Abstraction Sampling in Graphical Models

Rina Dechter
University of California, Irvine
Irvine, CA 92697
dechter@ics.uci.edu

Filjor Broka
University of California, Irvine
Irvine, CA 92697
fbroka@ics.uci.edu

Kalev Kask
University of California, Irvine
Irvine, CA 92697
kkask@ics.uci.edu

Alexander Ihler
University of California, Irvine
Irvine, CA 92697
ihler@ics.uci.edu

Abstract

We present a new sampling scheme for approximating hard to compute queries over graphical models, such as computing the partition function. The scheme builds upon exact algorithms that traverse a weighted directed state-space graph representing a global function over a graphical model. With the aid of an abstraction function and randomization, the state space can be compacted to facilitate tractable computation, yielding a Monte Carlo Estimate that is unbiased.

1 Introduction

Imagine that we want to compute a function over a weighted directed graph where the graph is given implicitly, e.g., using a generative state-space search model whose explicit state graph is enormous and does not fit in memory. Algorithms that need to traverse such a state-space graph will clearly fail. Knuth and Chen [7, 1] proposed a pioneering sampling scheme for estimating quantities that can be expressed as aggregates (e.g., sum) of functions defined over the nodes in the graph. They focused on estimating the number of nodes in such graphs [7]. Our work is inspired by their scheme and more recent work in the context of predicting search tree size [9, 8] and extends them to a more general class of functions defined over weighted directed graphs or trees. In particular, this extension is applicable to *reasoning tasks over graphical models e.g., probability of evidence or partition function* since they can be transformed into tasks over weighted directed state graphs [5].

The sampling scheme we will present uses an *abstraction function* that partitions the nodes in the state-space graph into subsets of *abstract* states under the intuition that nodes in the same abstract state represent similar subproblems and therefore a single member can represent all others. Given an abstraction function, our *abstraction sampling* algorithm generates a weighted directed subgraph $\tilde{G}$ from G using a generative randomized process. In particular, the scheme chooses randomly a single representative node from each encountered abstract state and associates it with a weight that estimates the total contribution of all such encountered states. An estimator to our query over G can then be computed over the generated representative graph $\tilde{G}$, which is supposedly far smaller. Clearly, if the number of abstract states is bounded, the generated graph size is small and the estimator can be computed efficiently. We call each sampled graph a *probe*. In this paper we will assume that the weighted state-space graphs are directed OR or AND/OR trees.

2 Background

A graphical model, such as a Bayesian or a Markov network [13, 3, 4] can be defined by a 3-tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, where where $\mathbf{X} = \{X_i : i \in V\}$ is a set of variables indexed by a set V and $\mathbf{D} =$

$\{D_i : i \in D\}$ is the set of finite domains of values for each X_i. Each function $\psi_\alpha \in \mathbf{F}$ is defined over a subset of the variables called its scope, X_α, where $\alpha \subseteq V$ are the indices of variables in its scope and D_α denotes the Cartesian product of their domains, so that $\psi_\alpha : D_\alpha \rightarrow R^{\geq 0}$. The **primal graph** of a graphical model associates each variable with a node and arcs connect nodes whose variables appear in the scope of the same local function. A graphical model represents a global function, often a probability distribution, defined by $Pr(X) \propto \prod_\alpha \psi_\alpha(X_\alpha)$. An important task is to compute the normalizing constant, also known as the partition function $Z = \sum_X \prod_\alpha \psi_\alpha(X_\alpha)$.

A graphical model can also be expressed via a weighted state space graph. In a simple OR search space, the states (or nodes) are partial assignments relative to a variable ordering, where each layer corresponds to a new assigned variable. A graphical model can also be transformed into a more compact AND/OR search space [5]. **AND/OR search spaces** provide a compact representation of all configurations by capturing conditional independence in the model, thus facilitating more effective algorithms [12]. The AND/OR search space is defined relative to a *pseudo tree* of the primal graph. A *pseudo tree* of an undirected graph $G = (V, E)$ is a directed rooted tree $\mathcal{T} = (V, E')$ such that every arc of G not in E' is a back-arc in $\mathcal{T}$ connecting a node in $\mathcal{T}$ to one of its ancestors.

Given a pseudo tree $\mathcal{T}$ of a primal graph G, the *AND/OR search tree* $T_\mathcal{T}$ guided by $\mathcal{T}$ has alternating levels of OR nodes corresponding to the variables, and AND nodes corresponding to an assignment from its domain with edge costs extracted from the original functions $\mathbf{F}$ [5]. Let s be a node in $T_\mathcal{T}$. We denote by $var(s)$ the last variable of the partial value assignment associated with s. So if s stands for $\bar{x}_{1..p} = (x_1, x_2, ..., x_p)$, then $var(s) = X_p$, in which case we say that *the variable of s is X_p*. Each AND node s (or the arc from its OR parent to the AND node) has a cost $c(s)$ defined to be the product of all factors ψ_α that are instantiated at s but not before. The size of the AND/OR search tree, $T_\mathcal{T}$ is exponential in the height of the pseudo tree.

We define a *solution subtree* $\hat{x}_M$ to be a subtree of $T_\mathcal{T}$ satisfying: (1) it contains the root of $T_\mathcal{T}$; (2) if an OR node is in $\hat{x}_M$, exactly one of its AND child nodes is in $\hat{x}_M$; (3) if an AND node is in $\hat{x}_M$ then all its OR children are in $\hat{x}_M$. The product of costs on any full solution tree equals the cost of a full configuration of the model $\mathcal{M}$. Each node s in $T_\mathcal{T}$ can be associated with a *value* $Z(s)$; for the partition function it expresses the conditioned partition function rooted at s. Clearly, $Z(s)$ can be computed recursively based on the values of the children of s: OR nodes by summation and AND nodes by multiplication. The value of $T_\mathcal{T}$ is the value of its root state, which is the partition function of the underlying model, $\mathcal{M}$. As noted, $Z(s)$ can be computed by a depth-first search scheme from leaves to root of the AND/OR tree . However, since $T_\mathcal{T}$ is normally too large to traverse we must resort to approximations. When the pseudo-tree is a chain we get back the regular OR tree, where each path corresponds to a full variable configuration.

3 Abstraction Sampling

Our proposed Abstraction Sampling is a Monte Carlo process that generates compact representatives $\tilde{T}_\mathcal{T}$ of $T_\mathcal{T}$, guided by an *abstraction function*. In [1] this function is called a stratifier, and it is inspired by the statistical technique of stratified sampling. We choose the term *abstraction* instead, because it inspires tapping into work on abstraction in AI [2].

Algorithm Given an abstraction function a, our scheme samples $\tilde{T}$ of T, where $root(T) = root(\tilde{T})$. Each sampled $\tilde{T}$, OR or AND/OR, is a tree defined over a subset of the same set of variables and represents a probability distribution $P_{\tilde{T}}(X)$ having a well defined $Z(\tilde{T})$. The idea is that in each probe of T we generate a compact subtree, representing several configurations. The *abstraction function* partitions the nodes in each level into equivalence classes, called *abstract states*, and we stochastically select a single representative from an abstract state. We will consider layered abstractions that put in the same abstraction state nodes that reside at the same level of the tree (nodes corresponding to the same variable). The abstract states are partially ordered in an order consistent with pseudo-tree (abstract states belonging to root variables are smallest).

AND/OR Abstraction Sampling (AOAS, Algorithm 1) builds a subtree $\tilde{T}_\mathcal{T}$ of $T_\mathcal{T}$, level-by-level, breadth first. At each step, it picks a leaf AND node having the smallest abstraction, and expands two levels down - child OR nodes, and their child AND nodes. Each seen AND node s is associated with a weight $w(s)$ representing the mass the abstract state stands for. The initial weight of the root node is a constant 1. For each of the newly generated AND nodes, the algorithm checks if there already exists an AND node with the same abstraction. If this is the case (line 11), it decides, with probability p proportional to $w(s)g(s)h(s)$ for each candidate node s, which of the representatives to keep and which to discard. $g(s)$ is the product of arc-costs from root to s, while $h(s)$ is a heuristic function approximating $Z(s)$. The weight of the remaining representative is adjusted to account for the discarded ones. Otherwise (line 21), if no AND node with the same abstraction exists, it adds the new AND node to $\tilde{T}_{\mathcal{T}}$ with the weight of its parent.

The sampling function incorporates a heuristic function h, that provides an upper bound on the partition function. While, as we will show, the algorithm is unbiased for any sampling probability, the heuristic yields a sampling function whose accuracy can significantly impact its convergence. Once a probe $\tilde{T}_{\mathcal{T}}$ is generated, its partition function can be computed in a depth-first manner to yield the estimate. The details of this last step are omitted for lack of space.

Proper Abstractions for AND/OR To guarantee the validity of our sampling scheme for general AND/OR trees (pseudo tree not necessarily a chain), and in particular its unbiasedness, we need to ensure that the sampled AND/OR tree would include only legitimate full solution trees of the underlying AND/OR tree. However, a brute-force application of the algorithm may generate probes that are not legitimate in that sense. We therefore require abstractions to be proper for AND/OR trees. Given a pseudo-tree $\mathcal{T}$, we call variable Y a *branching variable of X*, if Y is its closest ancestor in $\mathcal{T}$ that has at least 2 child nodes in $\mathcal{T}$. An abstraction is called *proper* if it assigns different abstract states to nodes ${s'}_j$ and s_j corresponding to variable X_j, whenever their ancestor nodes from the root to the branching variable of X_i correspond to different partial assignments. Clearly, any abstraction is proper for OR trees because there are no branching variables. Consequently, when sampling AND/OR trees we will always make sure to generate probes using a *proper* abstraction by simply imposing this restriction during the sampling process.

4 Properties of Abstraction Sampling

Unbiasedness We show that our sampling scheme generates an unbiased estimator of the partition function (proof omitted).

THEOREM **1** *Given a weighted directed AND/OR search tree T derived from a graphical model, a value function $Z(n)$ defined recursively over T, and a proper abstraction function over T, the estimate $\hat{Z}$, generated by AOAS, is unbiased.*

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

THEOREM **2 (complexity of proper AND/OR tree)** *Given a pseudo tree, the number of states in a probe by $AOAS$ is $O(n \cdot m^{b+1})$, where n is number of variables, b bounds the number of branchings along any path of the pseudo-tree and m bounds the number of states in the input abstraction function a per each variable (and level). For OR trees, $b = 0$, so size is bounded by $O(nm)$.*

Notice that when we have many branchings in the pseudo tree $\mathcal{T}$, the underlying AND/OR tree from which we sample is far more compact than the underlying OR tree. However, the AOAS probe size is easier to control on OR trees, as we find in our empirical evaluation.

Sampling Probabilities The proof of unbiasedness works for any sampling distribution p. The reason for choosing our specific sampling probabilities is to reduce variance.

THEOREM **3 (exact heuristics)** *If the sampling probability in AOAS uses an exact heuristic $h(n) = Z(n)$, namely if it satisfies $p = \frac{w(n')g(n')Z(n')}{w(n')g(n')Z(n')+w(n'')g(n'')Z(n'')}$, then $\hat{Z}$ is exact after a single probe (has zero variance), for any abstraction which is proper.*

We can define abstraction functions using symbolic properties. One example is *context-based abstractions*. The **context** of a variable in $\mathcal{T}$, denoted $C(X)$, identifies a subset of its ancestor variables, whose assignment uniquely determines the AND/OR subtree below the node [5]. An abstraction a is context-based, if given OR node (variable) X there exists $S \subseteq C(X)$ such that for any two child AND nodes s_1 and s_2 of X we have: $a(s_1) = a(s_2) \leftrightarrow \pi_S C(s_1) = \pi_S C(s_2)$. If $|S| = j$ the abstraction is said to be a j-level context-based abstraction. In particular, a 0-level abstraction puts all AND nodes of a variable in a single abstract state.

Table 1: Mean error aggregated over benchmark for a given tree type, abstraction level (j) and time. OR: $j \in \{0, 4, 8\}$. AO: $j \in \{0, 1, 2\}$. (#inst, $\bar{n}$, $\bar{w}$, $\bar{k}$, $|\bar{F}|$, $\bar{s}$) are number of instances and averages of number of variables, induced width, max domain size, number of functions, max scope size.

Benchmark	tree	#nodes/probe	1 min	20 min	60 min
#inst, $\bar{n}$, $\bar{w}$, $\bar{k}$, $|\bar{F}|$, $\bar{s}$					
DBN	OR	818, 7452, 117175	7.80, 6.00, **1.89**	6.20, 4.49, **0.99**	5.26, 3.69, **0.74**
47, 750, 59.3, 2, 14848, 2					
Grids	OR	179, 2764, 42202	7.31, 5.79, **4.72**	6.25, 4.84, **4.20**	5.24, 4.09, **3.66**
7, 271, 24.3, 2, 791, 4	AO	215, 7169, 1290451	7.20, 5,16, **3.41**	6.18, 4.58, **2.90**	4.87, 3.75, **2.20**
Linkage	OR	274, 6501, 200549	**0.38**, 0.57, 0.70	0.30, **0.29**, 0.39	**0.22**, 0.24, 0.28
17, 950, 29.5, 4.9, 950, 4	AO	286, 15353851, -	**0.31**, 0.57, -	**0.23**, 0.37, -	**0.21**, 0.26, -
Promedas	OR	104, 1010, 12707	**1.51**, 1.81, 2.36	**0.91**, 1.15, 1.24	**0.54**, 0.65, 0.61
8, 720, 28.0, 720, 3	AO	93, 25117, -	**1.44**, 1.91, -	**0.96**, 0.95, -	**0.36**, 0.72, -

5 Empirical Evaluation and Future Work

Methodology We evaluate our algorithm on instances from 4 benchmarks (DBN, Grids, Linkage, Promedas), for which we have exact values of the partition function, by running experiments using different configurations abstraction functions for 1 hour each. We implemented the algorithm in C++ and ran experiments on a 2.66 GHz processor with 4GB of memory. We use Weighted Mini-Bucket Elimination (WMBE) [6, 11] heuristic function, whose strength is controlled by a parameter called the i-bound. Higher i-bounds lead to stronger heuristics at the expense of higher computation and memory cost. We use i-bound 10 in our experiments. Abstraction sampling can be augmented on top of any importance sampling scheme, while our current choice of proposal is closely related to a state-of-the-art proposal based on weighted mini-bucket that ensures reasonably bounded importance weights, reducing variance [10].

As abstraction function we use context-based abstraction, parametrized by the level. We compare higher level abstractions to 0-level one corresponding to a baseline regular importance sampling scheme, with the above mentioned proposal. Higher level abstractions have increasing number of abstract states, thus leading in general to higher number of nodes expanded during sampling. For OR trees (using a fixed variable order) we experiment with abstractions of level 4 and 8, while for AND/OR trees (using a fixed pseudo tree) we experiment with levels 1 and 2 (probe size in AND/OR trees increases faster with level due to properness restriction). Since DBN instances have pseudo-trees which are chain-like (basically OR), we test only on OR trees for that benchmark. For Linkage and Promedas benchmarks some of the problem instances do not generate a probe in the time limit for level 2, due to the large probe size, so respective aggregate results are missing.

Performance Measure For each problem instance and experiment configuration (tree type, i-bound, abstraction level) we execute 11 independent 1 hour experiments, and record the partition function estimate $\hat{Z}$ at different time steps. We compute the median estimate $\hat{Z}_m$ over the 11 runs, and compute the log partition function absolute error $|\log_{10} \hat{Z}_m - \log_{10} Z|$. We present the mean of these errors aggregated by benchmark in Table 1.

Results We notice that for benchmarks where regular importance sampling (0-level) has large errors (DBN and Grids), abstraction sampling with high level abstraction shows much stronger performance (e.g for DBN OR after 60 min we have a reduction in error of more than 4 orders of magnitude). In Grids we notice that this performance gets even stronger when we move to AND/OR. In comparing OR and AND/OR at 0-level (where each probe consists of one configuration), we see that AND/OR outperforms OR across benchmarks, potentially due to computational benefits of exploiting subproblem independence. For the Linkage and Promedas benchmarks, where regular sampling already performs well (small errors), we see that we are not able to improve performance by moving to higher level abstractions. In Figure 1 in the Appendix, we also show selected graphs of convergence patterns, supporting the findings of the aggregate results.

Future work We plan to explore different abstraction function families in order to identify properties of strong-performing abstractions. We also are considering the extension of the existing algorithm from AND/OR trees to AND/OR graphs [5] to benefit from sampling on smaller search spaces.

Acknowledgments

This work was supported in part by NSF grants IIS-1526842 and IIS-1254071.

References

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

[2] J. C. Culberson and J. Schaeffer. Pattern databases. In *Computational Intelligence*, pages 318–334, 1998.

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

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

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

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

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

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

[9] Levi H. S. Lelis, Sandra Zilles, and Robert C. Holte. Predicting the size of IDA*'s search tree. *Artificial Intelligence*, 196(Supplement C):53 – 76, 2013.

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

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

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

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

Appendix

Algorithm 1: AOAS, a single probe

```
Require: A graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$ over $X = \{X_1, ..., X_n\}$, a pseudo tree $\mathcal{T}$. An
    implicit AND/OR tree $T_\mathcal{T}$ of $\mathcal{M}$. $c(s)$ is the cost of an OR-to-AND arc $(parent(s), s)$ in $T_\mathcal{T}$.
    $g(s)$ is the product of arc-costs from root to $s$ and $h(s)$ (heuristic function) An abstraction $a$.
    $X(s)$ denote the variable of node $s$
Ensure: A sampled subtree $\tilde{T}_\mathcal{T} = (\tilde{N}, E, C)$ of $T_\mathcal{T}$. Each $n \in \tilde{N}$ is a pair $n =< s, w(s) >$ where
    $w(s)$ is a weight. Note that OR node weight is always 1.
 1: initialize $\tilde{T}_\mathcal{T} \leftarrow \{< s, 1 >\}$,
 2: while $OPEN$ is not empty do
 3:   $< s, w(s) > \leftarrow$ remove node in OPEN with smallest abstraction $a$
 4:   Expand $s$, generating all its OR child nodes $ch(X(s)) \in \mathcal{T}$ denoted variables $\{X_1, ...X_r\}$,
      each yielding OR nodes $s_1, ..., s_r$ $(X(s_j) = X_j)$ and add them to $\tilde{T}_\mathcal{T}$.
 5:   for each OR child node $s_j$ do
 6:     expand it, generating all its AND child nodes $s_{j_i} =< X_j, x_{j_i} >$, $x_{j_i} \in D_{X_j}$ with weights
        $w(s_{j_i}) = w(s)$.
 7:     for each child $s_{j_i}$ do
 8:       $\{k\} \leftarrow a(s_{j_i})$
 9:       if $\tilde{T}_\mathcal{T}$ contains a representative $< s_{\{k\}}, w_{\{k\}} >$ of abstraction $\{k\}$ then
10:         $p \leftarrow \frac{w(s_{j_i})g(s_{j_i})h(s_{j_i})}{w(s_{j_i})g(s_{j_i})h(s_{j_i})+w_{\{k\}}g(s_{\{k\}})h(s_{\{k\}})}$
11:         with probability $p$ do:
12:           remove $s_{\{k\}}$ from $\tilde{T}_\mathcal{T}$ and OPEN
13:           add $< s_{j_i}, \frac{w(s_{j_i})}{p} >$ as a child of $s_j$ in $\tilde{T}_\mathcal{T}$ representing $\{k\}$ and add it to OPEN
14:         else
15:           $w_{\{k\}} \leftarrow \frac{w_{\{k\}}}{1-p}$
16:       else
17:         add $< s_{j_i}, w(s_{j_i}) >$ as a child of $s_j$ in $\tilde{T}_\mathcal{T}$ representing $\{k\}$ and add it to OPEN.
18: $\tilde{T}_\mathcal{T}$ is the final tree generated.
19: return $\hat{Z} \leftarrow$ compute by depth-first search the $Z$ value over the final tree $\tilde{T}_\mathcal{T}$ generated.
```

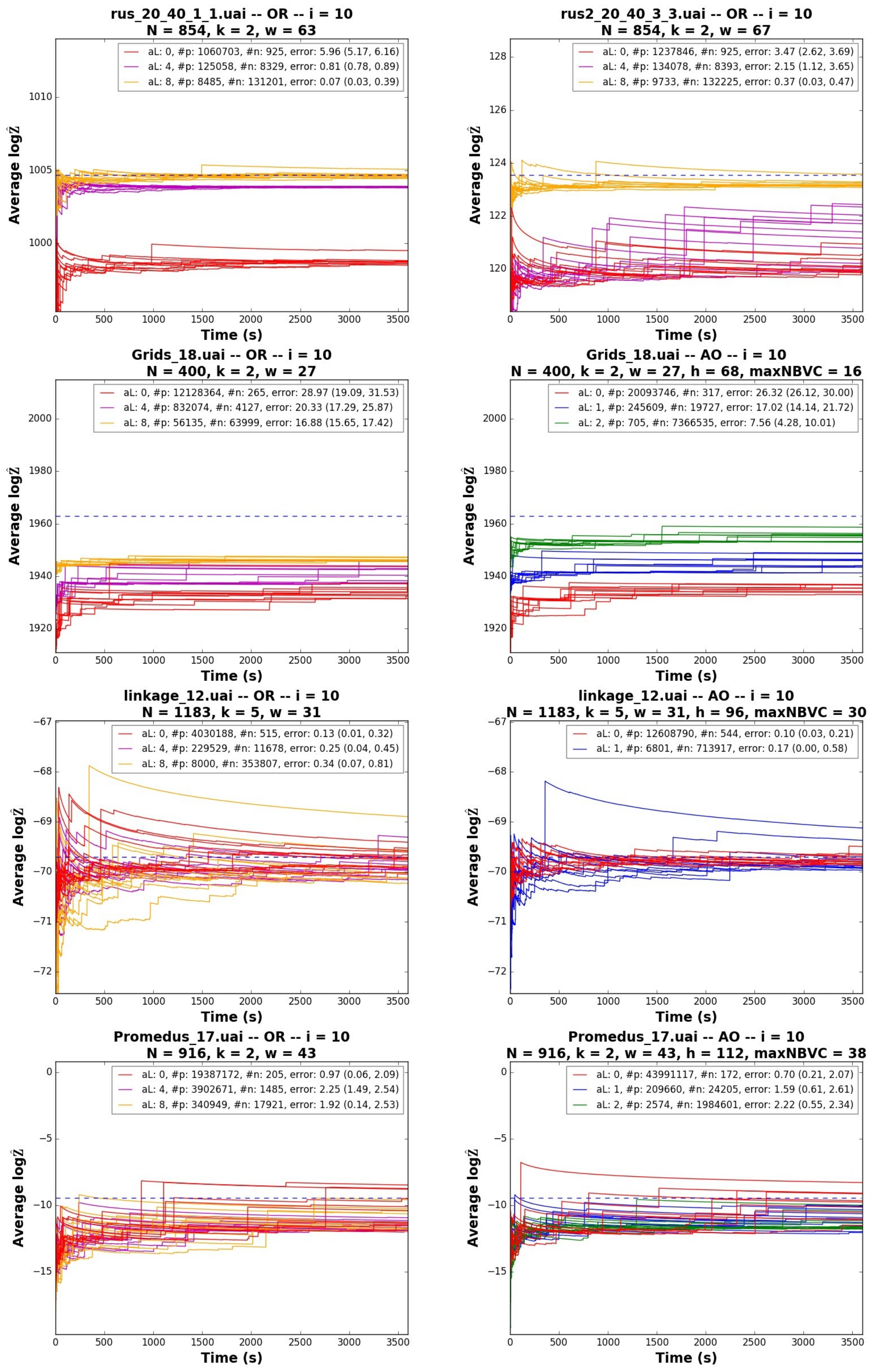

Figure 1: Convergence Patterns for Different Abstraction Levels (aL) for Selected Problem Instances. $\#p$ – number of probes, $\#n$ – average number of expanded nodes per probe, h – height of pseudo tree, maxNBVC – max number of branching variables in any path of pseudo tree, error – showing median error over 11 runs and error range in parentheses