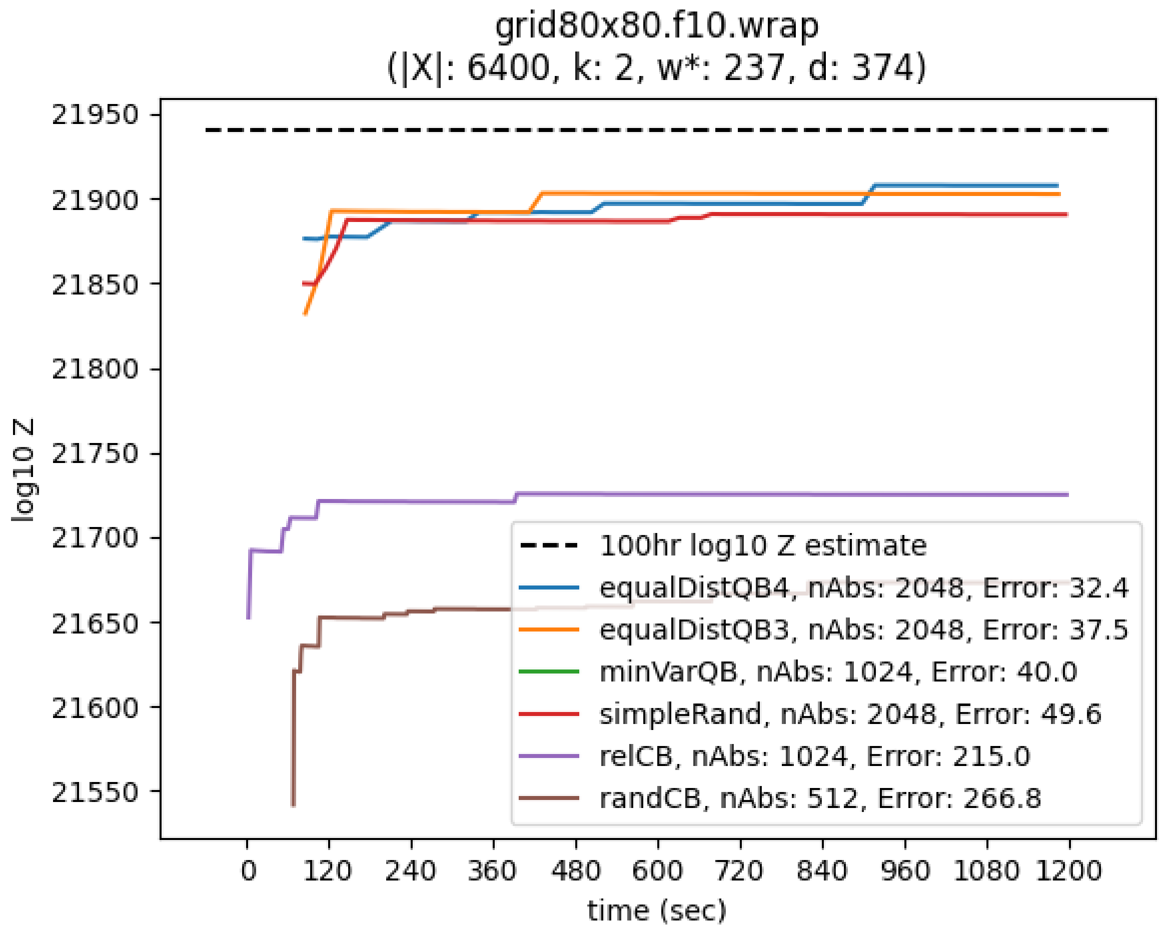
iB-10, m-100, LARGE			DBN		Grids		Linkage-Type4		Promedas	
Class	Scheme	nAbs	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error	Fail	Avg. Error
HB	simpleQB	256	0	4.179	0	108.953	0	189.141	14	16.389
	minVarQB	256	0	8.219	0	45.460	0	182.791	1	17.221
	equalDist	256	0	8.013	0	164.767	1	230.627	13	19.890
	equalDist2	256	0	8.233	0	119.203	0	231.620	6	18.944
	equalDist3	256	0	7.905	0	67.626	0	219.364	1	18.612
	equalDist4	256	0	7.588	0	54.643	0	199.565	1	17.186
	randQB	256	0	3.741	0	108.760	0	203.436	13	18.494
HRB	simpleQB	256	0	4.203	0	190.126	0	180.424	14	15.857
	minVarQB	256	0	7.770	0	29.575	0	188.654	1	17.492
	equalDist	256	0	7.947	0	151.765	1	235.331	13	20.390
	equalDist2	256	0	8.616	0	114.215	0	229.609	6	20.395
	equalDist3	256	0	7.653	0	37.005	0	222.866	1	19.932
	equalDist4	256	0	8.201	0	31.368	0	213.918	1	18.694
	randQB	256	0	3.254	0	150.130	0	205.219	13	19.157
QB	simpleQB	256	0	7.921	0	194.220	0	180.487	14	22.732
	minVarQB	256	0	2.848	0	22.838	0	182.296	1	11.742
	equalDist	256	0	6.443	0	140.283	1	192.449	13	17.245
	equalDist2	256	0	4.583	0	96.859	0	193.109	6	15.704
	equalDist3	256	0	3.036	0	25.042	0	170.706	1	11.426
	equalDist4	256	0	2.715	0	20.978	0	162.793	1	11.885
	randQB	256	0	7.791	0	163.214	0	205.186	13	23.984
CTX	rand	256	0	4.789	0	97.951	2	232.778	3	16.285
	rel	256	0	7.664	0	65.146	0	188.194	36	20.609
RAND	rand	256	0	3.070	0	26.185	0	178.273	3	13.957

7.3 TIME SERIES PLOT

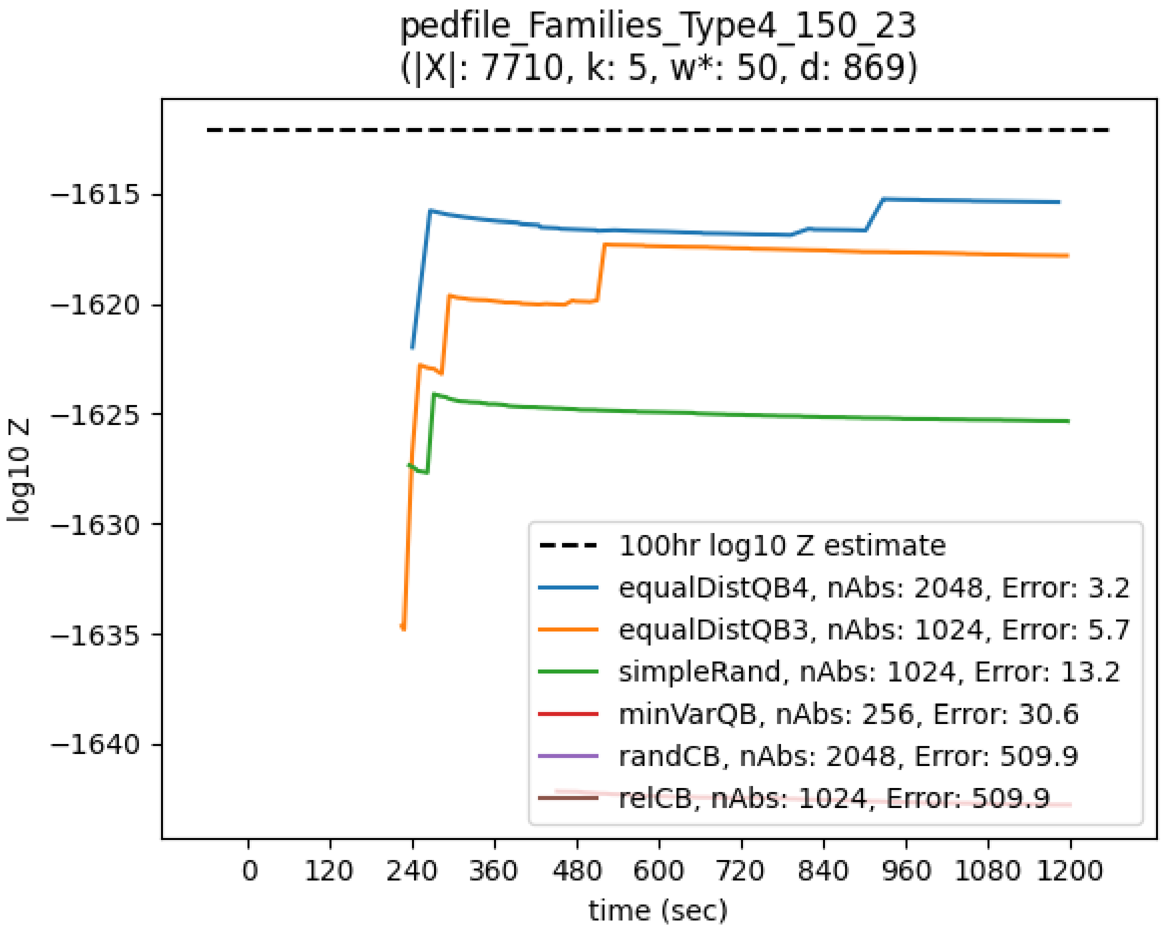
7.3.1 LARGE Problems



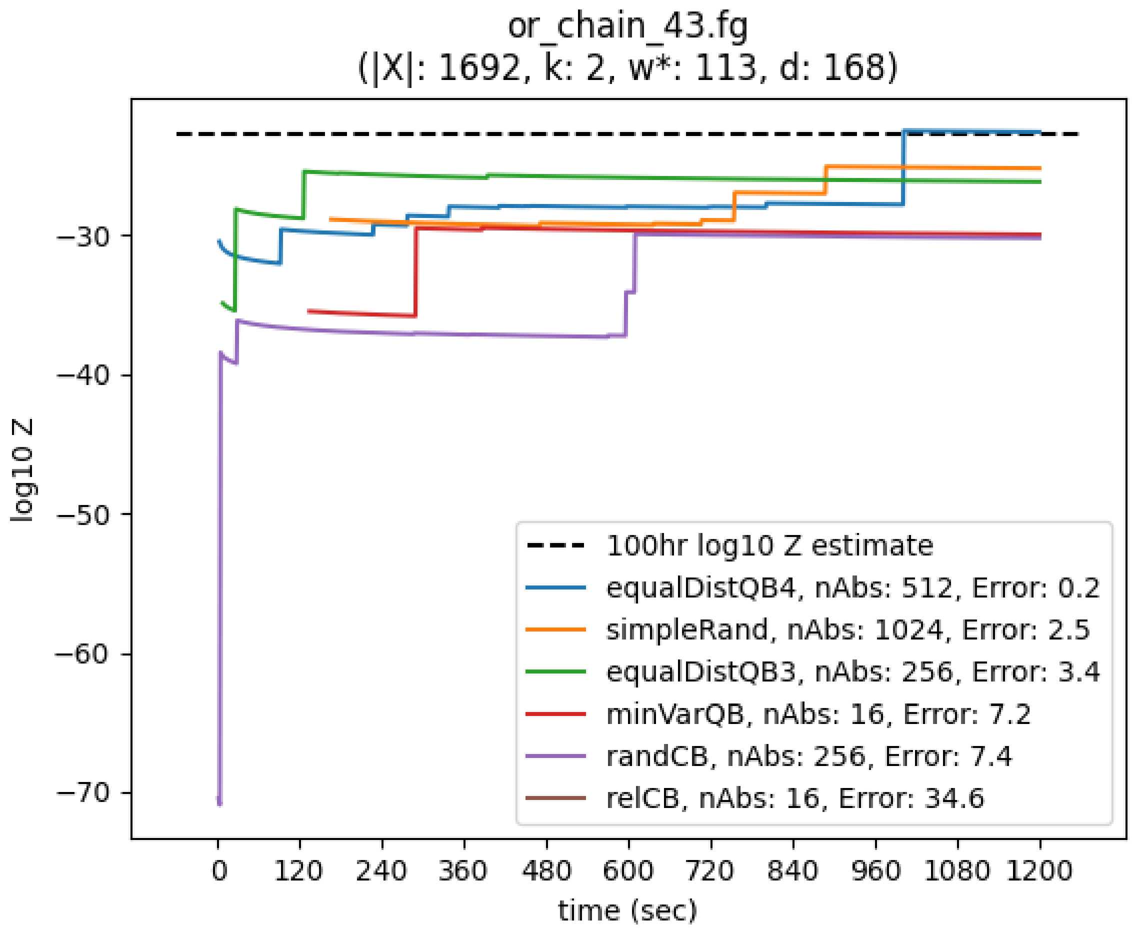
Plot 1: Z estimates from various algorithms versus time on DBN problem rus_100_200_7_1 using $iB = 10$. The dashed black line shows the estimated true Z value.



Plot 2: Z estimates from various algorithms versus time on Grids problem grid80x80.f10.wrap using $iB = 10$. The dashed black line shows the estimated true Z value.



Plot 3: Z estimates from various algorithms versus time on Linkage-Type4 problem grid20x20.f15 using $iB = 10$. The dashed black line shows the estimated true Z value.



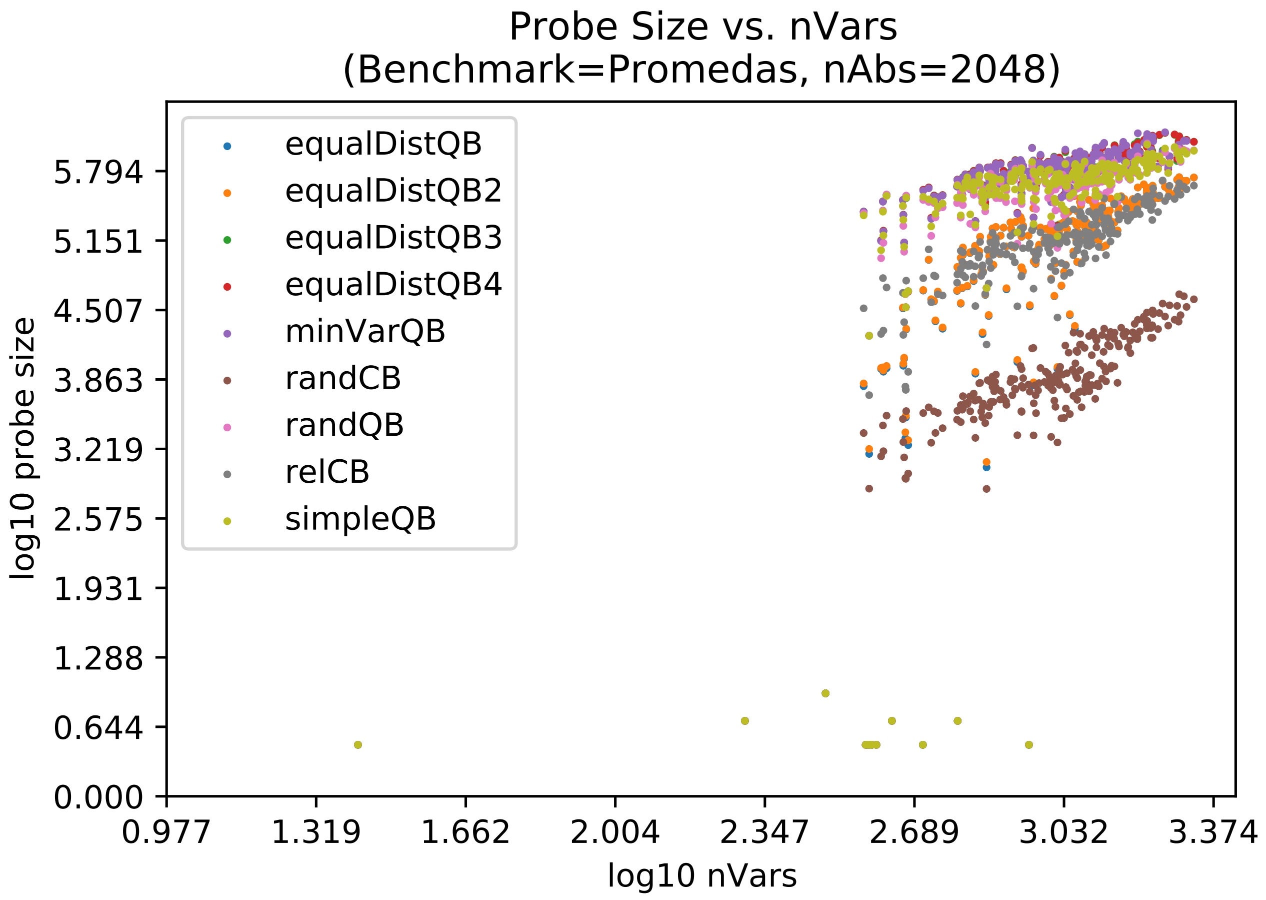
Plot 4: Z estimates from various algorithms versus time on Promedas problem or_chain_43.fg using $iB = 10$. The dashed black line shows the estimated true Z value.

8 ADDITIONAL RESULTS

8.1 PROBE SIZE

In our running abstraction example discussed in Supplemental Section 6, we observed that despite employing the same granularity, certain Ordered Partitioning Schemes may underutilize the allotted number of abstract states. Moreover, paths extended during initial iterations may become incomplete in subsequent iterations . These truncated paths may be pruned altogether and cut the number of nodes. To assess how effectively different schemes handle continual extension of paths, we fixed $nAbs$ at 2048 and plotted the Probe Size against the number of variables for each problem in the Promedas benchmark (Plot 5).

Plot 5: For the given abstraction granularity and benchmark, the size of the probe (in log10) relative to the number of problem variables (in log10) using iB-10.

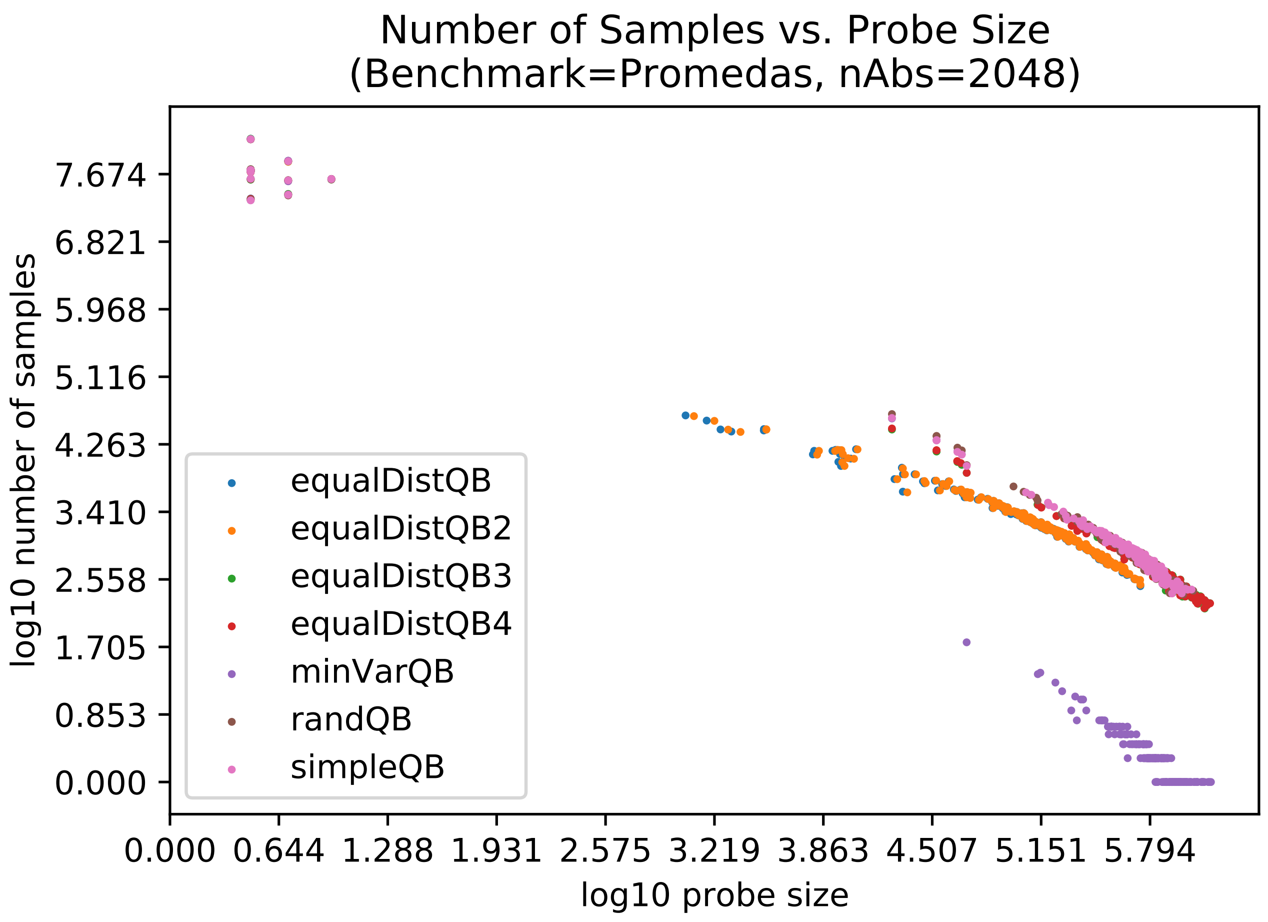


Even with the same granularity different abstraction functions can lead to vastly different utilization of abstract states, pruning, and thus probe sizes. Plot 5 highlights this. As seen in the plot (and generalizes across the different benchmarks and abstraction value classes) the simpleQB, minVarQB, equalDistQB3, equalDistQB4, and randQB schemes tend to produce larger probes, indicating more of the allotted abstract states utilized and fewer branches being pruned.

8.2 ABSTRACTION SPEED

In order to understand more about the speed of each scheme at performing abstractions, in Figure **??** we plot the number of samples versus average probe size for problems of the Promedas benchmark. (For other benchmarks and $nAbs$, please see the Supplemental Materials).

Plot 6: For the given abstraction granularity and benchmark, the number of samples (in log10) relative to the probe size (in log10) using iB-10.



Plot 6 shows the number of samples that were able to be drawn relative to the size of generated probes, thus providing an understanding of the speed abstractions occur. As expected, we notice the minVar scheme (which utilizes a computationally intensive hierarchical clustering process to abstract nodes) has the lowest sample efficiency. The other schemes have comparable abstraction speeds.

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`.

H. Kahn and A. W. Marshall. Methods of reducing sample size in monte carlo computations. *Journal of the Operations Research Society of America*, 1(5):263–278, 1953. ISSN 00963984. URL `http://www.jstor.org/stable/166789`.

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.

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`.

Robert Mateescu, Rina Dechter, and Radu Marinescu. AND/OR multi-valued decision diagrams (aomdds) for graphical models. *J. Artif. Intell. Res. (JAIR)*, 33:465–519, 2008. doi: 10.1613/jair.2605. URL `http://dx.doi.org/10.1613/jair.2605`.

Art B. Owen. *Monte Carlo theory, methods and examples*. `https://artowen.su.domains/mc/`, 2013.

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

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`.