Variable and Value Ordering for MPE Search

Sajjad Siddiqi and **Jinbo Huang**
National ICT Australia and Australian National University
Canberra, ACT 0200 Australia
{sajjad.siddiqi, jinbo.huang}@nicta.com.au

Abstract

In Bayesian networks, a *most probable explanation* (MPE) is a most likely instantiation of all network variables given a piece of evidence. Recent work proposed a branch-and-bound search algorithm that finds exact solutions to MPE queries, where bounds are computed on a relaxed network obtained by a technique known as *node splitting*. In this work we study the impact of variable and value ordering on such a search algorithm. We study several heuristics based on the entropies of variables and on the notion of *nogoods*, and propose a new meta-heuristic that combines their strengths. Experiments indicate that search efficiency is significantly improved, allowing many hard problems to be solved for the first time.

1 Introduction

In Bayesian networks, a *most probable explanation* (MPE) is a most likely instantiation of all network variables given a piece of evidence. Exact algorithms for MPE based on *inference* include variable elimination, jointree, and, more recently, compilation [Chavira and Darwiche, 2005]. While variable elimination and jointree algorithms have a complexity exponential in the treewidth of the network and are hence impractical for networks of large treewidth, compilation is known to exploit *local structure* so that treewidth is no longer necessarily a limiting factor [Chavira and Darwiche, 2005].

When networks continue to grow in size and complexity, however, all these methods can fail, particularly by running out of memory, and one resorts instead to *search* algorithms. Most recently, Choi *et al.* [2007] proposed a branch-and-bound search framework for finding exact MPE solutions where bounds are computed by solving MPE queries on a relaxed network. The latter is obtained by *splitting* nodes of the network in such a way that (i) its treewidth decreases, making the MPE easier to solve, and (ii) the MPE probability of the relaxed network is no less than that of the original. This framework allows one to capitalize on advances in exact inference methods (such as the ones we mentioned above) and focus instead on generating good network relaxations.

Moreover, Choi *et al.* have shown that the well-known *mini-bucket* heuristic [Dechter and Rish, 2003] is none other than a specific strategy for obtaining such relaxed networks, and hence it can be used in conjunction with any exact inference algorithm—not just variable elimination—to solve MPE. In particular, they have shown that by replacing variable elimination with compilation, it is possible to improve search efficiency by orders of magnitude on hard networks.

In this paper, we enrich this framework by studying the impact of variable and value ordering on the search efficiency. Specifically, we study heuristics based on the entropies of variables and on the notion of *nogoods*. The idea is to start the search with a high probability solution computed from the entropies of variables, and then use dynamically computed scores for variables to favor *nogood* assignments, which tend to cause early backtracking. Compared with the "neutral" heuristic used in [Choi *et al.*, 2007], we show that our new heuristics further improve efficiency significantly, extending the reach of exact algorithms to networks that cannot be solved by other known methods.

The rest of the paper is organized as follows. We review the framework of [Choi *et al.*, 2007] in Section 2, and present our new heuristics in Section 3. Section 4 reports an extensive empirical study, and Section 5 concludes the paper.

2 Search for MPE

In this section we briefly review the search framework based on node splitting proposed in [Choi *et al.*, 2007], which provides the basis for our new contributions. We start with a formal definition of MPE. Let N be a Bayesian network with variables $\mathbf{X}$, a *most probable explanation* for some evidence $\mathbf{e}$ is defined as:

$$MPE(N, \mathbf{e}) =^{def} arg\ max_{\mathbf{x} \sim \mathbf{e}} Pr_N(\mathbf{x})$$

where $\mathbf{x} \sim \mathbf{e}$ means that the assignments $\mathbf{x}$ and $\mathbf{e}$ are consistent, i.e, they agree on every common variable. We will also write $MPE_p(N, \mathbf{e})$ to denote the probability of the MPE solutions.

MPE queries involve potentially maximizing over all (uninstantiated) network variables, and are known to be NP-hard in general. The central idea behind node splitting is to simplify the network in a systematic way so that the new network is easier to solve and yet its solution can only "go wrong" in one direction. These approximate solutions can then be used as bounds to prune a branch-and-bound search for an exact MPE.

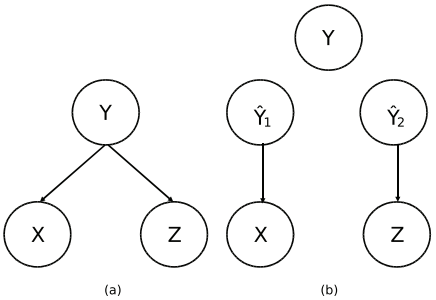

Figure 1: An example of node splitting.

2.1 Node Splitting

Node splitting creates a *clone* $\hat{X}$ of a node X such that $\hat{X}$ inherits some of the children of X. Formally:

Definition 1 (Node Splitting). *Let X be a node in a Bayesian network N with children $\mathbf{Y}$. We say that X is* **split according to children $\mathbf{Z} \subseteq \mathbf{Y}$** *when it results in a network N' that is obtained from N as follows:*

- *The edges outgoing from X to its children $\mathbf{Z}$ are removed.*
- *A new root node $\hat{X}$ with a uniform prior is added to the network with nodes $\mathbf{Z}$ as its children.*

A special case of splitting is when X is split according to every child, in which case X is said to be *fully split*. Figure 1 shows an example where Y has been split according to both of its children X and Z, and $\hat{Y}_1, \hat{Y}_2$ are clones of Y.

If $\mathbf{e}$ is an assignment to variables in network N, we write $\mathbf{e}^{\rightarrow}$ to denote the compatible assignment to their clones in N', meaning that a variable and its clones (if any) are assigned the same value. For example, if $\mathbf{e} = \{Y = y\}$, then $\mathbf{e}^{\rightarrow} = \{\hat{Y}_1 = y, \hat{Y}_2 = y\}$.

An interesting property of a split network is that the probability of the MPE with respect to the split network gives us an upper bound on the probability of the MPE with respect to the original network, after normalization. Formally:

$$MPE_p(N, \mathbf{e}) \leq \beta MPE_p(N', \mathbf{e}, \mathbf{e}^{\rightarrow})$$

where β is a constant equal to the total number of instantiations of the clone variables. For example, suppose that in the network of Figure 1a, variables have binary domains and $Pr(Y = y) = 0.6$, $Pr(Y = \bar{y}) = 0.4$; all the parameters in the CPTs of X and Z are 0.5; and $\mathbf{e} = \{Y = \bar{y}\}$. Recall that both Y_1, Y_2 are given uniform priors. Then $MPE_p(N, \mathbf{e}) = 0.10$ and $MPE_p(N', \mathbf{e}, \mathbf{e}^{\rightarrow}) = 0.025$. The value of β is 4 in this case and we have $MPE_p(N, \mathbf{e}) \leq 4 * 0.025$.

In this example the upper bound equals the exact solution. In general, this is guaranteed if all the split variables have been instantiated in $\mathbf{e}$ (and their clones in $\mathbf{e}^{\rightarrow}$). Hence to find an exact MPE one need only search in the space of instantiations of the split variables, as opposed to all network variables. At leaves of the search tree, solutions computed on the relaxed network give candidate MPE solutions, and elsewhere they give upper bounds to prune the search.

Algorithm 1 BNB-MPE : Computes probability of MPE for evidence $\mathbf{e}$

procedure BNB-MPE ($N', \mathbf{e}$)
inputs: $\{N$: split network$\}$, $\{\mathbf{e}$: network instantiation$\}$
global variables: $\{Y$: split variables$\}$, $\{\beta$: number of possible assignments to clone variables$\}$

```
1:  bound = βMPE(N', e, e→)
2:  if bound > solution then
3:      if e is complete instantiation of variables Y then
4:          solution = bound      /* bound is exact */
5:      else
6:          pick some X ∈ Y such that X ∉ E
7:          for each value x_i of variable X do
8:              e ← e ∪ {X = x_i}
9:              BNB-MPE (N', e)
10:             e ← e\{X = x_i}
```

2.2 Branch-and-Bound Search

Algorithm 1 formalizes such a branch-and-bound search (this code only computes the MPE probability; however the actual MPE can be recovered with minor book-keeping). The procedure receives a split network N' and the evidence $\mathbf{e}$. At each call to BNB-MPE, the bound $\beta MPE(N', \mathbf{e}, \mathbf{e}^{\rightarrow})$ is computed (line 1). If the bound becomes less than or equal to the current solution (line 2), meaning that any further advancement in this part of the search tree is not going to give a better solution, the search backtracks. Otherwise, if the bound is greater than the current solution and the current assignment is complete over split variables, meaning that the bound has become exact, the current solution is replaced with the better one (lines 3–4). If the assignment is not complete then an unassigned split variable X is chosen (line 6), $\mathbf{e}$ is appended with the assignment $X = x_i$ (line 7), and BNB-MPE is recursively called (line 8). After the recursive call returns the assignment $X = x_i$ is removed from $\mathbf{e}$ and other values of X are tried in the same way.

The choice of which variables to split (variables $\mathbf{Y}$ in the pseudocode) has a fundamental impact on the efficiency of this approach. More split variables may make the relaxed network easier to solve, but may loosen the bounds and increase the search space. Choi *et al.* [2007] studied a strategy based on jointree construction: A variable is chosen and fully split that can cause the highest reduction in the size of the jointree cliques and separators. Once a variable is fully split a jointree of the new network is constructed, and the process repeated until the treewidth of the network drops to a preset target. This heuristic was shown to outperform the mini-bucket heuristic in [Choi *et al.*, 2007], and is therefore used in our present work. Also, given the success reported in [Choi *et al.*, 2007] we will use compilation as the underlying method for computing MPEs of relaxed networks.

3 Variable and Value Ordering

As mentioned earlier, by using a neutral heuristic Choi *et al.* [2007] has left unaddressed an important aspect of perhaps any search algorithm—variable and value ordering. Given the advancements already reported in their work based on combining splitting strategies with new inference methods, one

cannot but wonder whether more sophisticated variable and value ordering might not take us even farther.

We have thus undertaken this investigation and now take delight in reporting it here. The first type of heuristic we examined is based on computing the entropies of variables, and the second on a form of learning from nogoods. We will discuss some of our findings, which have eventually led us to combine the two heuristics for use in a single algorithm.

3.1 Entropy-based Ordering

We start by reviewing the notion of Shannon's entropy ξ [Pearl, 1979], which is defined with respect to a probability distribution of a discrete random variable X ranging over values $x_1, x_2, \ldots, x_k$. Formally:

$$\xi(X) = -\sum_{i=1}^{k} Pr(X = x_i) \log Pr(X = x_i).$$

Entropy quantifies the amount of uncertainty over the value of the random variable. It is maximal when all probabilities $Pr(X = x_i)$ are equal, and minimal when one of them is 1.

Hence as a first experiment we consider a heuristic that favors those instantiations of variables that are more likely to be MPEs. The idea is that finding an MPE earlier helps terminate the search earlier. To that end, the heuristic prefers those variables that provide more certainty about their values (i.e., have smaller entropies), and favor those values of the chosen variable that are more likely than others (i.e., have higher probabilities).

This heuristic can be used in either a static or dynamic setting. In a static setting, we order the split variables in increasing order of their entropies, and order the values of each variable in decreasing order of their probabilities, and keep the order fixed throughout the search. In a dynamic setting, we update the probabilities of the split variables at each search node and reorder the variables and values accordingly.

The heuristic requires computing the probabilities of the values of all split variables, which can be obtained conveniently as we have already assumed that the split network will be compiled for the purpose of computing its MPE—the compilation, in the form of *arithmetic circuits* (ACs), admits linear-time procedures for obtaining these probabilities. Furthermore, in an attempt to improve the accuracy of the probabilities (with respect to the original network), we take the average of the probabilities of a split variable and its clones (for the same value) when evaluating the heuristic.

While we will present detailed results in Section 4, we note here that our initial experiments clearly showed that a static entropy-based ordering immediately led to significantly better performance than the neutral heuristic. On the other hand, the dynamic version, although often effective in reducing the search tree, turned out to be generally too expensive, resulting in worse performance overall.

3.2 Nogood-based Ordering

Hence we need to look further if we desire an effective heuristic that remains inexpensive in a dynamic setting. Here the notion of learning from *nogoods* comes to rescue.

Nogoods

In the *constraint satisfaction* framework where it was originally studied, a nogood is a partial instantiation of variables that cannot be extended to a complete solution. In our case, we define a nogood to be a partial instantiation of the split variables (i.e., search variables) that results in pruning of the node (i.e., the upper bound being $\leq$ the current best candidate solution).

Note that at the time of pruning, some of the assignments in the nogood $\mathbf{g}$ may contribute more than others to the tightness of the bound (and hence the pruning of the node), and it may be possible to retract some of those less contributing assignments from $\mathbf{g}$ without loosening the bound enough to prevent pruning. These refined nogoods can then be *learned* (i.e., stored) so that any future search branches containing them can be immediately pruned.

Nogood learning in this original form comes with the significant overhead of having to re-compute bounds to determine which assignments can be retracted (see the discussion of Table 1 in Section 4.1). However, it gives us an interesting motivation for an efficient variable and value ordering heuristic described below.

Ordering Heuristic

The idea is to favor those variables and values that can quickly make the current assignment a nogood, so that backtracking occurs early during the search. Hence we give scores to the values of variables proportionate to the amounts of reduction that their assignments will cause in the bound, and favor those variables and values that have higher scores.

Specifically, every value x_i of a variable X is associated with a score $S(X = x_i)$, which is initialized to 0. The score $S(X)$ of the variable X is the average of the scores of its values. Once a variable X is assigned a value x_i, the amount of reduction in the bound caused by it is added to the score of $X = x_i$. That is, the score $S(X = x_i)$ is updated as $S(X = x_i) = S(X = x_i) + (cb - nb)$, where cb is the bound before assigning the value x_i to X and nb is the bound after the assignment. The heuristic chooses a variable X with the highest score and assigns values to it in decreasing order of the scores of those values.

During initial experiments we observed that over the course of the search the scores can become misleading, as past updates to the scores may have lost their relevance under the current search conditions. To counter this effect, we reinitialize the scores periodically, and have found empirically that a period of 2000 search nodes tends to work well.

Finally, we combine this heuristic with the entropy-based approach by using the entropy-based static order as the initial order of the variables and their values, so that the search may tend to start with a better candidate solution. We now proceed to present an empirical study which shows, among other things, that this combined heuristic gives significantly better performance both in terms of search space and time.

4 Experimental Results

In this empirical study we evaluate the different heuristics we have proposed, and in particular show that with the final combined heuristic we achieve significant advances in efficiency

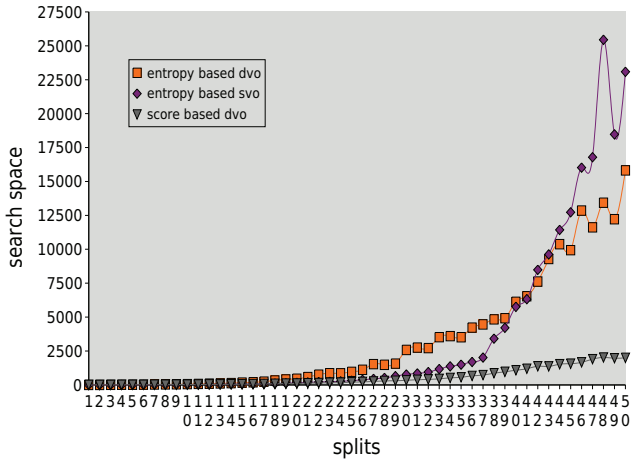

Figure 2: Comparing search spaces on grid networks.

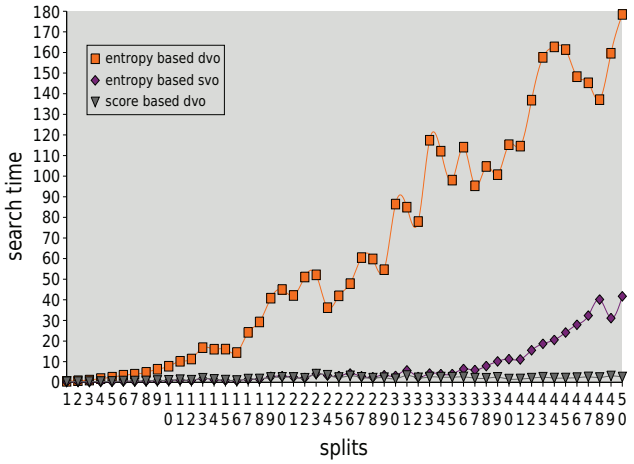

Figure 3: Comparing search times on grid networks.

and scalability, able to solve problems that are intractable for other known methods.

We conducted our experiments on a cluster of computers consisting of two types of (comparable) CPUs, Intel Core Duo 2.4 GHz and AMD Athlon 64 X2 Dual Core Processor 4600+, both with 4 GB of RAM running Linux. A memory limit of 1 GB was imposed on each test case. Compilation of Bayesian networks was done using the C2D compiler [Darwiche, 2004; 2005]. We use a trivial seed of 0 as the initial MPE solution to start the search. In general, we keep splitting the network variables until treewidth becomes ≤ 10, unless otherwise stated.

We used a variety of networks: the grid networks introduced in [Sang *et al.*, 2005], a set of randomly generated networks as in [Marinescu *et al.*, 2003], networks for genetic linkage analysis, and a set of 42 networks for decoding error-correcting codes as generated in [Choi *et al.*, 2007]. The last group were trivial to solve by all the three heuristics, so we do not report results for them.

For each heuristic, we report the number of cases solved, search time, and search space, where the search space are time are averages over solved cases. We also compare the performance of heuristics on those cases that were solved by all heuristics. As a reference point, we generated a random static variable and value order for the split variables in each split network instance and compared all the heuristics against it. The comparison showed that the new heuristics generally provide many orders of magnitude savings in search time and space over the random heuristic, and hence we will only include the new heuristics in the presented results. Finally, we will write SVO and DVO as shorthand for static and dynamic variable and value ordering, respectively.

4.1 Grid Networks

We first use these networks to show, in Table 1, that the score-based dynamic ordering (referred to as SC-DVO) outperforms nogood learning itself. For this purpose, we consider a similar dynamic ordering referred to as NG-DVO in the table. This heuristic starts with the same initial order, uses the same method of variable and value selection, and also reinitializes the scores after the specified period. However, the scores are updated as follows: When a nogood is learned, the score is incremented for each literal in the nogood. We compare the performance of both techniques by computing the MPE for the empty evidence on each network. For NG-DVO we also report the average number of nogoods learned and the average number of branches pruned by nogoods. The results clearly show that score-based DVO is significantly faster than nogood learning and results in a much reduced search space.

We then show that the score-based DVO performs consistently better than other heuristics when the number of splits in a network is varied. For this purpose, we consider only 80 instances of these networks, in the range 90-20-1 to 90-30-9, which are some of the harder instances in this suite. The treewidths of these instances range from high twenties up to low forties. However, all of them can be compiled with a single split using the strategy mentioned above.

For each instance we split the network using 1 up to 50 splits, making a total of $50 * 80 = 4000$ cases to solve. For each case we compute the MPE for the empty evidence under a time limit of 15 minutes. The entropy-based DVO solved 3656 cases, entropy-based SVO solved 3968 cases, and score-based DVO solved 3980 cases, where the score-based DVO solved all those cases solved by the others. We plot a graph of search space against the number of splits, and a graph of search time against the number of splits. Each of data point on the graphs is the average of search spaces/time for those queries solved by all three heuristics.

In Figure 2, we observe that, interestingly, entropy-based SVO can often perform significantly better than entropy-based DVO, although its search space starts to grow faster than the DVO when the number of splits increases. This can be ascribed to the fact that DVO can exploit the changing probabilities of variables during search. The most significant result, however, is that for any number of splits the search space of score-based DVO is generally much smaller than that of the other two, and becomes orders of magnitude smaller when the number of splits grows.

In Figure 3, we observe that entropy-based DVO gives poor performance, apparently because it has to do expensive prob-

networks	cases	cases solved								common cases			
		NG-DVO					SC-DVO			NG-DVO		SC-DVO	
		solved	time	space	nogoods	pruned	solved	time	space	time	space	time	space
Ratio 50	90	77	89.5	2337	875	576	90	33.2	3365	89.5	2337	7.7	1218
Ratio 75	150	108	74.8	3202	1019	1159	139	49.7	14875	74.8	3202	3.4	1648
Ratio 90	210	135	42.3	6161	873	4413	160	26.2	13719	42.3	6161	1.6	1402

Table 1: Comparing nogood learning with score-based DVO on grid networks.

networks	cases	cases solved									common cases					
		ENT-DVO			ENT-SVO			SC-DVO			ENT-DVO		ENT-SVO		SC-DVO	
		solved	time	space	solved	time	space	solved	time	space	time	space	time	space	time	space
Ratio 50	2250	1890	95.7	6286	2121	51.6	8262	2248	25.9	2616	94.1	6215	24.3	4619	7.3	1082
Ratio 75	3750	2693	60.3	5195	3015	40.4	15547	3624	38.0	10909	57.2	4965	15.3	6722	1.6	712
Ratio 90	5250	3463	23.4	2387	3599	22.2	12919	3995	8.2	4453	20.0	2102	10.0	6740	0.3	275

Table 2: Results on grid networks.

ability updates at each search node. Again, we note that the score-based DVO is generally much faster than the other two heurstics, and becomes orders of magnitude faster when the number of splits grows.

To further assess the performance of heuristics on a wide range of MPE queries, we considered the whole suite of grid networks and randomly generated 25 queries for each network. The results, summarized in Table 2, again confirm that the performance of entropy-based DVO is significantly better than that of the other two heuristics.

4.2 Randomly Generated Networks

Next, to evaluate the scalability of the heuristics we tried them on a number of randomly generated networks of increasing size and treewidth, according to the model of [Marinescu *et al.*, 2003]. Networks are generated according to the parameters (N, K, C, P), where N is the number of variables, K is their domain size, C is the number of CPTs, and P is the number of parents in each CPT. The network structure is created by randomly picking C variables out of N and, for each, randomly selecting P parents from the preceding variables, relative to some ordering. Each CPT is generated uniformly at random. We used $C = 90\%N$, $P = 2$, $K = 3$, and N ranging from 100 to 200 at increments of 10. The treewidths of these networks range from 15 to 33, and generally increases with N. For each network size we generated 20 network instances, and for each instance generated a random MPE query, making a total of 20 queries per network size. The time limit to solve each query was set to 1 hour.

The results of these experiments are summarized in Table 3. The score-based DVO solved all those cases that the other two could solve, plus more, and scales much better when the network size increases. On cases solved by all three, the performance of score-based DVO is also the best in terms of both search time and space.

4.3 Networks for Genetic Linkage Analysis

We now consider even harder cases from genetic linkage analysis (http://bioinfo.cs.technion.a.c.il/superlink/). We extracted 8 networks that have high treewidths and cannot be compiled without splitting, and considered 20 randomly generated MPE queries on each of them, for a total of 160 queries. Since these networks are significantly harder, the time limit on each query was set to 4 hours.

network	treewidth	splits	SC-DVO		
			solved	time	space
pedigree13	43	83	20	1728	168251
pedigree19	36	59	12	6868	532717
pedigree31	44	84	16	6672	446583
pedigree34	35	79	12	5109	332899
pedigree40	37	72	2	6575	745088
pedigree41	43	84	9	7517	783737
pedigree44	33	54	5	7041	673165
pedigree51	54	75	16	3451	268774

Table 4: Results on networks for genetic linkage analysis.

The results on these networks are summarized in Table 4. We report the estimated treewidths of the networks and the number of splits performed on each of them, and only show results for score-based DVO, as the two entropy-based heuristics could only solve 9 trivial cases when the MPE for the given evidence was already 0 and the search was not performed at all. The score-based DVO, by contrast, solved most of the cases, which further establishes its better scalability.

networks	cases	cases solved				time on common cases	
		SamIam		SC-DVO		SamIam	SC-DVO
		solved	time	solved	time		
Ratio 50	90	49	23.8	90	31.1	23.8	1.4
Ratio 75	150	45	14.1	148	185.4	14.1	0.3
Ratio 90	210	48	18.9	169	114.5	18.9	0.1

Table 5: Comparison with SamIam on grid networks.

4.4 Comparison with Other Tools

First we compare our MPE solver SC-DVO with an independent Bayesian network inference engine known as SamIam (http://reasoning.cs.ucla.edu/samiam/). As we were unable to run SamIam on our cluster, these experiments were conducted on a machine with a 3.2 GHz Intel Pentium IV processor and 2 GB of RAM running Linux. For each network we compute the MPE for the empty evidence under a time limit of 1 hour.

The random and genetic linkage analysis networks proved too hard to be solved by SamIam. For the grid networks, the results of the comparison are summarized in Table 5. We observe that SC-DVO solved roughly from 2 to 3 times more

size	cases solved									common cases					
	ENT-DVO			ENT-SVO			SC-DVO			ENT-DVO		ENT-SVO		SC-DVO	
	solved	time	space	solved	time	space	solved	time	space	time	space	time	space	time	space
100	20	131	867	20	47	696	20	31	532	131	867	47	696	31	532
110	20	187	1800	20	70	1270	20	67	952	187	1800	70	1270	67	952
120	19	923	9374	20	471	7760	20	210	3061	923	9374	434	7640	188	2950
130	12	1100	10849	15	1100	19755	18	561	7462	1100	10849	729	11168	291	3989
140	7	1715	19621	11	1340	28574	15	1090	10545	1680	20016	741	17965	311	4529
150	1	2505	10800	6	1859	41628	15	1619	25293	2505	10800	809	8727	634	6789
160	0	n.a.	n.a.	1	2379	22428	6	2257	36884	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
170	2	0.1	0	2	0.1	0	3	637	14319	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.

Table 3: Results on random networks.

networks	cases	cases solved				time on common cases	
		SMBBF		SC-DVO		SMBBF	SC-DVO
		solved	time	solved	time		
Ratio 50	9	7	5.39	9	14.33	5.39	4.34
Ratio 75	15	10	16.16	15	199.11	16.16	2.1
Ratio 90	15	10	7.32	15	3.87	7.32	0.429

Table 6: Comparison with SMBBF on grid networks.

instances than SamIam, and on cases solved by both our solve is also orders of magnitude faster.

We then compare SC-DVO with the best-first search algorithm (SMBBF) of [Marinescu and Dechter, 2007]. This tool is only available for Windows, and we used a machine with an Intel Core 2 Duo 2.33 GHz and 1 GB of memory running Windows. Note that our solver SC-DVO was run on a cluster with comparable CPU speeds. However, since Windows reserves some of the memory for the operating system, we accordingly reduced the memory limit for SC-DVO from 1 GB to 768 MB for these experiments. Again a time limit of 1 hour was imposed for each query, computing the MPE for the empty evidence .

SMBBF requires a parameter i that bounds the mini-bucket size. The results of [Marinescu and Dechter, 2007] do not appear to suggest any preferred value for i; hence we somewhat arbitrarily set $i = 20$ taking into account the relatively high treewidth of our benchmarks.

Due to limited computational resources we used a sample of the grid networks under each ratio category such that each selected network represented a class of networks comparable in difficulty. The comparison of is shown in Table 6. In contrast to SC-DVO, SMBBF failed to solve a significant number of the networks; it was also slower on the networks that were solved by both.

Finally, from the random networks we arbitrarily took 4 networks of size 100, 110, 120, and 130, respectively, and from the networks for genetic linkage analysis we took pedigree13, the apparently easiest one. None of these instances could be solved by SMBBF, which quicky ran out of memory and got terminated by the operating system. By contrast, SC-DVO solved all of them.

5 Conclusion

We have presented a novel and efficient heuristic for dynamically ordering variables and their values in a branch-and-bound search for MPE. A comprehensive empirical evaluation indicates that significant improvements in search time and space are achieved over less sophisticated heuristics and over existing MPE solvers. On the whole, we have extended the reach of exact MPE algorithms to many hard networks that are now solved for the first time.

Acknowledgments

Thanks to the anonymous reviewers for their comments. National ICT Australia is funded by the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

References

[Chavira and Darwiche, 2005] Mark Chavira and Adnan Darwiche. Compiling bayesian networks with local structure. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1306–1312, 2005.

[Choi *et al.*, 2007] Arthur Choi, Mark Chavira, and Adnan Darwiche. Node splitting: A scheme for generating upper bounds in bayesian networks. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 57–66, 2007.

[Darwiche, 2004] Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 328–332, 2004.

[Darwiche, 2005] Adnan Darwiche. The C2D compiler user manual. Technical Report D-147, Computer Science Department, UCLA, 2005. http://reasoning.cs.ucla.edu/c2d/.

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

[Marinescu and Dechter, 2007] Radu Marinescu and Rina Dechter. Best-first AND/OR search for most probable explanations. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[Marinescu *et al.*, 2003] Radu Marinescu, Kalev Kask, and Rina Dechter. Systematic vs. non-systematic algorithms for solving the MPE task. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2003.

[Pearl, 1979] Judea Pearl. Entropy, information and rational decisions. *Policy Analysis and Information Systems, Special Issue on Mathematical Foundations*, 3(1):93–109, 1979.

[Sang *et al.*, 2005] Tian Sang, Paul Beame, and Henry Kautz. Solving bayesian networks by weighted model counting. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 475–482. AAAI Press, 2005.