Bucket Errors in MBE Trees

William Lam

June 23, 2015

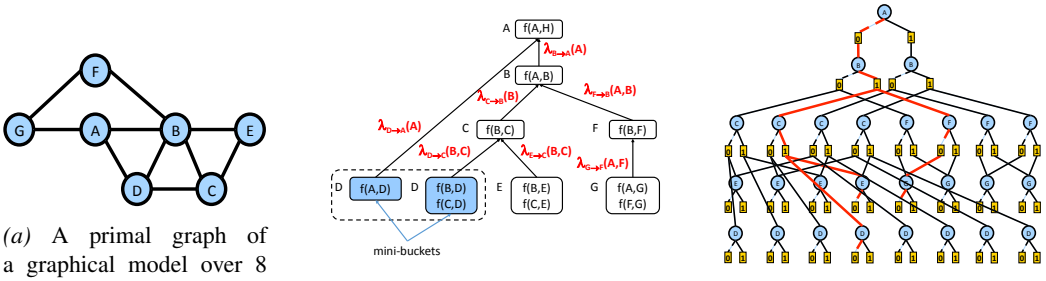
1 Introduction

We study here the *bucket error* associated with the partitioning in the bucket trees generated by mini-bucket elimination. The notion of bucket error was introduced in [2] and was shown to coincide with the 1-level look-ahead residual. In this report, we present statistics on the distributions of the error in the bucket tree to better our understanding of where errors occur. Our goal is to improve the MBE look-ahead heuristic's performance as well as gaining an understanding of the behavior of search schemes which also tighten the lower-bound (assuming the minimization task).

2 Background

A *graphical model* is a 4-tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \otimes)$, 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. $\mathbf{F} = \{f_\alpha : \alpha \in F\}$ is a set of discrete functions, where $\alpha \subseteq V$ and $X_\alpha \subseteq X$ is the scope of f_α. The functions' scopes imply a *primal graph* G where each variable X_i is a node and an edge (X_i, X_j) is in G iff the pair of variables appears in the scope of any f_α. $\otimes = \{\prod, \sum, \bowtie\}$ is a combination operator that defines the function represented by the graphical model $\mathcal{M}$ as $\otimes_{\alpha \in F} f_\alpha(X_\alpha)$.

While our work is applicable to a wide range of reasoning problems in graphical models, we focus here on the *min-sum problem*, $\mathcal{M} =< X, D, F, \sum >$, which is to compute $C^* = \min_{\mathbf{X}} \sum_{\alpha \in F} f_\alpha(X_\alpha)$, and the assignment that achieves this minimum. It is easy to show that the MPE/MAP task - finding an assignment to X that maximizes the probability distribution - can be reduced to the min-sum problem using a -log transformation.



(a) A primal graph of a graphical model over 8 variables. We assume pairwise functions.

(b) Mini-bucket tree for the model with i-bound of 3.

(c) The AND/OR context minimal search graph.

Figure 1: Graphical model, mini-bucket tree, and AND/OR search graph.

Bucket elimination (BE [1]) solves the min-sum problem by eliminating variables, one at a time, in sequence. It works on a structure called *pseudo-tree* (also known as bucket-tree), T. The nodes of the pseudo-tree are associated with *buckets*, each containing a set of variables and a set functions of the problem such that: (1) Each function must be assigned to a single bucket, in which case all its variables must also belong to that bucket, (2) for any two buckets, B_u and B_v, if X belongs to B_u and B_v, it must also belong to any bucket on the path in T between B_u and B_v (known as the running intersection property).

BE works by traversing the pseudo-tree from leaves towards the root, processing each bucket along the way, by combining all functions in the bucket (by sum in our case) and eliminating the bucket's variable (by min in our case). The resulting new function, also called a message, is placed in the ancestor bucket that is closest and contains a variable in the new function's scope. The complexity of BE is time and space exponential in *induced width* w^* of the underlying primal graph of the problem [1].

Mini-Bucket Elimination $MBE(i)$ is an approximation algorithm designed to avoid the time and space complexity of the full Bucket Elimination. It differs from BE in that, when processing a bucket, its functions are partitioned into mini-buckets, each containing no more than i variables (the i-bound). Subsequently, a message is generated from each mini-bucket using the same combination and elimination as in BE. The output of a bucket is not a single function, but rather a set of functions, one per mini-bucket, each placed in an ancestor bucket using the same rule as in BE.

DEFINITION **1 (The mini-bucket messages at B_Y)** *Assuming a set of mini-buckets MB_Y^r, when r indexes the mini-buckets of B_Y, we define the combined mini-bucket message at B_Y to be μ_Y*

$$\mu_Y = \sum_{r,\beta \in anc(Y)} \lambda_{Y\to\beta}^r = \sum_r min_Y \left[\psi_Y^r + \sum_{r,\alpha \in desc(Y)} \lambda_{\alpha\to Y}^r \right] \tag{1}$$

(The left equality is written in terms of messages generated in mini-buckets MB_Y^r while the right part shows explicitly the computation).

In contrast, the exact message that would have been generated with no partitioning at B_Y denoted as μ_Y^* is

$$\mu_Y^* = min_Y \left[\psi_Y + \sum_{\alpha \in desc(Y)} \lambda_{\alpha\to Y} \right] \tag{2}$$

Notice that μ_Y^* is not necessarily the output function generated by the exact Bucket-Elimination algorithm at B_Y because while μ_Y^* is exact for B_Y it contains partitioning errors introduced in earlier buckets.

DEFINITION **2 (The local bucket-error at B_Y)** *Let Y be a child of X_p in T. Given a run of MBE, the local bucket error function at B_Y denoted $Err_Y(s_p)$*

$$Err_Y(s_p) = \mu_Y^*(s_p) - \mu_Y(s_p) \tag{3}$$

Briefly, the local bucket-error at Y considers all the functions residing in B_Y as a new subproblem. It then takes the min-sum of all of them, generating function μ^*, and then, given the current partition into mini-buckets it sum all the min-sum functions created in each mini-bucket. This is μ_Y. The difference between these two functions computed *within* B_Y is its *bucket-error*.

Algorithm 1: Bucket Error Evaluation (BEE)

Input: A Graphical model and a pseudo-tree, $M = \langle X, D, F, \sum, min \rangle$, $X = \{X_1, ..., X_n\}$, $F = \{f_1, ..., f_r\}$. i-bound, r index mini-buckets

Output: The error function for each bucket

1 **Initialization:** Run the MBE(i) algorithm from leaves to root on the pseudo-tree T.
2 **for** *each* $B_Y, Y \in X$ **do**
3 $\quad \mu_Y \leftarrow \sum_r min_Y[\psi_Y^r + \sum_{j_r, \alpha \in desc(Y)} \lambda_{\alpha \to Y}^{j_r}]$
4 $\quad \mu_Y^* = min_Y[\psi_Y + \sum_{\alpha \in desc(Y)} \lambda_{\alpha \to Y}]$
5 $\quad Err_Y \leftarrow \mu_Y^* - \mu_Y$
6 **return** Err functions

An algorithm for computing the local bucket error (BEE) is given in Algorithm 1. Computing μ_Y (line 3) takes very little time since all λ functions are computed by $MBE(i)$ beforehand, as we assume it is already executed. Therefore the computation is dominated by μ_Y^*. This operation is identical to exact processing of bucket Y by min-sum operators over all its functions, when we eliminate Y by minimization, thus the complexity is exponential in the number of variables in B_Y upon $MBE(i)$ termination. This number is captured by a parameter called *pseudo-width* of a bucket.

3 Evaluating the Error of a Bucket with BEE

We propose different metrics evaluate the error of a bucket with BEE. The first metric is to use the Err_Y function directly in 3, which gives the *absolute error*. An alternative is the *relative error*, which gives a percentage of the amount that the mini-bucket message μ_Y gives compared to the exact message μ_Y^*.

DEFINITION **3 (The local relative bucket-error at B_Y)** *Let Y be a child of X_p in T. Given a run of MBE, the local relative bucket error function at B_Y denoted $RelErr_Y(s_p)$*

$$RelErr_Y(s_p) = \frac{Err_Y(s_p)}{\mu_Y^*(s_p)} \tag{4}$$

The relative error serves as a way to normalize errors such that the measure is not sensitive to the values of the functions. It is not certain which is preferable and the data presented in the following section provides a start to the investigation.

Since we would like to have a scalar value representing the error of a bucket Y, we can take the average, variance, and maximum of each of the absolute and relative errors. In our current work [2], we use the average relative error as a measure and use a threshold against this value to decide whether a bucket should be included in look-ahead.

4 Experiments: Summary Statistics

We evaluated on 3 benchmarks: pedigree, grids, and pdb. For each instance, we ran BEE with a maximum table enumeration limit of 10^6. If an error function has more than that number of values, then we sample up to 10^6 values instead. In addition, we tested a range of different i-bounds for each instance. For the pedigree and grid instances, we tested with i-bounds ranging from 2 to

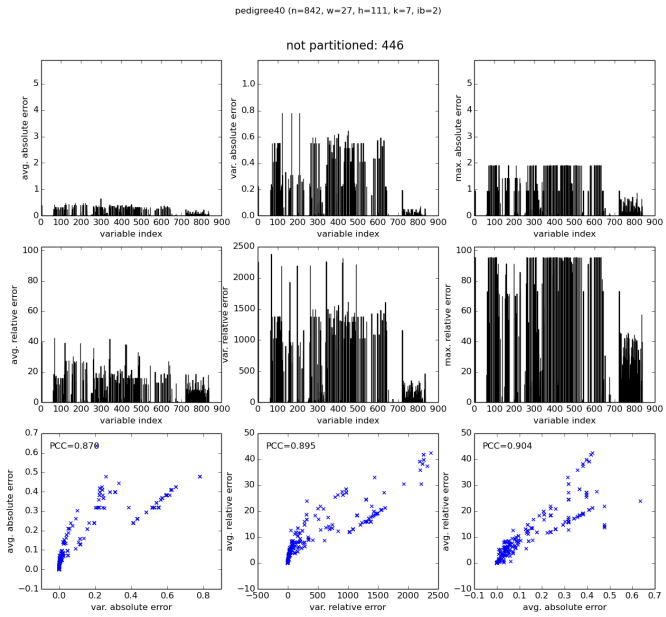


Figure 2: pedigree40 errors, i-bound 2

18, in increments of 2. For the pdb instances, we tested with i-bounds ranging from 1 to 5. For each problem and i-bound setting, we plot the variable indices against the average, variance, and maximum for each of the two error measures. The first row of each figure corresponds these three statistics for the absolute errors. The second row of each figure is the same, but for the relative errors. In addition, to evaluate the differences between these, we plot on the last row, the variance against the average for the absolute and relative errors and the average absolute error against the average relative error. Each of these is annotated with the Pearson's correlation coefficient.

The appendix contains figures giving an alternative view, showing the average absolute and average relative errors of different i-bounds in a single page per problem instance.

In this report, we report on only a few instances; a full set of the data can be found at `http://graphmod.ics.uci.edu/~wlam/bee-plots/`.

4.1 Case Study: pedigree40

This is the hardest instance in the benchmark based on the runtime taken to prove an optimal solution using the AND/OR Branch and Bound with the MBE-MM heuristic. Figures 2 through 6 show the results for i-bounds of 2, 6, 10, 14, and 18. In the plots, we see that the maximum errors for both the absolute and relative errors tend to be much higher than the average error. For example, in Figure 6, the maximums can be up to 20 times larger than the averages. This indicates that for at least one state, the approximation error is large, but is small for most the other states. In addition, as the average is sensitive to outliers, the average and maximum are highly correlated (not shown in a plot).

In the scatter plots showing the variance against the average for each of the error measures, the correlation is quite high (0.8137 on average for absolute error and 0.8314 on average for relative error, across all of the i-bounds), so this indicates that the difference of using the average versus

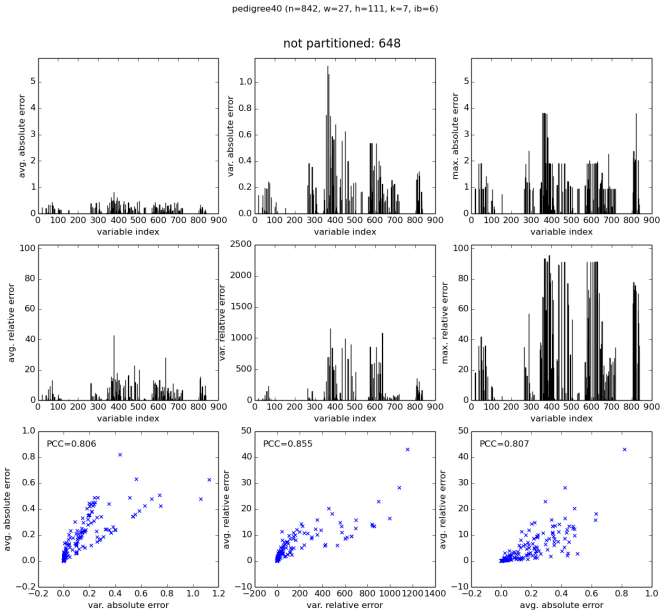


Figure 3: pedigree40 errors, i-bound 6

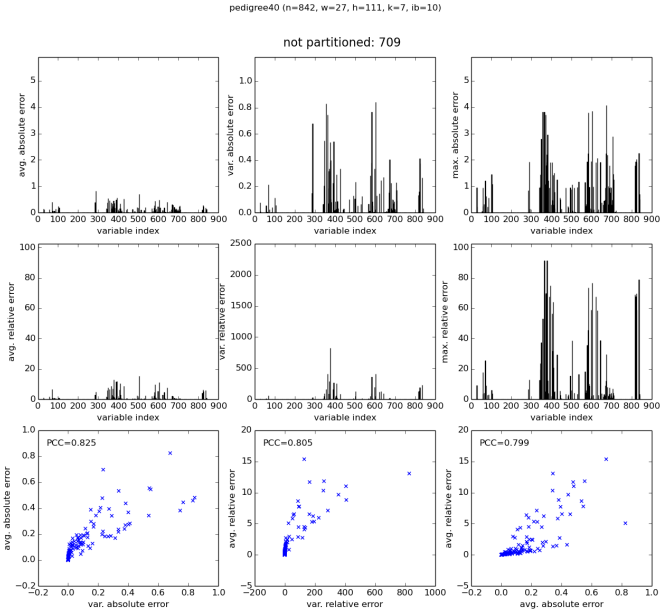


Figure 4: pedigree40 errors, i-bound 10

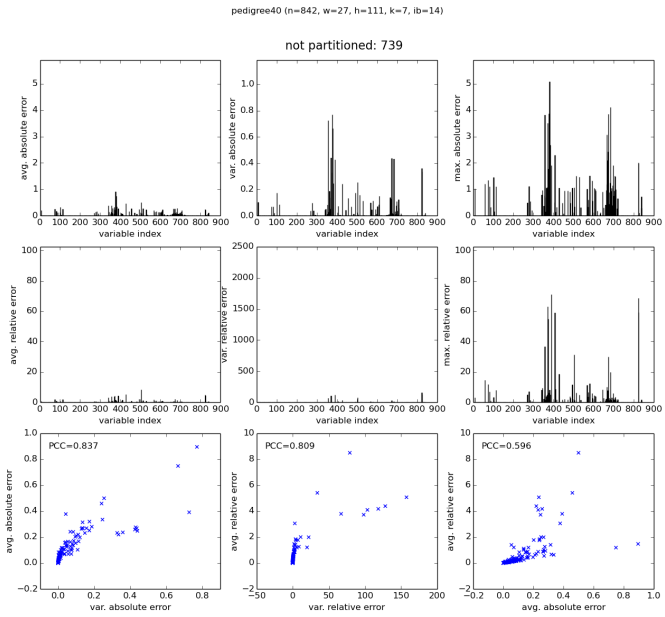


Figure 5: pedigree40 errors, i-bound 14

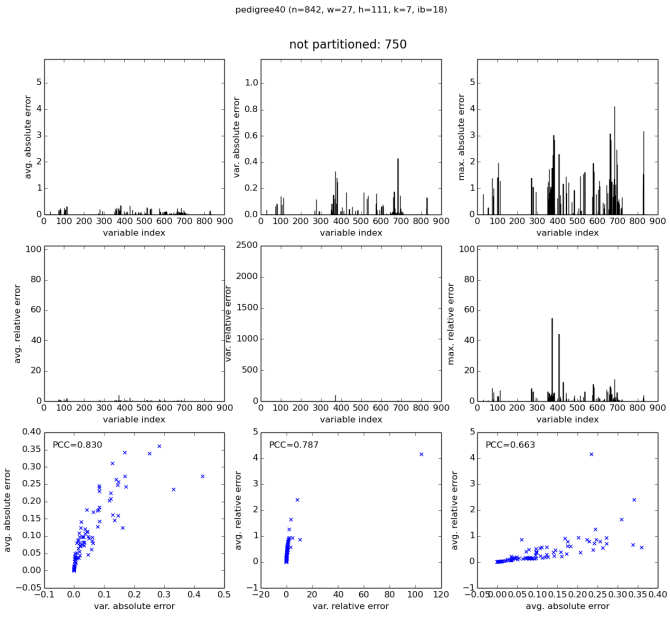


Figure 6: pedigree40 errors, i-bound 18

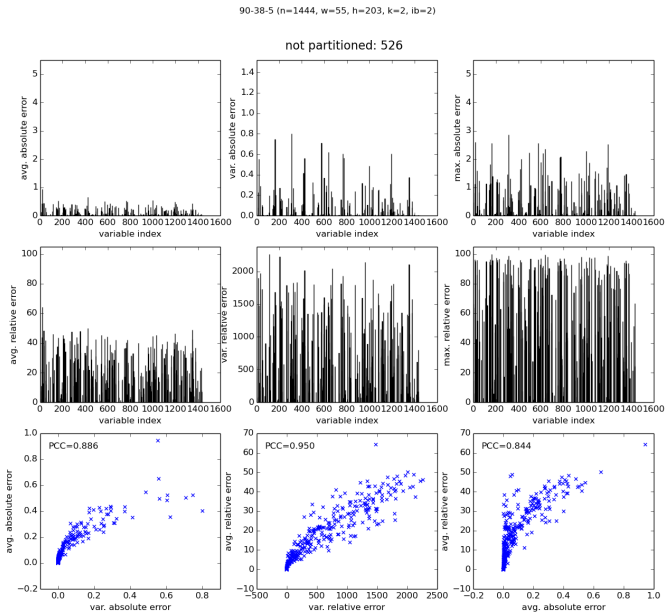


Figure 7: 90-38-5 errors, i-bound 2

the variance may not be great, given some threshold for each. Intuitively, we considered that the variance could be used as a way to detect cases where we could change value ordering through look-ahead, but this high correlation may suggest otherwise.

When computing the PCC between the absolute and relative errors, we observe here that it is high when the i-bound is low, but generally decreases as the i-bound increases. This indicates that the normalizing effect of the relative error does not matter as much when all of the absolute errors are high at the low i-bounds. When the i-bound is high (Figure 6), we see that the relative error is very similar across the buckets, but the absolute error differs. This indicates that the difference between the two measures should be investigated, but only when the i-bound is close to the induced width. The absolute error may have more power to discriminate the quality of the bucket.

4.2 Case Study: Grid 90-38-5

Here we choose a representative grid instance with an induced width that is considerably higher than the highest i-bound we tested ($w = 55$) compared to the pedigrees. Figures 7 through 11 show the results for i-bounds of 2, 6, 10, 14, and 18. We observe the same behavior where the maximum is much higher than the average for both error measures and can make the same line of reasoning as before concerning outlier states with respect to the error.

The scatter plots plotting the correlation are overall similar where we see that the correlations are also quite high (0.8658 on average for absolute error and 0.9227 on average for relative error, over the i-bounds).

Finally, looking at the PCC between the absolute and relative errors, the PCC is overall high at 0.8430 and there is little correlation between the i-bound and the PCC. However, we notice that on low i-bounds (Figure 7), there are many buckets for which the absolute error is very similar, while

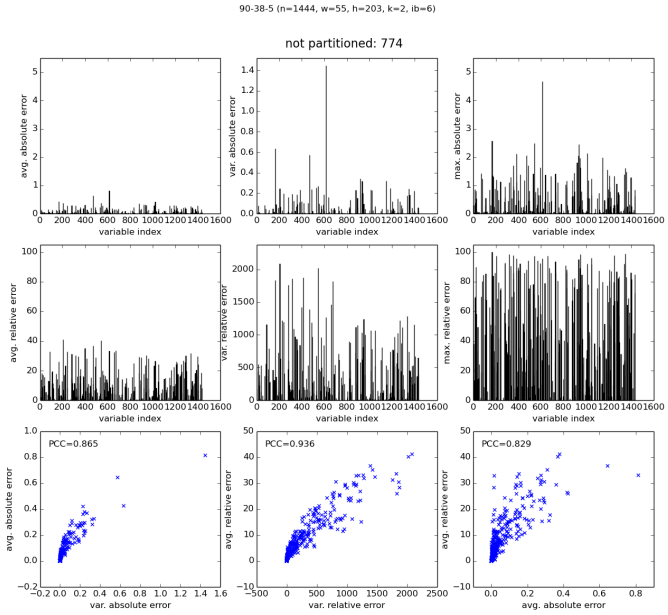


Figure 8: 90-38-5 errors, i-bound 6

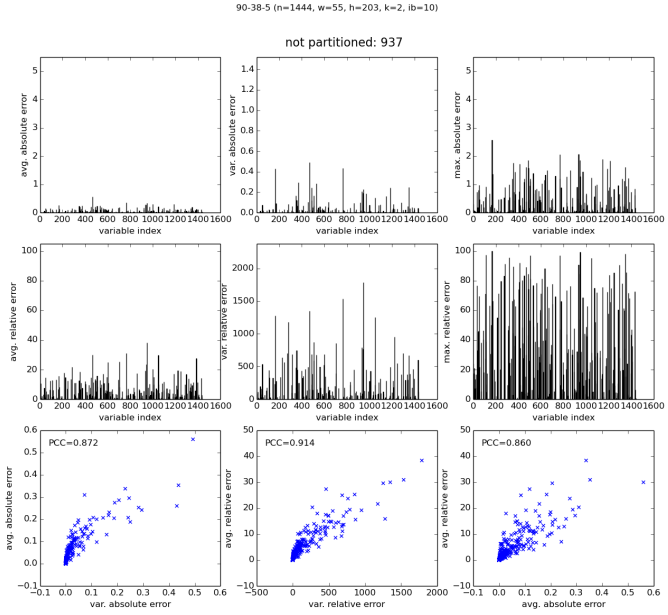


Figure 9: 90-38-5 errors, i-bound 10

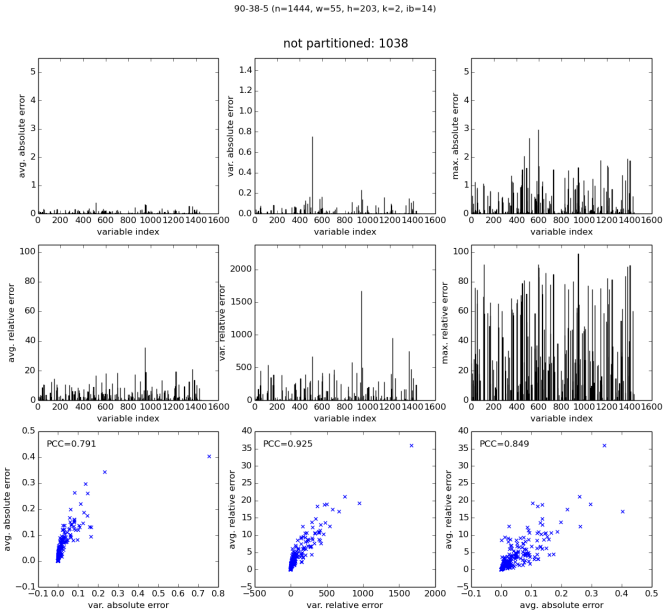


Figure 10: 90-38-5 errors, i-bound 14

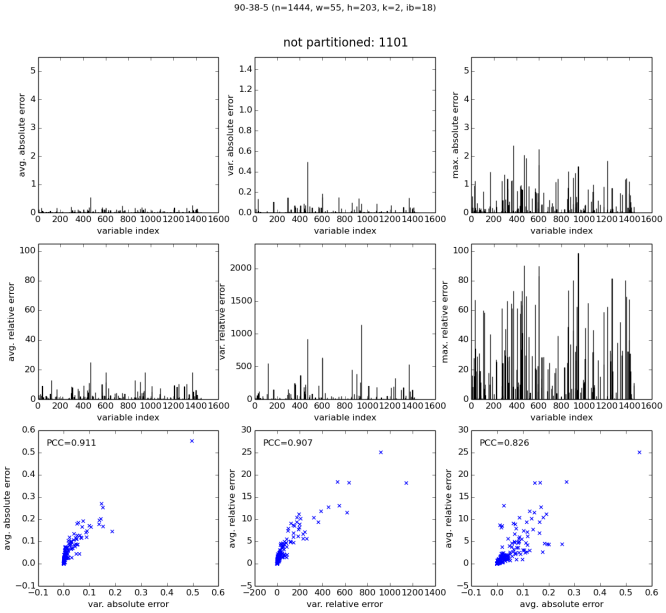


Figure 11: 90-38-5 errors, i-bound 18

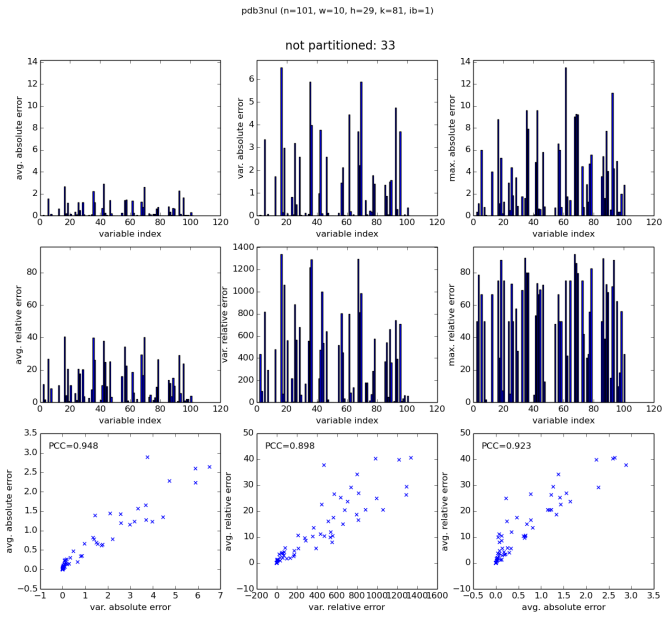


Figure 12: pdb3nul errors, i-bound 1

the relative error can be much different. This is contrary to what we see in pedigree40. So therefore, although, the correlation is high, the relative error may fare better at discriminating error in buckets here.

4.3 Case Study: Protein pdb3nul

The pdb (protein side-chain prediction) benchmark is different from the other two benchmarks we evaluated on in that it has relatively lower induced width, but an extremely high domain size. As a result, we tested i-bounds of 1 through 5 instead, due to memory constraints. This results in table sizes for the error functions that are similar in size to those of the other two benchmarks. We show the plots in Figures 12 through 16. The maximum is much higher than the average on these instances as well.

The variance-average correlations are also quite high (0.9661 on average for absolute error and 0.9156 on for relative error, over the i-bounds).

For the PCC between the absolute and relative errors, the PCC is overall high at 0.8849, but we see that it does decrease as the i-bound increases. In terms of the ability to discriminate error, we see that at low i-bounds, the relative error varies for the many buckets with low absolute error. However, as the i-bound increases, the absolute and relative error are consistently low at the low errors and vary equally at higher errors.

4.4 Discussion

Overall, we always see that the maximum errors are significantly higher than the average errors. This indicates that there are certain assignments where the error correction is significantly higher.

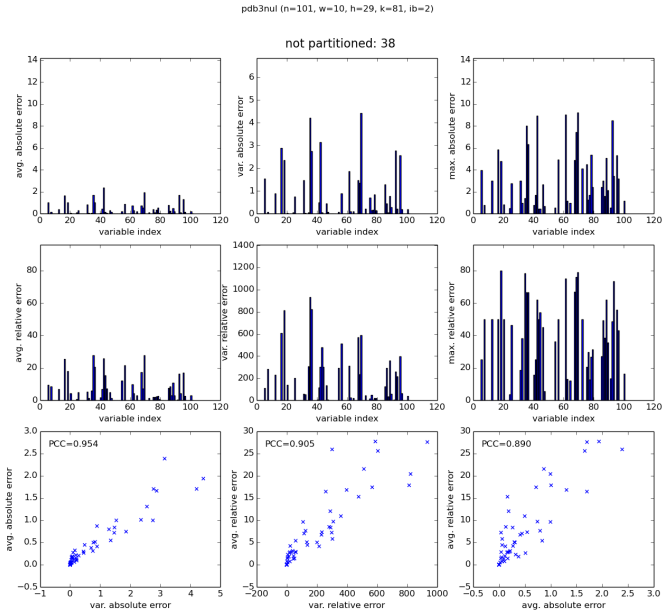


Figure 13: pdb3nul errors, i-bound 2

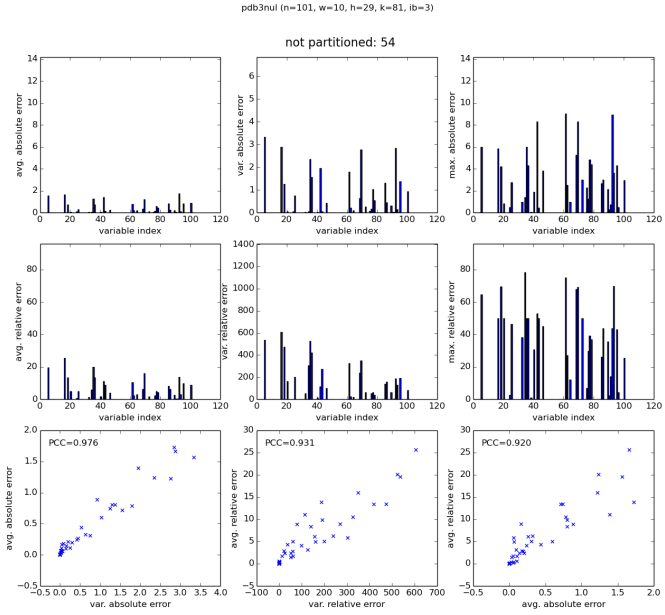


Figure 14: pdb3nul errors, i-bound 3

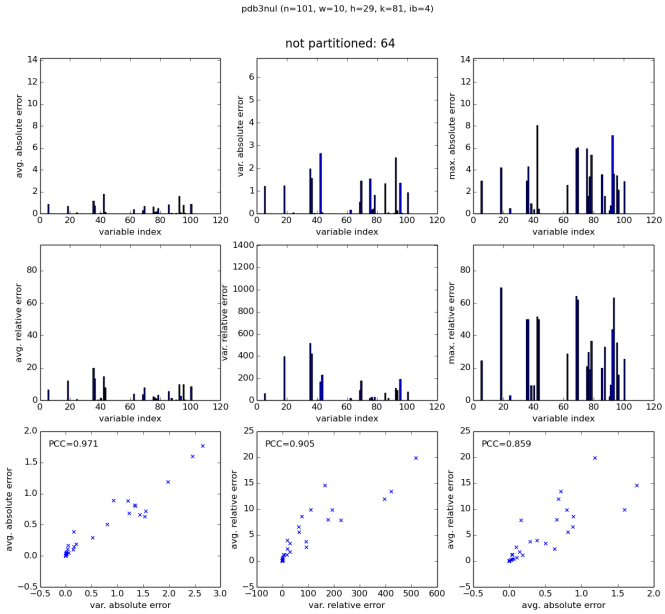


Figure 15: pdb3nul errors, i-bound 4

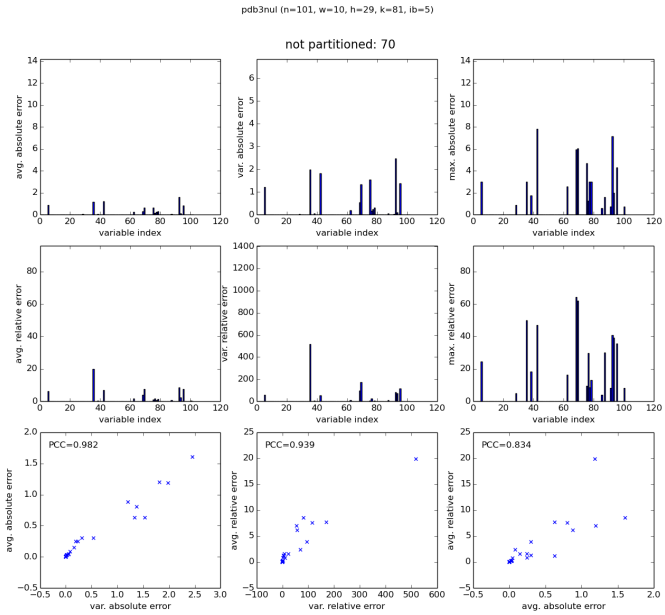


Figure 16: pdb3nul errors, i-bound 5

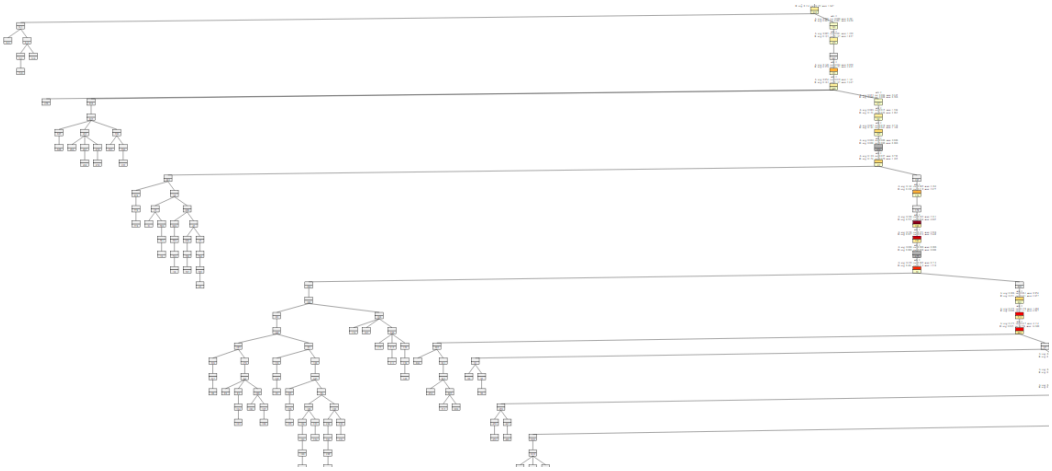


Figure 17: Extracted structure from pseudotree showing errors for pedigree40 with an i-bound of 18.

This raises the question of how often we would request these assignments on heuristic evaluation. Since the average is heavily influenced by the maximum, we would like to be sure that we are not weighing in assignments that are less likely used in the heuristic computation. This would result in a more accurate measure. If the maximums are likely used, then our current metric is probably sufficient.

We see that the variance and average on both metrics are highly correlated, which indicates that it may not make much of a difference whether we use the variance or average.

Lastly, the behavior of the absolute and relative errors vary over the benchmarks. It may be interesting to choose which one is preferable to use based on how well it discriminates values.

5 Experiments: Location of Errors

The previous section does not address the *location* of the errors in the problems. In this section, we also show extracted portions of the pseudotree for the highest i-bound used. White nodes indicate there is no partitioning. Gray nodes indicate that there was partitioning, but the errors was found to be zero. For nodes with errors, they are colored with a gradient from yellow to dark red, with the latter end of the spectrum indicating higher error.

The full set of plots can be found at `http://graphmod.ics.uci.edu/~wlam/pseudotree-errors/`.

5.1 Case Study: pedigree40

We show in Figure 17 an extracted portion from its pseudotree annotated with errors given with an i-bound of 18. (The full pseudotree is too large to show and can be viewed at `http://graphmod.ics.uci.edu/~wlam/pseudotree-errors/pedigree/pedigree40_LH(1)-MBEMM-i18.svg`.) Here, we extracted a portion of the tree containing a sequence of nodes with errors (see the right side). We see from this that errors tend to accumulate along these, which are the separators of the tree decomposition. At the same time, on the left side, the decomposed subproblems have zero partitioning, and therefore, zero error. This

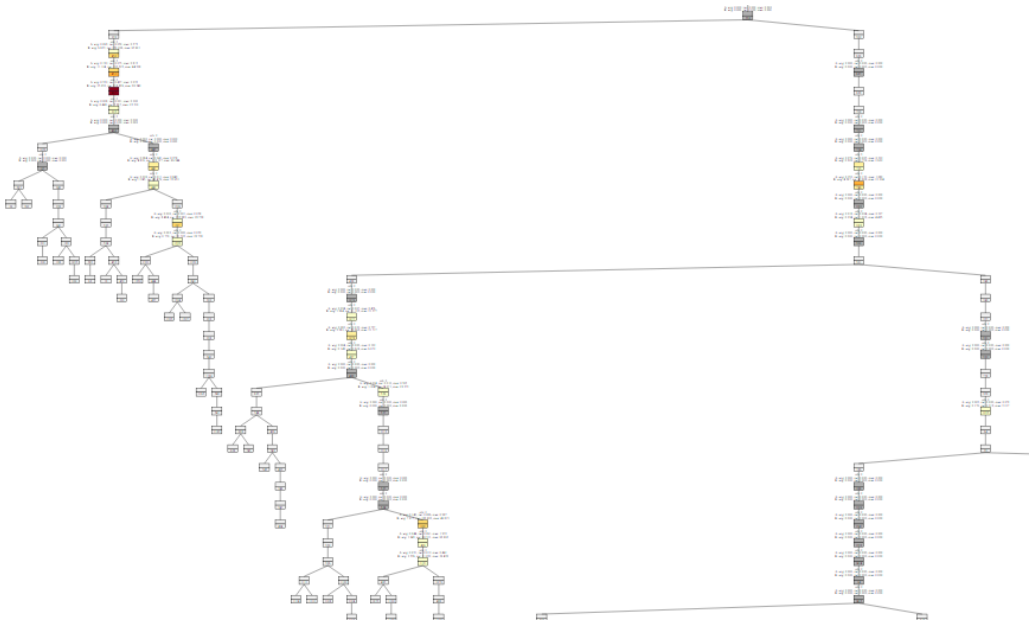


Figure 18: Extracted structure from pseudotree showing errors for 90-38-5 with an i-bound of 18.

remains the pattern down to the last decomposition down the chain of nodes with errors. This is the result of the min-fill ordering correctly choosing to eliminate nodes which induce the fewest edges greedily and this results in bucket elimination working well since it solves many easy subproblems.

5.2 Case Study: Grid 90-38-5

We show in Figure 18 an extracted portion from its pseudotree annotated with errors given with an i-bound of 18. (The full pseudotree is too large to show and can be viewed at `http://graphmod.ics.uci.edu/~wlam/pseudotree-errors/grids/90-38-5_LH(1)-MBEMM-i18.svg`.) Here, we also see that partitioning occurs in the same fashion along chains corresponding to separators of the tree decompostion. However, many of the nodes are gray (indicating no error), which demonstrates much computation can be saved when performing look-ahead. For decomposed subproblems at the leaves, we also see that there is no partitioning.

5.3 Case Study: Protein pdb3nul

We show in Figure 19 its pseudotree annotated with errors given with an i-bound of 5. In this instance, the graph is denser, and therefore partitioning occurs at the leaves as well. We do observe here that many of the nodes are gray (indicating no error), once again showing that much computation can be saved for look-ahead here. In particular, we only see two small groups of nodes with significant error, where maximum path length of these nodes is 2. For this instance, it is good since the domain size is 81, and the look-ahead computation does not scale well as the depth increases. Since the maximum path length of the nodes with significant error is 2, we know we are doing the best we can with a look-ahead depth of 2.

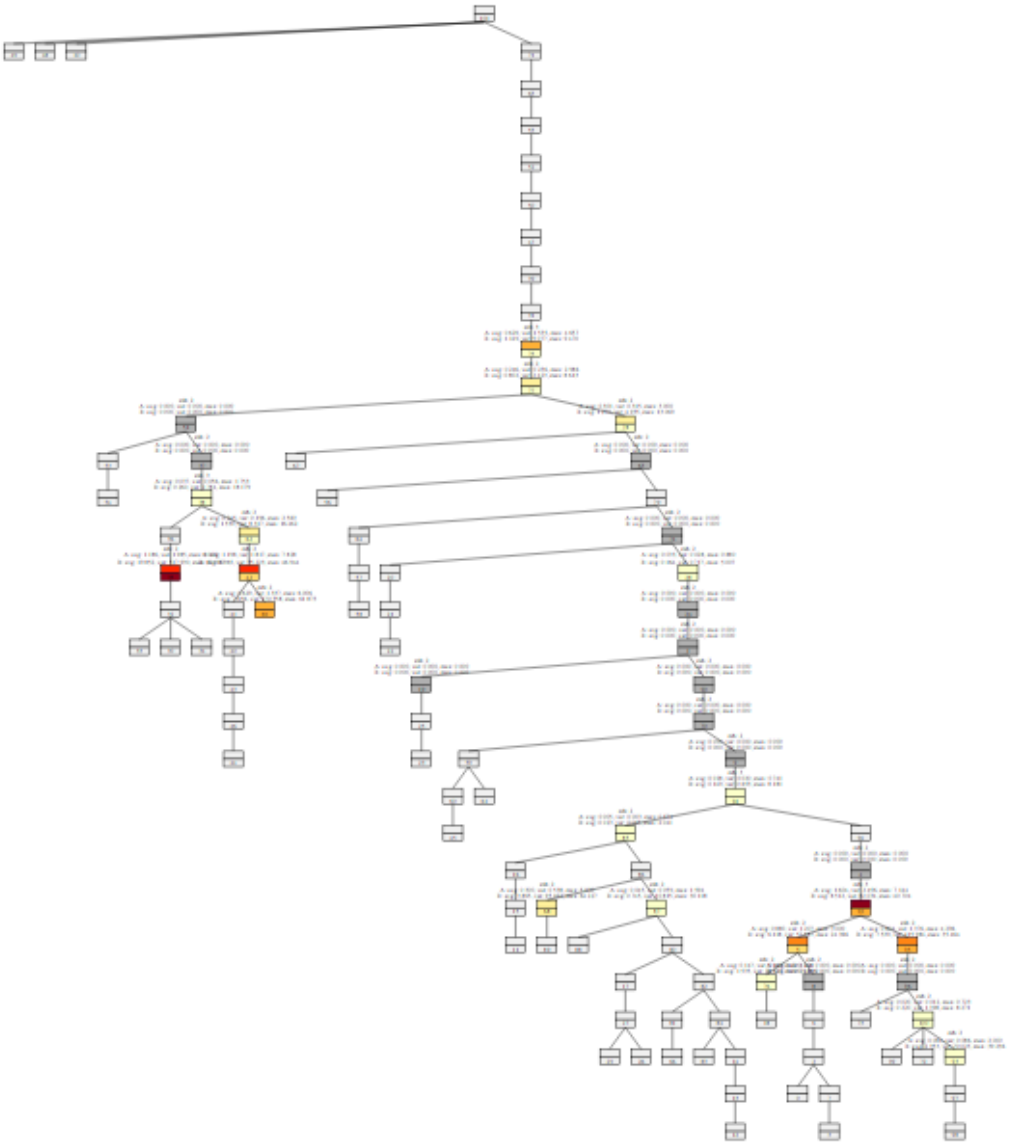


Figure 19: Extracted structure from pseudotree showing errors for pdb3nul with an i-bound of 5.

5.4 Discussion

In general, we observe that errors accumulate on chains at the separators of the underlying tree decomposition corresponding to the bucket-tree. We plan to develop an improved look-ahead scheme that works faster by considering the pattern that errors show up as clusters. For example, we can reuse the computation for of a depth n look-ahead subtree for a depth $n-1$ look-ahead subtree at its child, since pruned depth n subtree may turn out to be identical.

Next, the most significant errors do not show up until a certain depth. First, we must reach a depth greater than the i-bound before there is any partitioning. Next, the errors may not be significant until there are enough variables there after to produced a significant amount of partitioning (and therefore error) in a bucket. The main implication of this observation is that when using any search algorithm to produce an anytime heuristic bound on the entire problem, this bound may not tighten until we reach a node with significant error. Therefore, we do not expect to see the heuristic bound tighten quickly when the i-bound is high. Still, it will be interesting to use Best-First search on lower i-bounds.

6 Next Steps

Based on the comparison of the metrics, we should run actual AOBB experiments using each of the absolute and relative errors as BEE metrics. We would have liked to use variance as a way to encourage different value ordering in the search, but the data here shows that it is highly correlated with the average already.

Based on the locations of error in the pseudotree structures, a more efficient look-ahead scheme as described can improved performance significantly. In addition, we plan to implement Best-First search as an upper-lower bounding scheme and run experiments.

References

[1] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[2] Rina Dechter, Kalev Kask, and William Lam. Some properties of look-ahead with mini-bucket heuristics. Technical report, Unpublished, UC Irvine, ICS, 2015.

A Error Plots by i-bound

We show here the error plots by i-bound for the average absolute error and average relative error. The full set of plots over all instances can be found at `http://graphmod.ics.uci.edu/~wlam/bee-plots/by-ibound`.

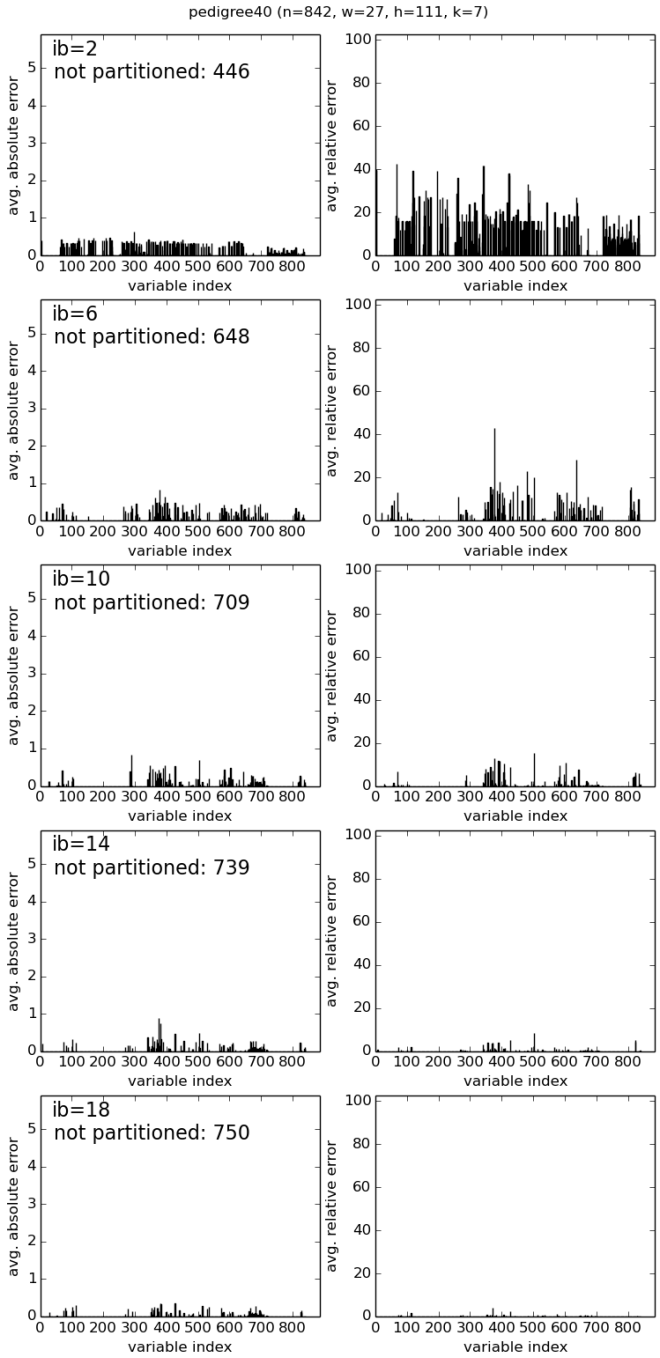


Figure 20: Extracted structure from pseudotree showing errors for pedigree40 with an i-bound of 5.

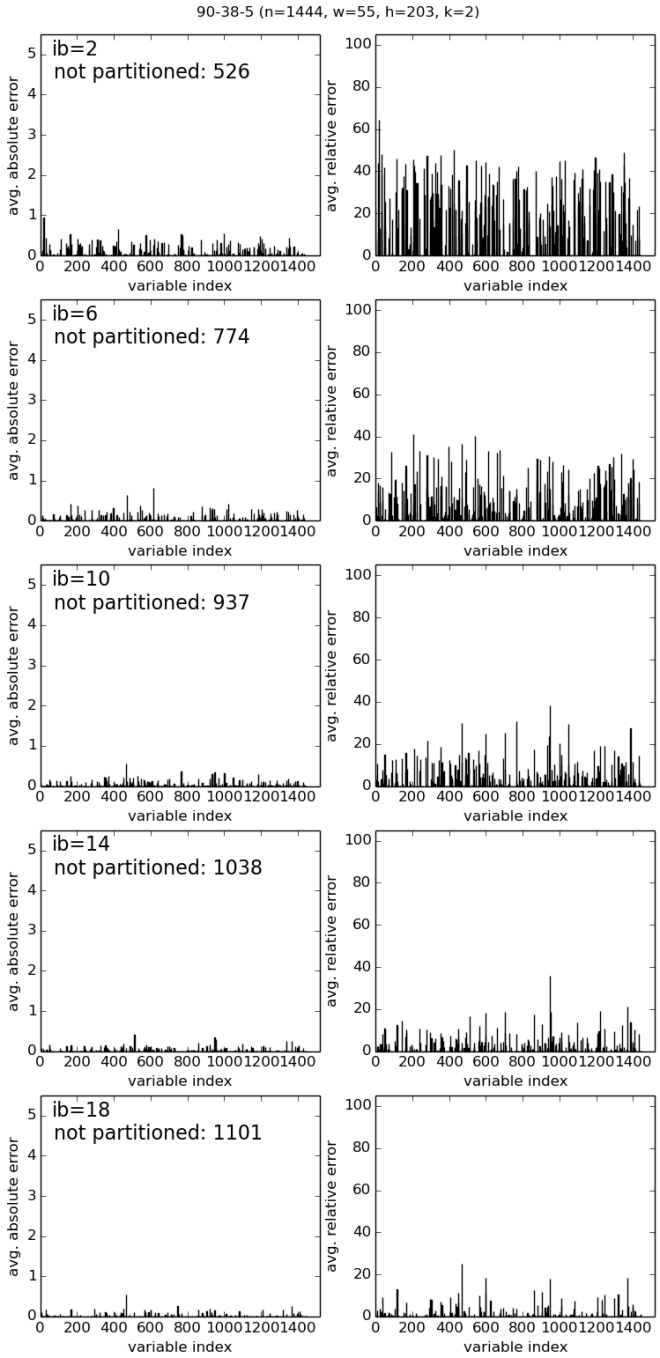


Figure 21: Extracted structure from pseudotree showing errors for 90-38-5 with an i-bound of 5.

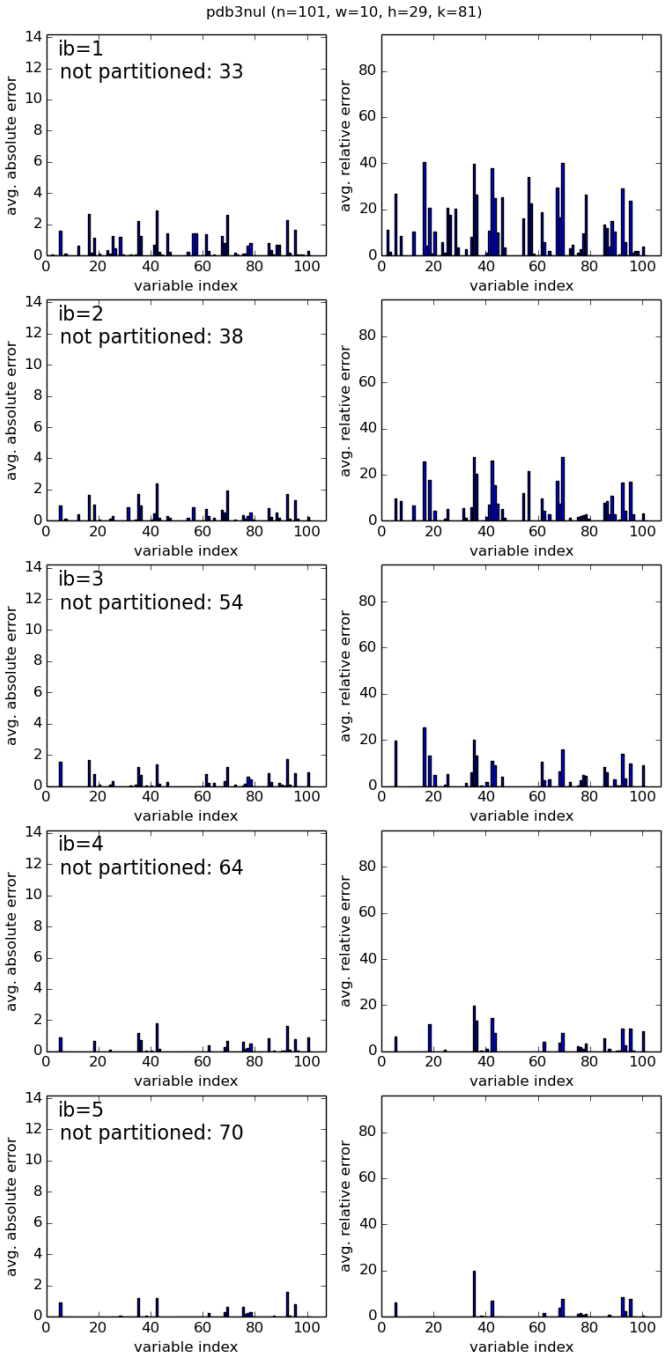


Figure 22: Extracted structure from pseudotree showing errors for pdb3nul with an i-bound of 5.