MAP Search in Bayesian Networks Using Joint Bounds

Changhe Yuan and **Eric A. Hansen**
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
{cyuan,hansen}@cse.msstate.edu

Abstract

Maximum a Posteriori assignment (MAP) is the problem of finding the most probable instantiation of a set of variables in a Bayesian network given partial evidence for the remaining variables. The state-of-the-art exact solution method is branch-and-bound search using a jointree upper bound proposed by Park and Darwiche (2003). In this approach, almost all of the search time is due to computation of the bounds. We start with the observation that the bound computation typically produces what we call *joint bounds*, i.e., bounds for joint configurations of groups of variables. Based on this observation, we show how to improve search efficiency by instantiating multiple variables at each step of the search, instead of a single variable, which reduces the number of times an upper bound needs to be computed. We demonstrate the effectiveness of this approach in solving a range of benchmark Bayesian networks.

Introduction

Bayesian networks (Pearl 1988) provide a compact graphical representation of uncertain relations among random variables in a domain that supports efficient probabilistic reasoning. An important inference problem in Bayesian networks, MAP, is to find the most probable configuration of a set of variables (the explanatory variables) given partial evidence for the remaining set of variables (the observed or evidence variables). MAP can be used to diagnose a system and determine the most likely state, in order to decide whether the system is in an anomaly state, and, if so, whether it needs repair, replacement, or further testing. There are many practical applications of MAP inference and it has received much attention in Bayesian network research.

The state-of-the-art exact solution method for MAP is depth-first branch-and-bound (DFBnB) search using an upper bound proposed by Park and Darwiche (2003). The upper bound is computed using a modified jointree algorithm in which the messages of the original jointree algorithm are redefined so that the probabilities obtained in the end are not marginal probabilities, but upper bounds on the probabilities of consistent joint configurations. Although this provides effective bounds, its computation can be time and memory-intensive, and our experiments show that more than 95% of search time is devoted to computing these bounds.

In this paper, we start with the observation that this upper bound computation typically produces what we call *joint bounds*, that is, bounds for joint configurations of groups of variables. Based on this observation, we show how to improve search efficiency by instantiating multiple variables at each step of the DFBnB search, instead of a single variable at a time. This creates an equivalent, but smaller, search tree in which the reduction in the size of the tree corresponds to a reduction in the number of times that jointree upper bounds need to be computed, resulting in faster search. We demonstrate the effectiveness of this approach in solving a range of benchmark Bayesian networks. Our results show that leveraging joint bounds in this way improves the efficiency with which DFBnB can find optimal solutions, and also enables DFBnB to find good (but possibly suboptimal) solutions faster.

Upper Bounds for MAP Search

The Maximum a Posteriori assignment (MAP) problem is defined as follows. Let $\mathbf{M}$ be a set of explanatory variables in a Bayesian network; from now on, we call these the *MAP variables*. Let $\mathbf{E}$ be a set of evidence variables whose states have been observed. The remaining variables, denoted $\mathbf{S}$, are variables for which the states are unknown and not of interest. Given an assignment $\mathbf{e}$ for the variables $\mathbf{E}$, the MAP problem is to find an assignment $\mathbf{m}$ for the variables $\mathbf{M}$ that maximizes the probability $P(\mathbf{m}, \mathbf{e})$ (or equivalently, $P(\mathbf{m}|\mathbf{e})$). Formally,

$$\hat{\mathbf{m}}_{MAP} = argmax_{\mathbf{M}} \sum_{\mathbf{S}} P(\mathbf{M}, \mathbf{S}, \mathbf{E} = \mathbf{e}) \,, \qquad (1)$$

where $P(\mathbf{M}, \mathbf{S}, \mathbf{E} = \mathbf{e})$ is the joint probability distribution of the network given the assignment $\mathbf{e}$. In the special case in which $\mathbf{S}$ is empty, this is referred to as the Most Probable Explanation (MPE) problem. Of the two problems, the MAP problem is more difficult. The

decision problem for MAP is NP^{PP}-complete (Park 2002); in contrast, the decision problem for MPE is only NP-complete (Shimony 1994). MAP is difficult not only because the size of its search space is equal to the product of the cardinalities of all MAP variables, but because computing the probability of any instantiation of the MAP variables is PP-complete (Roth 1996).

We can rewrite $P(\mathbf{M}, \mathbf{S}, \mathbf{E} = \mathbf{e})$ as the product of the conditional probability distributions in a Bayesian network, using the so-called chain rule (Pearl 1988). In Equation (1), note that the maximization and summation operators are applied to different sets of variables. The MAP variables in $\mathbf{M}$ can be maximized in different orders, and the variables in $\mathbf{S}$ can be summed out in different orders. The summations and maximizations are not commutable, however. All summations have to be done before any maximization can be carried out; violating this restriction produces incorrect solutions. In fact, we have the following theorem (Park 2002).

Theorem 1 *Let $\phi(\mathbf{M}, \mathbf{S}, \mathbf{Z})$ be a potential over disjoint variable sets $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{Z}$. For any instantiation $\mathbf{z}$ of $\mathbf{Z}$, the following inequality holds:*

$$\sum_S max_M \phi(\mathbf{M}, \mathbf{S}, \mathbf{Z} = \mathbf{z}) \geq max_M \sum_S \phi(\mathbf{M}, \mathbf{S}, \mathbf{Z} = \mathbf{z}) \ .$$

Theorem 1 tells us that if we relax the orderings among the summations and maximizations, we obtain upper-bound probabilities. Park and Darwiche's (2003) technique for computing upper bounds is thus an example of the general approach to computing bounds by finding exact solutions of a relaxed version of a problem. Their specific technique uses the jointree or clustering algorithm (Lauritzen & Spiegelhalter 1988) to build a secondary structure called a clique tree and performs inference on the tree by message passing. The clique tree provides a systematic way of commuting the summations and maximizations in a MAP problem. Their message-passing scheme is mostly the same as in the original jointree algorithm, except the individual messages are redefined. For any cluster i, the message λ_{ij} sent from cluster i to another cluster j is defined as summing out $\mathbf{Y}$ followed by maximizing $\mathbf{X}$ from the potential of cluster i, where $\mathbf{X} \in \mathbf{M}$ and $\mathbf{Y} \in \mathbf{S}$ are variables that appear in cluster i but not in cluster j. After message passing on the clique tree is complete, for any cluster with maximization variables $\mathbf{X} \in \mathbf{M}$ and summation variables $\mathbf{Y} \in \mathbf{S}$, summing out $\mathbf{Y}$ followed by maximizing $\mathbf{X}$ produces an upper bound on the probability of the MAP solution. Furthermore, for any variable $X_i \in \mathbf{X}$, summing out $\mathbf{Y}$ and maximizing $\mathbf{X} - \{X_i\}$ produces upper bounds for all values of X_i.

Park and Darwiche use these upper bounds in a DFBnB search to solve the MAP problem. The nodes of the search tree represent partial instantiations of the MAP variables $\mathbf{M}$. The root node corresponds to the empty instantiation, and the leaves correspond to different complete instantiations of the MAP variables. For each internal node of the tree, its successor nodes are determined by instantiating a single variable that had not previously been instantiated, and there is one successor node for each possible value of that variable. Because the clustering algorithm computes upper bounds for all MAP variables simultaneously, dynamic ordering can be used in selecting the next MAP variable to instantiate. Park and Darwiche select the variable whose states have the most asymmetric bounds.

Another approach to computing upper bounds for MAP is mini-bucket elimination (Dechter & Rish 2003). It attempts to use variable elimination to solve the original MAP problem, but if an elimination operation generates a potential that is too large, it generates a set of smaller potentials that approximate the large potential. Experimental results show that mini-bucket upper bounds are much looser than jointree upper bounds (Park & Darwiche 2003). Moreover, the mini-bucket method does not produce simultaneous bounds for multiple variables, and thus it does not allow dynamic ordering when instantiating variables. Finally, mini-bucket upper bounds are not joint bounds, as defined in the following section.

Joint Bounds

The central observation of this paper is that the jointree or clustering method for computing upper bounds on MAP probabilities, described above, has a property that can be leveraged to improve search efficiency. We begin with the following definition.

Definition 1 *In a MAP problem, a potential $\phi(\mathbf{X})$ is a* joint bound *for $\mathbf{X}$ if, for any instantiation $\mathbf{x}$ of $\mathbf{X}$, the following inequality holds*

$$\phi(\mathbf{x}) \geq \max_{\mathbf{M}-\mathbf{X}} \sum_{\mathbf{S}} P(\mathbf{M} - \mathbf{X}, \mathbf{x}, \mathbf{S}, \mathbf{E} = \mathbf{e}).$$

At the end of message passing, each cluster on the clique tree contains a potential ψ over its maximization variables $\mathbf{X} \in \mathbf{M}$ and summation variables $\mathbf{Y} \in \mathbf{S}$. More importantly, the potential has already factored in the cluster's original potential and all incoming messages. Now, if we only sum out the variables in $\mathbf{Y}$, we get a potential ϕ over $\mathbf{X}$. We have the following theorem for potential ϕ.

Theorem 2 *When message passing is over, for each cluster on the clique tree with final potential ψ over maximization variables $\mathbf{X} \in \mathbf{M}$ and summation variables $\mathbf{Y} \in \mathbf{S}$, the following potential is a joint bound for $\mathbf{X}$:*

$$\phi(\mathbf{X}) = \sum_{\mathbf{Y}} \psi \ . \tag{2}$$

Proof*: Without loss of generality, let us focus on the root cluster (a clique tree can be rearranged by using any cluster as the root). In computing the root potential, messages are computed starting from the leaves of the tree and passed through all the other clusters until reaching the root. This is equivalent to recursively shifting maximizations inside summations. In particular,*

the root cluster provides a way to shift the maximizations over $\mathbf{M} - \mathbf{X}$ *inside the summations over* $\mathbf{Y}$ *and generate upper bounds. By simple induction,* $\mathbf{M}-\mathbf{X}$ *can be further mixed with* $\mathbf{S}-\mathbf{Y}$ *according to the clique tree to relax the bounds further. According to Theorem 1, after summing out* $\mathbf{Y}$*, the potential* $\phi(\mathbf{X})$ *provides upper bounds for all the configurations of* $\mathbf{X}$*.* □

If we continue to maximize variables from ϕ, we get upper bounds for individual variables, which we call *individual bounds* to distinguish them from joint bounds. These are the bounds used in the DFBnB algorithm of Park and Darwiche. However, no further maximization is necessary. As we show below, the joint bounds can be directly used in MAP search.

MAP Search Using Joint Bounds

To leverage joint bounds to improve the efficiency of MAP search, we only need to make a simple change to the function that generates the successors of a node in the search tree. Using individual bounds, the successor function instantiates one variable at a time. To leverage joint bounds, we modify the successor function so that it can instantiate multiple variables at a time.

To illustrate the difference, we use a three-binary-variable MAP problem with fixed variable ordering; later, we discuss how to use dynamic variable ordering. Figure 1 shows the search trees that are generated when different joint bounds are available. When only individual bounds are available, one variable is instantiated at a time and the search tree shown in Figure 1(a) is generated. If joint bounds over the variables A and B are available, these two variables can be instantiated at the same time when generating the successors of the root node. The result is the search tree shown in Figure 1(b). When joint bounds over all three variables are available, the search tree shown in Figure 1(c) is generated.

By allowing multiple variables to be instantiated at the same time when generating the successors of a node, joint bounds allow substantial reduction in the time spent computing bounds. Before the successors of a node are generated, the clique tree has to be reinitialized with correct values for evidence and instantiated variables, and with correct potentials for all the clusters, and inference needs to be performed on the clique tree, in order to compute the bounds. This is very time (and memory) intensive, and the larger the clique tree, the more expensive it is. Joint bounds make it possible to reduce this overhead. For example, in order to expand the search trees in Figure 1, we need to compute bounds once for each internal node, which means 7 times in **(a)**, 5 times in **(b)**, and only once in **(c)**. Moreover, computing the joint bounds for the 8 successors of the root node in Figure 1(c) is no more expensive – in fact, it is less expensive – than computing the individual bounds for the 2 successors of the the root node in Figure 1(a), because individual bounds are derived from joint bounds.

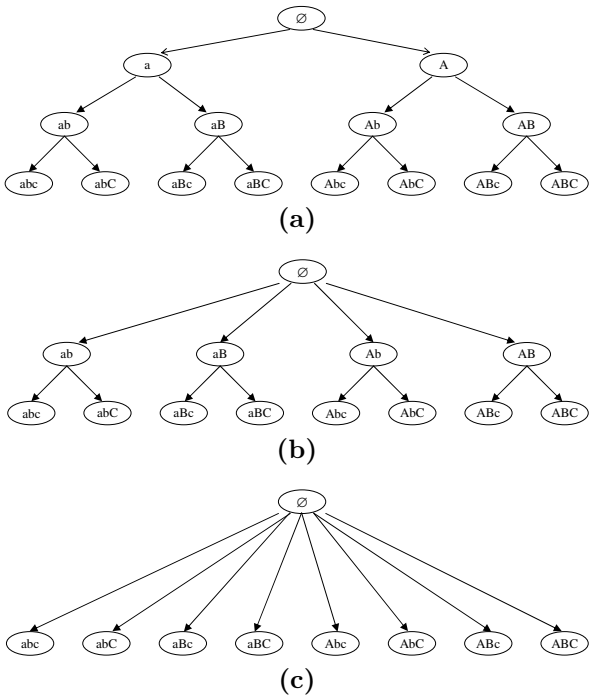


Figure 1: Search trees for a three-binary-variable MAP problem (a) full search tree instantiate one variable at a time, (b) search tree utilizing two-variable joint heuristic, (c) search tree utilizing three-variable joint heuristic.

In fact, if we expand the search tree using a static ordering, we have the following theoretical result:

Theorem 3 *If a static ordering is used in expanding a search tree, using joint bounds will generate no more nodes than using individual bounds.*

Proof*: First, note that as a search tree goes deeper, upper bound values are monotonically nonincreasing. This is true because instead of maximizing or summing over a variable, we fix the variable to a particular value. Furthermore, the heuristic values are the same for the same node in both search trees using joint bounds and individual bounds, because the same evidence has been instantiated for the node. Given these results, we can show that a node* N *expanded in the joint-bound search tree must be expanded by the individual-bound tree. If not, there must be an ancestor* M *of* N *that is pruned in the individual-bound tree. Let the probability of the current best solution be* f_M *at the moment of pruning. Since a fixed ordering is used, the same set of potential solutions must have been visited or pruned. Therefore,* $f_N = f_M > h_M$*. However, since* $h_M \geq h_N$*, we have* $f > h_N$*, which contradicts the assumption that* N *is not pruned in the joint-bound search tree.* □

With dynamic ordering, we cannot similarly guarantee that use of joint bounds will not make the search less efficient. Using joint bounds, and instantiating multiple variables at a time, some nodes may be generated that

would not have been generated if the search algorithm instantiated a single variable at a time and applied bounds earlier. However, experiments show that this adds little or no overhead to DFBnB search. Moreover, there is also a potential advantage in using joint bounds and instantiating multiple variables at a time. DFBnB uses the bounds of the successor nodes to choose which subtree to explore next, and a poor choice can significantly slow the search. By selecting the subtree based on the joint bounds after instantiating multiple variables, instead of the individual bounds after instantiating a single variable, DFBnB can select the best subtree to explore based on additional information, and this can improve search efficiency. Therefore, there is a tradeoff between the advantage of taking larger steps in expanding the search tree versus the advantage of generating intermediate search nodes to guide dynamic ordering.

Dynamic Variable Ordering Using joint bounds, it is still possible to dynamically choose which variable(s) to instantiate next. We use the following heuristics to make this choice. First, we prefer to use joint bounds when they are available. If more than one is avaiable, we choose the one that is most asymmetric according to (Park & Darwiche 2003). If B_V is the best upper bound obtained for a configuration of variables V, and T_V is the sum of bounds that are better than the best solution found so far, we choose the joint bound that maximizes the ratio B_V/T_V. Note that we do not always favor joint bounds with more variables, due to the tradeoff discussed previously. When only individual bounds are available, we use the same criterion to choose a single variable to instantiate.

Experimental Results

Table 1 compares the performance of DFBnB search using individual and joint bounds in solving a range of benchmark Bayesian networks. For each network, we generated 50 random test cases with all root nodes as MAP variables and all leaf nodes as evidence variables. For each test case, the states for the evidence variables were randomly selected (making sure that their joint probability was non-zero). The experiments were performed on a 3.2GHz processor with 3.25G RAM.

Solved Test Cases

The first group of results in Table 1 are for the test cases that could be solved exactly within the time limit. These results show that network size alone is not a reliable predictor of problem difficulty. Munin4 and CPCS360 are large networks with hundreds of variables, but their MAP problems are relatively easy to solve. The reason, we believe, is that these networks have a simple, layered structure. By contrast, Barley and Mildew are small networks, but their MAP problems are very hard to solve. Among the reasons they are difficult to solve, Barley has one node with 67 states, and Mildew has very little asymmetry in its CPTs, and both factors contribute to a very large search space.

The results show that using joint instead of individual bounds significantly improves the search efficiency of DFBnB, especially in solving CPCS179, CPCS360, Hailfinder, Munin4, and Water. In some cases, it makes DFBnB several times faster. In addition, it helps DFBnB find the best solution faster. This is useful for problems that are too difficult to solve exactly. Finally, the results for Hailfinder show the best solution is often found very early in the search, and most search time is spent proving the optimality of the solution.

Results for difficult networks

For several of the benchmark Bayesian networks, including Andes, Barley, Diabetes, Mildew, and Pigs, the test cases could not be solved exactly within a time limit of 40 minutes. For these networks, we report results for two experiments. For each network in the experiments, we generated 10 test cases.

In the first experiment, we used as many MAP variables as possible while still allowing the test cases to be solved within the time limit. The results are presented in the second group in Table 1. The third column shows the actual number of MAP variables selected from the total number of root nodes (which is shown in parentheses). Again, we used all leaf variables as evidence. Because there are not many MAP variables in these cases, fewer joint bounds are available, and yet using joint bounds still improves search efficiency for some of the networks. However, we do notice that for the Pigs network, the search is slightly slower using the joint bounds. This reflects the potential drawback that we discussed previously: joint bounds may generate more nodes.

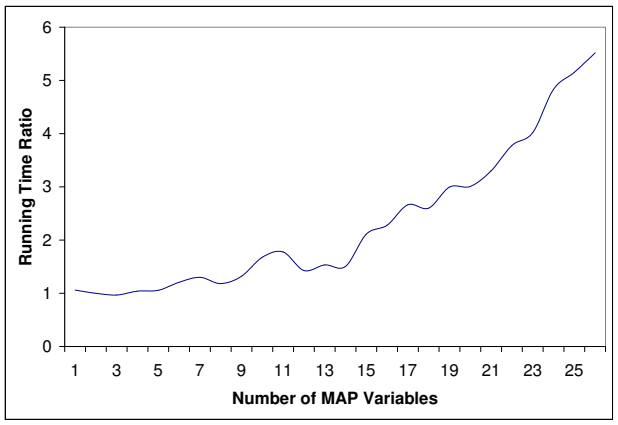
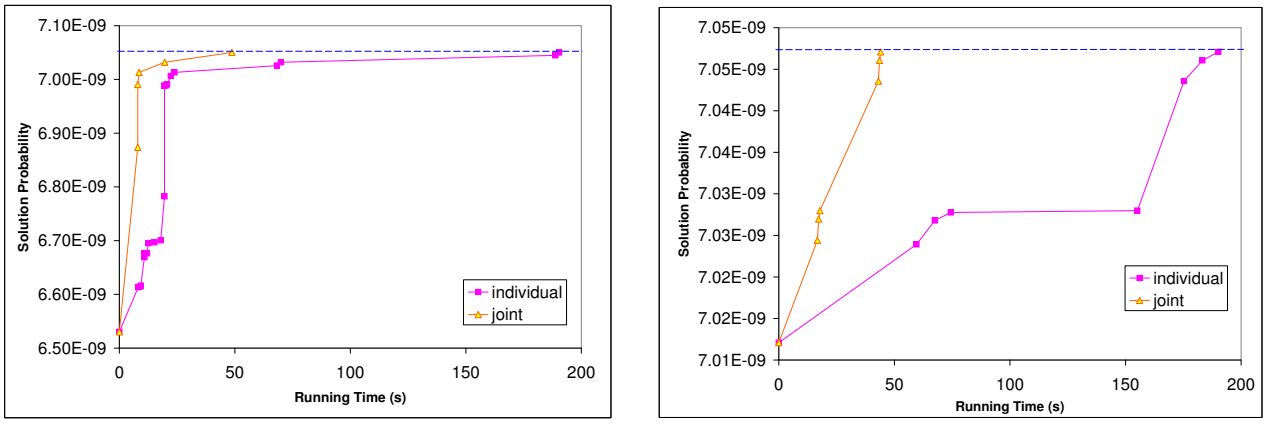


Figure 3: Ratio of the running time of DFBnB using individual bounds to the running time of DFBnB using joint bounds in solving the MAP problem for the Munin4 network, as a function of the number of MAP/evidence variables.

	Domain characteristics			Total Time		Solution Time	
	total variables	MAP variables	average BF	individual	joint	individual	joint
Insurance	27	2	2.2	2	2	<1	<1
Carpo	61	15	2	4	3	<1	<1
Plarge10	50	4	25	3	3	<1	<1
Alarm	37	12	2.2	2	1	2	<1
Hepar II	70	9	2.2	4	4	1	1
Win95pts	76	34	2	10	3	8	2
BN78	54	13	2	54	47	39	23
BN79	54	13	2	62	46	31	23
CPCS179	179	12	2.1	99	29	96	26
CPCS360	360	25	2	957	14	953	11
Water	32	8	3.6	1,241	180	1,230	175
Hailfinder	56	17	3.8	10,142	7,365	142	98
Munin4	1041	259	4.3	23,153	4,096	23,150	4,096
BN0	100	13	2	113,549	78,535	73,010	48,077
BN10	85	15	2	131,249	46,436	84,177	22,143
Barley	48	5(10)	12.4	123,499	112,239	2,706	2,021
Diabetes	413	10(76)	13	272,224	272,184	3,001	3,000
Mildew	35	10(16)	4.4	361,491	353,936	57,055	20,313
Pigs	441	80(145)	3	458,120	495,381	95	76
Andes	223	25(89)	2	573,131	470,838	121,533	66,637
Barley	48	10	12.4	-	-	13,805	9,883
Diabetes	413	76	13	-	-	24,766	20,594
Mildew	35	16	4.4	-	-	190,160	44,110
Pigs	441	145	3	-	-	648	359
Andes	223	89	2	-	-	466,184	242,913

Table 1: Comparison of the average running times of DFBnB using individual and joint bounds in solving MAP problems for benchmark Bayesian networks. Time is measured in milliseconds. In labeling the columns, 'BF' is an abbreviation for branching factor; 'total time' is the time it takes the algorithm to converge to an exact solution; and 'solution time' is the time it takes the algorithm to find what turns out to be the best solution.



(a) **(b)**

Figure 2: Comparison of DFBnB using individual and joint bounds in finding the best solution for two test cases of the Mildew network. (The horizontal line shows the best solution found.)

In the second experiment, we used all the root nodes as MAP variables. Since this prevented the problems from being solved within the time limit, the third group of results in Table 1 only reports the average time the algorithms took to find the best solution. The results show that using joint bounds allows DFBnB to find the best solution faster. Figure 2 shows two typical test cases for the Mildew network. In the first, using individual bounds almost keeps pace with using joint bounds at first, but the best solution is found much sooner using joint bounds. Because both algorithms found the same best solution long before running out of time, we suspect that most of their search time is spent in trying to prove the optimality of the solution. The difference is more dramatic in the second case. DFBnB using joint bounds found the best solution after only 40 seconds, whereas DFBnB using individual bounds did not find the same solution until after 200 seconds. Similar results were obtained for the other four networks, although the first solution found for these networks was often the best ever found.

Effect of Number of MAP Variables

The number of MAP variables affects the number of joint bounds that are available. If there are very few MAP variables, a clique tree may only have individual bounds. The number of joint bounds increases with the number of MAP variables. Figure 3 shows the effect of the number of MAP variables on the relative benefit of using joint bounds in solving the Munin4 network. As the number of MAP/evidence variables increases, the relative advantage of using joint bounds also increases.

Related Work

Many algorithms have been proposed to solve the MAP problem in Bayesian networks. We briefly review some of them, beginning with two exact algorithms.

DFBnB using jointree (clustering) upper bounds (Park & Darwiche 2003): This work is the most closely related to our own and has been discussed throughout this paper.

DFBnB using arithmetic circuit upper bounds (Huang, Chavira, & Darwiche 2006): This work proposes a new way to compute upper bounds for MAP by compiling Bayesian networks into arithmetic circuits (Chavira & Darwiche 2005). It is designed for Bayesian networks that have a lot of local structure, and may not be effective for other networks. Currently, it requires DFBnB to use fixed variable ordering.

These exact algorithms are most effective for finding optimal solutions in small to medium-sized networks, or networks with a lot of local structure. DFBnB can also be applied to more difficult MAP problems that cannot be solved exactly, since DFBnB often finds good solutions quickly and can be stopped before convergence. Other approximate search algorithms for difficult MAP problems include the following.

Local Search (Park & Darwiche 2001): Hill climbing has been shown to be effective in finding approximate solutions to MAP problems, and often finds optimal or close-to-optimal solutions, although it does not provide any guarantee of the quality of a solution. Tabu search makes the local search even more effective.

Genetic Algorithm (de Campos, Gamez, & Moral 1999): This work uses the junction tree algorithm as an evaluation function in a genetic algorithm that solves MAP problems.

Annealed MAP (Yuan, Lu, & Druzdzel 2004): This algorithm applies Markov Chain Monte Carlo methods (Geman & Geman 1984) and simulated annealing (Kirkpatrick, Gelatt, & Vecchi 1983) to solve MAP problems. It has been shown to be effective on large Bayesian networks.

DWA* (Sun, Druzdzel, & Yuan 2007): Since hill climbing often produces surprisingly good solutions, this algorithm solves MAP problems using A* search guided by a dynamically weighted greedy heuristic. Although the algorithm is not guaranteed to find an optimal solution, it was shown to be effective on many Bayesian networks.

Conclusion

The central observation of this paper is that the jointree (or clustering) upper bound proposed in (Park & Darwiche 2003) for the MAP problem produces joint bounds for groups of variables. We showed how to use these joint bounds to significantly improve the efficiency of a depth-first branch-and-bound search algorithm for solving the MAP problem. The improvement of search efficiency is due to a reduction in the number of times that upper bounds need to be computed, and we demonstrated the improvement on a range of benchmark Bayesian networks. Our results show that leveraging joint bounds in this way not only improves the efficiency with which DFBnB finds optimal solutions, it also enables DFBnB to find good (but possibly suboptimal) solutions faster.

Although our approach reduces the number of times upper bounds need to be computed, computing the jointree upper bound is still the bottleneck of the algorithm. Further improvement of search efficiency will require finding more efficient techniques for computing these bounds. We are also interested in whether similar joint bounds can be exploited for other search problems.

Acknowledgements

We thank the anonymous reviewers for their insightful comments that have led to improvements in the paper. All experimental data have been obtained using SMILE, a Bayesian inference engine developed at the Decision Systems Laboratory and available at `http://genie.sis.pitt.edu`.

References

Chavira, M., and Darwiche, A. 2005. Compiling bayesian networks with local structure. In *Proceedings*

of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05), 1306 1312.

de Campos, L. M.; Gamez, J. A.; and Moral, S. 1999. Partial abductive inference in bayesian belief networks using a genetic algorithm. *Pattern Recognition Letters* 20:12111217.

Dechter, R., and Rish, I. 2003. Mini-buckets: A general scheme for approximating inference. *Journal of ACM* 50(2):1–61.

Geman, S., and Geman, D. 1984. Stochastic relaxations, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6(6):721–742.

Huang, J.; Chavira, M.; and Darwiche, A. 2006. Solving map exactly by searching on compiled arithmetic circuits. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, 143148.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* (4598):671–680.

Lauritzen, S. L., and Spiegelhalter, D. J. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B (Methodological)* 50(2):157–224.

Park, J. D., and Darwiche, A. 2001. Approximating MAP using local search. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI–01)*, 403–410.

Park, J. D., and Darwiche, A. 2003. Solving MAP exactly using systematic search. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI–03)*, 459–468.

Park, J. D. 2002. MAP complexity results and approximation methods. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI–02)*, 388–396.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Roth, D. 1996. On the hardness of approximate reasoning. *Artificial Intelligence* 82(1-2):273–302.

Shimony, S. E. 1994. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence* 68:399–410.

Sun, X.; Druzdzel, M. J.; and Yuan, C. 2007. Dynamic weighting A^* search-based MAP algorithm for Bayesian networks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2385–2390.

Yuan, C.; Lu, T.; and Druzdzel, M. J. 2004. Annealed MAP. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI–04)*, 628–635.