Efficient Computation of Jointree Bounds for Systematic MAP Search

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

The MAP (maximum a posteriori assignment) problem in Bayesian networks is the problem of finding the most probable instantiation of a set of variables given partial evidence for the remaining variables. The state-of-the-art exact solution method is depth-first branch-and-bound search using dynamic variable ordering and a jointree upper bound proposed by Park and Darwiche [2003]. Since almost all search time is spent computing the jointree bounds, we introduce an efficient method for computing these bounds incrementally. We point out that, using a static variable ordering, it is only necessary to compute relevant upper bounds at each search step, and it is also possible to cache potentials of the jointree for efficient backtracking. Since the jointree computation typically produces bounds for joint configurations of groups of variables, our method also instantiates multiple variables at each search step, instead of a single variable, in order to reduce the number of times that upper bounds need to be computed. Experiments show that this approach leads to orders of magnitude reduction in search time.

1 Introduction

The MAP (maximum a posteriori hypothesis) problem in Bayesian networks is to find the most probable configuration of a set of variables (the explanatory variables) given partial evidence on the remaining set of variables (the observed or evidence variables). MAP inference has received much attention in Bayesian network research and has many practical applications. For example, 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.

The state-of-the-art exact solution method for MAP is depth-first branch-and-bound (DFBnB) search using a jointree 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-intensive. Our experiments show that more than 95% of search time is devoted to computing these bounds.

In this paper, we introduce a combination of techniques for efficient computation of jointree bounds. We first describe an *incremental* bound computation scheme that only computes relevant bounds at each step of the search. In the approach of Park and Darwhiche [2003], a jointree is fully reevaluated at each search step. We point out that, with a static variable ordering, we only need to evaluate a very small portion of the jointree at each step to get the necessary upper bounds for the next search step. We also cache potentials of the jointree incrementally during the forward search and restore them in the reverse order when backtracking, to further improve search efficiency. Finally, we observe that the 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 further improve search efficiency by instantiating multiple variables at each step of the DFBnB search, instead of a single variable, which reduces the number of times upper bounds need to be computed. The use of joint bounds also reduces the extra memory requirements of our method of incremental bounds computation.

We demonstrate the effectiveness of these techniques in solving a range of benchmark Bayesian networks. Experimental results show an orders-of-magnitude improvement in the efficiency of systematic MAP search, and this advantage grows with the size of the problem.

2 Upper bounds for MAP search

The MAP (maximum a posteriori assignment) 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,

$$\hat{\mathbf{m}}_{MAP} = \arg\max_{\mathbf{M}} \sum_{\mathbf{S}} P(\mathbf{M}, \mathbf{S}, \mathbf{E} = \mathbf{e}) \;, \tag{1}$$

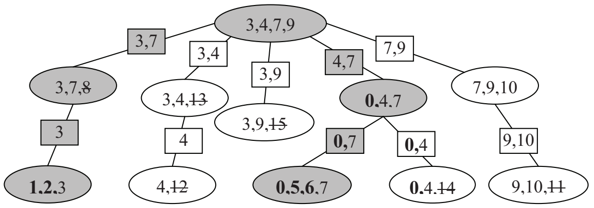

Figure 1: An example jointree for upper bound computation.

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].

Jointree upper bound In Equation (1), 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. But the summations and maximizations are not commutable. As a result, the complexity of variable elimination-based methods for solving MAP depends on the constrained treewidth [Dechter and Rish, 2003; Park, 2002]. Theoretically, a jointree satisfying the constrained ordering can be used to solve MAP problems exactly using a modified jointree algorithm [Lauritzen and Spiegelhalter, 1988]. However, the approach is typically infeasible because such jointrees are often too large to be constructed successfully.

If the orderings among the summations and maximizations is relaxed, however, an upper bound on the probability of a MAP solution can be computed. The following theorem is due to Park and Darwiche [2003].

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:*

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

Based on the result, Park and Darwiche [2003] compute upper bounds for MAP search using the jointree algorithm [Lauritzen and Spiegelhalter, 1988] with redefined messages.

Figure 1 shows an example of a jointree. The oval nodes denote clusters, the square nodes denote separators, and the numbers indicate variables. Evidence can be entered to the jointree by instantiating their values in associated clusters, shown as the variables with strikethroughs. Then, the jointree can be used to compute upper bounds for MAP search by passing redefined messages. For any two clusters $C1$ and $C2$ with separator S, and let $\mathbf{M1}$ be the MAP variables in $C1$, the message sent from $C1$ to $C2$ is redefined as

$$\phi'_{1 \to 2} = \max_{\mathbf{M1} \backslash S} \sum_{C1 \backslash \mathbf{M1} \cup S} \phi(C1) \prod_{i \in \mathrm{NB}(1) \backslash \{2\}} \phi'_{i \to 1}, \qquad (2)$$

where $C1 \backslash S$ are the variables in $C1$ but not in S, and $\mathrm{NB}(1)$ are all the neighboring clusters of $C1$. Then, a full message propagation throughout the jointree can be applied to evaluate the jointree. After the evaluation, each cluster can be marginalized to get upper bounds for its associated MAP variables. For example, suppose $X \in \mathbf{M1}$, then

$$U(X) = \max_{\mathbf{M1} \backslash \{X\}} \sum_{C1 \backslash \mathbf{M1}} \phi'(C1). \qquad (3)$$

It can be shown that $U(X)$ for any value x provides an upper bound for consistent joint configurations with X being x. Note that a full jointree evaluation produces upper bounds for all the MAP variables simultaneously.

Park and Darwiche use these upper bounds in a depth-first branch-and-bound (DFBnB) search algorithm 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. Figure 2(a) shows an example search tree for three MAP variables.

The simultaneous upper bounds for all MAP variables computed by the jointree evaluation allows the use of dynamic variable ordering. To select the next MAP variable to instantiate, Park and Darwiche select the variable whose states have the most asymmetric bounds.

Related work Another approach to computing upper bounds for MAP is mini-bucket elimination [Dechter and Rish, 2003]. It tries 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 and Darwiche, 2003].

More recent work computes upper bounds by compiling Bayesian networks into arithmetic circuits [Chavira and Darwiche, 2005; Huang *et al.*, 2006]. This approach is designed for Bayesian networks that have a lot of determinism and local structure; in this case, compilation can generate more compact representations for upper bound computation. The approach may not be as effective for other networks.

For difficult MAP problems that cannot be solved exactly, DFBnB can often find good solutions without running to convergence. There are also other search algorithms that can find approximate solutions for MAP, including local search [Park and Darwiche, 2001], genetic algorithms [de Campos *et al.*, 1999], simulated annealing [Yuan *et al.*, 2004] and weighted A* guided by a non-admissible heuristic [Sun *et al.*, 2007]. In this paper, we focus on exact search algorithms.

3 Incremental jointree bounds

The systematic MAP search algorithm of Park and Darwiche [2003] requires full evaluation of a jointree at each node of the search tree in order to compute bounds. We next describe a combination of techniques that allows efficient, incremental computation of these bounds.

3.1 Incremental bounds computation

First, we observe that, if a static variable ordering is used in MAP search, we can use an *incremental* bounds computation scheme to compute only relevant upper bounds at each search step. We illustrate the idea with an example. In Figure 1, let the bold-face variables be the MAP variables, and let the static search ordering be $V1$,$V2$,$V0$,$V5$,$V6$. After instantiating $V1$, we need to compute bounds for $V2$ and instantiate it. To achieve that, it is unnecessary to evaluate the whole jointree. Since $V2$ is in the same cluster as $V1$, we only need to enter the state of $V1$ as evidence to the cluster and marginalize it to get upper bounds for $V2$. After $V1$ and $V2$ are both instantiated, their values are entered as evidence to the cluster. Messages can then be sent to the other parts of the jointree to get upper bounds for the remaining MAP variables. However, since the next variable is $V0$, we only need to send messages along the shaded path from cluster $(1, 2, 3)$ to cluster $(0, 4, 7)$. None of the other parts of the jointree need to be involved in the propagation. Therefore, in our incremental bounds computation scheme, we propose to perform only the necessary message propagations to get upper bounds for the next instantiating variable at each step. The incremental scheme requires the DFBnB algorithm to use a static variable ordering in MAP search. Although the dynamic ordering used in [Park and Darwiche, 2003] can exploit asymmetry early, we expect the savings from the incremental bounds computation to offset any additional search required.

A simple comparison of the time complexities of full and incremental upper bound computation shows the advantage of the incremental approach. Let N be the size of a jointree, let K be the maximum number of MAP variables in any cluster, let L be the maximum cluster size, and let D be the maximum distance between two neighboring clusters that need to be searched in a chosen static ordering. The time complexity of one iteration of bounds computation is on the order of $N * K * L$ for full bounds compared to $D * L$ for incremental bounds. Note that typically $D \ll N * K$.

3.2 Efficient backtracking

DFBnB search often needs to backtrack to a previously-generated search node. This requires retracting the corresponding jointree to its state when the node was generated. Still using Figure 1 as an example, suppose we are at a search node where $V1$, $V2$ and $V0$ are instantiated and we need to backtrack to a search node where only $V1$ and $V2$ are instantiated, and the state of $V2$ has been entered as evidence to the cluster $(1, 2, 3)$. The shaded path from cluster $(1, 2, 3)$ to $(0, 4, 7)$ has a new set of cluster and separator potentials as a result of the incremental bounds computation. One way to retract the jointree is to reinitialize it with correct evidence and perform a full jointree evaluation, which is the *full jointree bounds* used in [Park and Darwiche, 2003]. Instead, our incremental approach caches the cluster and separator potentials of the jointree that are to be modified in the bound computation in the order that they are changed. In the above example, the changed parts are the potentials of the clusters and separators in the shaded path from cluster $(1, 2, 3)$ to $(0, 4, 7)$. When backtracking, we simply restore the potentials in the reverse order and roll back the changes, which retracts the jointree to the previous state much more efficiently.

Our backtracking scheme requires caching potentials and thus increases the size of the upper-bound jointree. However, we can bound the extra memory it requires by bounding the additional memory required by each cluster as follows.

Theorem 2 *The maximum number of cached copies that a cluster of the jointree may need is equal to the sum of three numbers: the number of child branches of the cluster with MAP variables to search; the number of MAP variables to search on the cluster; and* 1 *if there exists any non-descendant MAP variables to search on the jointree,* 0 *otherwise.*

Proof*: If a child branch of the cluster has MAP variables to search, that branch may need to send message to the cluster after being searched. The cluster potential needs caching before being overridden by each incoming message. The potential also needs caching before each MAP variable on the cluster is searched and entered as evidence to the jointree. Finally, non-descendant MAP variables may need to send a message to this cluster through its parent.* □

For example in Figure 1, cluster $(3, 4, 7, 9)$ has two child branches containing MAP variables. But the cluster itself contains no MAP variables. It is the root of the jointree and has no non-descendant MAP variables. Therefore, the cluster only needs enough extra memory for two cached copies at most. Note that Theorem 2 considers the general case in order to establish an upper bound. Depending on the search order of the MAP variables, the actual memory requirement could be less. For example, cluster $(3, 4, 7, 9)$ only needs one cached copy if the search order is $1, 2, 0, 5, 6$. The bounded memory requirement can be further reduced with several enhancements to be discussed in the next section.

There are many possible static variable orderings. We choose the post-order traversal sequence of the jointree in order to preserve locality in each branch of the jointree and maximally make use of joint bounds to be defined in the next section. In Figure 1, for example, the post-order traversal sequence is 1,2,0,5,6, which reduces to two joint bounds $U(1, 2)$ and $U(0, 5, 6)$.

Combining the techniques of incremental bounds computation and efficient backtracking, we have a new method that we refer to as *incremental jointree bounds*. The method is related to the query-driven message-passing technique defined in [Huang and Darwiche, 1996], especially its *evidence update* step. It can be viewed as repeated application of the evidence update step for computing the bounds incrementally. The major difference is that the *evidence retraction* step of query-driven message-passing proposes to reinitialize the jointree with new evidence and perform full reevaluation. By contrast, our method of caching potentials for backtracking avoids the expensive full reevaluations and is the key for enabling efficient backtracking search.

3.3 Enhancements

The extra memory required by our incremental jointree bounds can become an issue, especially for large Bayesian networks. In this section, we discuss several techniques to reduce the memory requirement.

First, after initial evidence is entered and propagated, only parts of the jointree that contain MAP variables need to be involved in further message passing. Therefore, only a small fraction of the jointree needs caching in practice. Furthermore, we note that a cluster containing MAP variables can be left out of the incremental message passing if these MAP variables have already been searched somewhere else. In Figure 1, variable $V0$ also appears in cluster $(0, 4, 14)$. If we search cluster $(0, 4, 7)$ first, variable $V0$ is already searched before we come to $(0, 4, 14)$. There is no need to involve this branch in incremental bounds computation. We can use a pre-processing step before the start of the search to mark the clusters and branches that have MAP variables to search and need to be involved in incremental bounds computation. In Figure 1, these branches correspond to the shaded part.

Second, we note that the reason for caching the potentials of the jointree is for backtracking. There is no need for caching if backtracking is unnecessary. One such situation is when the search stack has only one open search node, in which case we do not need to perform any caching. Only when the stack has more than one search node will we cache potentials.

Third, the jointree method for computing upper bounds on MAP probabilities has a property that can be leveraged to further reduce the memory requirement. We introduce the concept of a *joint bound*, which is a bound for a group of variables.

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 jointree 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 3 *When message passing is over, for each cluster on the jointree 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* $\mathbf{X}$*:*

$$\phi(\mathbf{X}) = \sum_{\mathbf{Y}} \psi \ . \tag{4}$$

Proof*: Without loss of generality, let us focus on the root cluster (a jointree can be rearranged with 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 according to Theorem 1. By simple induction,* $\mathbf{M}-\mathbf{X}$ *can be further mixed with* $\mathbf{S}-\mathbf{Y}$ *according to the jointree to relax the bounds further. After summing out* $\mathbf{Y}$*, the potential* $\phi(\mathbf{X})$ *provides upper bounds for all the configurations of* $\mathbf{X}$. □

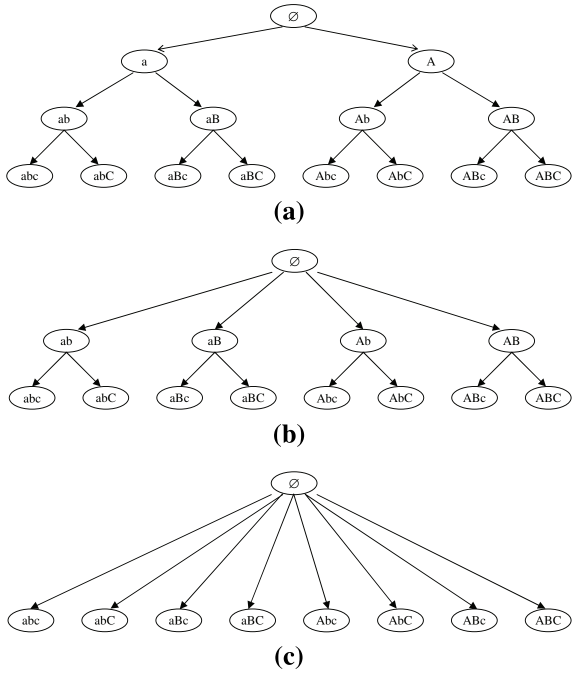

Figure 2: Search trees for a three-binary-variable MAP problem, where (a) shows the full search tree with one variable instantiated at a time, (b) shows the search tree that results from using a two-variable joint bound, and (c) shows the search tree that results from using a three-variable joint bound.

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; the joint bounds can be directly used in MAP search.

To leverage joint bounds, we 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. Figure 2 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 2(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 2(b).

When joint bounds over all three variables are available, the search tree shown in Figure 2(c) is generated.

The use of joint bounds can further reduce the bounded extra memory requirement of incremental jointree bounds. Since we search and instantiate all the MAP variables in a cluster at once, we only need to cache the potential once before the instantiation. We have the following theorem.

Theorem 4 *The maximum number of cached copies that a cluster needs is the sum of three numbers: the number of child branches of the cluster containing MAP variables to search;* 1 *if there are MAP variables to search in this cluster,* 0 *otherwise; and* 1 *if there exist non-descendant MAP variables to search,* 0 *otherwise.*

It turns out there is another benefit from using joint bounds. By allowing multiple variables to be instantiated at once when generating the successors of a node, joint bounds allow reduction in the size of a search tree. Before the successors of a node are generated, the jointree has to be entered with correct evidence, and inference needs to be performed to compute other bounds. This is very time-intensive, and the larger the jointree, 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 2, 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 2(c) is no more expensive – in fact, it is less expensive – than computing the individual bounds for the two successors of the the root node in Figure 2(a), because individual bounds over A are derived from the joint bounds $U(A, B, C)$.

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

Theorem 5 *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. Then, we claim 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, however, we cannot similarly guarantee that use of joint bounds will not lead to generation of more search nodes. 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 [Yuan and Hansen, 2008].

4 Experimental Results

We implemented our algorithm and the previous state-of-the-art algorithm [Park and Darwiche, 2003] using the SMILE Bayesian network package [Druzdzel, 1999]. Experiments were performed on a 3.2 GHz processor with 4 gigabytes of RAM running a 64-bit version of Windows XP.

Table 1 compares the performance of our algorithm, which is DFBnB using incremental bounds and static ordering (DFBnB+IS), to the performance of the Park and Darwiche algorithm, which uses the full jointree bound and dynamic ordering (DFBnB+FD). The Bayesian networks used as test problems are divided into two groups. The first group contains networks that are relatively easy to solve, while the second group is much more difficult. For the first group, 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 sampled from the prior distributions, which ensures their joint probability is non-zero. For the second group, we generated 10 test cases with as many root nodes as MAP variables so that they are solvable by both algorithms within 30 minutes. We still use all leaf nodes as evidence variables for these cases.

Results for easy networks The first group of results in Table 1 are for the easy networks. These results show that using incremental instead of full jointree bounds significantly improves the search efficiency of DFBnB for most networks. In some cases, it makes DFBnB orders of magnitude faster.

For all the networks except CPCS360, the number of search nodes generated by DFBnB+IS is smaller than the number generated using DFBnB+FD. Since search using dynamic ordering usually generates fewer nodes than search using static ordering, this may seem surprising. For four of these problems (Win95pts, CPCS179, Munin4, and Water), note that the number of search nodes is less than or equal to the number of MAP nodes, which indicates that the first solution explored turned out to be an optimal solution. The results for the fifth problem (Hailfinder) can be explained in two ways. First, recall that using joint bounds reduces the depth of the search tree and thus the number of search nodes. More importantly, dynamic ordering is only an approximate heuristic and it can sometimes lead to an order that results in generation of more nodes.

The 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 rather easy to solve. For these problems, the first solutions explored turned out to be the optimal solutions. 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. Both factors contribute to a very large search space.

In solving CPCS360, DFBnB+IS was slower than DFBnB+FD. This is possible for two reasons. First, because incremental bounds require static variable ordering, more search nodes can be generated than using dynamic ordering. Second, incremental bounds incurs overhead for caching and

	Domain characteristics			DFBnB+FD			DFBnB+IS		
network	vars	MAPs	BF	time (ms)	nodes	memory (KB)	time (ms)	nodes	m-ratio
Win95pts	76	34	2	13	34	51	2	20	1.0
CPCS179	179	12	2.1	104	12	1,277	3	5	1.0
CPCS360	360	25	2	1,083	25	69,920	1,696	26	1.0
Water	32	8	3.6	1,344	8	128,512	920	5	2.1
Hailfinder	56	17	3.8	8,301	19,393	151	297	9,352	2.6
Munin4	1041	259	4.3	33,539	259	261,982	3,117	195	1.0
Barley	48	5(10)	12.4	225,687	155	414,141	27,922	337	1.2
Pigs	441	48(145)	3	1,021,734	52,425	17,745	784,976	248,364	3.2
Andes	223	27(89)	2	1,033,211	80,145	8,903	44,617	32,592	3.1
Diabetes	413	12(76)	13	1,230,844	2,068	166,856	106,787	4,567	1.7
Mildew	35	10(16)	4.4	1,656,304	3,126	209,689	41,117	5,397	1.6

Table 1: Comparison of the average running times and memory requirements of DFBnB using full and incremental jointree bounds in solving MAP problems for benchmark Bayesian networks. The column headings have the following meanings: 'vars' is the total number of variables; 'MAPs' is the number of MAP variables; 'BF' is the average branching factor; 'time' is the time (in milliseconds) it takes the algorithm to converge to an exact solution; 'nodes' is the number of search nodes generated; 'memory' is the amount of memory in kilobytes used for the jointree of the algorithms; 'm-ratio' is the the amount of memory needed in comparison to 'DFBnB+FD'; 'F' stands for full jointree bounds; 'D' stands for dynamic ordering; 'I' stands for incremental bounds; and 'S' stands for static ordering.

restoring potentials, especially when the jointree is very large.

We also empirically tested the extra memory requirements of the incremental bounds computation method. Almost all the main memory consumption of both DFBnB+FD and DFBnB+IS comes from the jointrees used for computing upper bounds. Our experiments show that memory needed for storing search nodes is at most 2% of the size of the jointree and is usually around $10^{-3}\%$. The column labeled 'memory' shows the total amount of memory in kilobytes needed by the jointree in DFBnB+FD. Some of these jointrees, such as those of Munin, Barley, and Mildew, are extremely large. The column labeled 'm-ratio' shows the ratio of the amount of memory used for the jointree in DFBnB+IS compared to DFBnB+FD. In most networks, the extra memory required is relatively small, and no more than twice that of the original jointrees. Although memory can potentially be an issue, we did not have a network that is solvable by DFBnB+FD but not DFBnB+IS because of memory.

Results for difficult networks For several benchmark Bayesian networks, including Andes, Barley, Diabetes, Mildew, and Pigs, the test cases with all root nodes as MAP variables could not be solved exactly within a time limit of 30 minutes by DFBnB+FD. For these networks, we used as many MAP variables as possible while still allowing the test cases to be solved within the time limit. This is the second group of results shown in Table 1. The column labeled MAPs shows the actual number of MAP variables selected from the total number of root nodes (which is shown in parentheses). The results show that DFBnB+IS outperforms DFBnB+FD by orders of magnitude, even when it generates more search nodes. This shows that the improved speed from using incremental bounds offsets the extra search required.

Jointree promotion We also implemented and tested the jointree promotion technique proposed by Park and Darwiche [2003], which was not evaluated separately in their paper. The idea is to push MAP variables towards the root in order to compute better upper bounds near the root, although this technique may make the bounds worse close to the leaves. Since we use static ordering, we can push the MAP variables in a specific direction to get better bounds in the beginning layers of the search tree.

Table 2 shows that the promotion technique has mixed results. For some networks, such as Pigs and Diabetes, it significantly reduces running time. But for others, it leads to increased running time, especially for Munin4. The worse performance is due to the much larger jointree that results from using promotion. For Munin4, the new jointree is almost 11.7 times larger than the original and much more time-consuming to evaluate. Note that the promotion technique increases the memory requirements of the search much more than simply using incremental bounds. Another interesting observation is that, in many cases, the extra memory required by DFBnB+ISP in comparison to DFBnB+FD is only slightly higher than DFBnB+FDP. The reason for this is that, after a jointree is promoted, a smaller fraction of the jointree needs to be involved in the incremental bounds computation.

In most cases, incremental bounds and static ordering (DFBnB+IS) outperforms full bounds enhanced by both dynamic ordering and the promotion technique (DFBnB+FDP). The improvement is even greater when we enhance DFBnB+IS with the promotion technique (DFBnB+ISP).

Effect of number of MAP variables The results shown in Table 3 demonstrate the improved scalability of our algorithm. For the five difficult networks, we generated test

Domains	DFBnB+FDP			DFBnB+ISP		
net	time (ms)	nodes	m-ratio	time (ms)	nodes	m-ratio
Win95pts	16	34	2.5	2	21	2.6
CPCS179	107	12	1.4	5	5	1.9
CPCS360	1,534	25	2.0	1,218	26	2.0
Water	1,619	8	2.8	636	5	3.8
Hailfinder	5,272	7,561	1.8	295	7,996	3.4
Munin4	717,670	259	11.7	60,047	190	11.7
Barley	246,360	137	4.6	58,281	164	10.7
Pigs	6,762	49	6.1	12,331	731	8.4
Andes	316,476	12,242	2.7	80,734	10,925	5.1
Diabetes	336,281	40	6.6	42,687	34	7.0
Mildew	1,577,515	2,867	1.1	59,461	5,362	1.7

Table 2: Comparison of MAP search using full and incremental bounds when enhanced by the jointree promotion technique. Note that 'm-ratio' is the amount of memory needed in comparison to 'DFBnB+FD', which is shown in Table 1, and 'P' stands for the jointree promotion technique.

cases with an increasing number of MAP variables. Note that the number of MAP variables affects not only the difficulty of the test cases, but the number of available joint bounds. If there are very few MAP variables, a jointree may only have individual bounds. The number of joint bounds increases with the number of MAP variables. Table 3 shows the effect of the number of MAP variables on the relative benefit of using incremental bounds in solving the difficult networks. DFBnB with incremental bounds clearly has better scalability. It solved test cases with more MAP variables for each of the networks within the time limit, especially for Diabetes, Andes, and Mildew. On the Andes network, incremental bounds could solve test cases with up to 35 MAP variables, whereas full bounds could not handle more than 27 MAP variables. Incremental bounds also solved the test cases with all root nodes as MAP variables for the Mildew network, while DFBnB with full bounds ran out of time after 10 MAP variables. We also note that the promotion technique can sometimes make the algorithm much more scalable. For the Pigs network, for example, both DFBnB+ISP and DFBnB+FDP solve much larger MAP problems.

5 Conclusion

We have developed a combination of techniques for incrementally computing jointree upper bounds in systematic MAP search. These include an incremental bounds computation scheme that avoids full jointree evaluation at each step of the search; a method for caching and restoring potentials of the upper bound jointree for efficient backtracking; and the use of joint bounds to further speed up the search. Experimental results show that these techniques make it possible for depth-first branch-and-bound search to find optimal solutions to much more difficult instances of the MAP problem than could be solved by the previous state-of-the-art algorithm.

Although our approach allows upper bounds to be computed more efficiently, computing the upper bounds 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 techniques can be exploited for other search problems.

Acknowledgements

The first author would like to acknowledge support from the National Science Foundation Research grant IIS-0842480. 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 and Darwiche, 2005] M. Chavira and A. Darwiche. Compiling bayesian networks with local structure. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, page 1306 1312, 2005.

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

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

[Druzdzel, 1999] Marek J. Druzdzel. SMILE: Structural Modeling, Inference, and Learning Engine and GeNIe: A development environment for graphical decision-theoretic models. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI–99)*, pages 902–903, Orlando, FL, July 18–22 1999.

[Huang and Darwiche, 1996] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15:225–263, 1996.

Barley

MAPs	1	2	3	4	5	6	7
DFBnB+FD	15	1,843	40,749	78,937	225,687	-	-
DFBnB+FDP	<1	2,070	4,031	8,539	246,360	-	-
DFBnB+IS	<1	31	734	4,453	27,922	828,562	-
DFBnB+ISP	<1	16	633	9,453	58,281	804,437	-

Pigs

MAPs	15	30	45	48	51	57	63	69	75	78
DFBnB+FD	868	12,828	387,328	1,021,734	-	-	-	-	-	-
DFBnB+FDP	1671	4,422	8,289	6,762	9,992	11,684	97,809	21,667	838,387	-
DFBnB+IS	331	1,562	50,906	784,976	-	-	-	-	-	-
DFBnB+ISP	368	633	2,382	12,331	2,828	52,115	310,409	85,819	1,260,362	-

Andes

MAPs	9	18	27	29	31	33	35	37	38	39
DFBnB+FD	1,437	18,953	1,033,211	-	-	-	-	-	-	-
DFBnB+FDP	428	13,187	316,476	1,318,469	-	-	-	-	-	-
DFBnB+IS	187	3,141	44,617	177,500	951,648	1,004,851	1,185,992	-	-	-
DFBnB+ISP	206	4,016	80,734	191,664	1,049,156	1,274,422	1,279,781	-	-	-

Diabetes

MAPs	8	10	12	14	16	17	18	19	20	21
DFBnB+FD	55,359	524,914	1,230,844	-	-	-	-	-	-	-
DFBnB+FDP	9,016	77,484	336,281	1,050,906	-	-	-	-	-	-
DFBnB+IS	4,984	26,500	106,787	414,749	1,209,737	-	-	-	-	-
DFBnB+ISP	672	7,851	42,687	94,367	515,797	489,334	654,718	891,344	1,109,281	-

Mildew

MAPs	2	4	6	8	10	12	13	14	15	16
DFBnB+FD	804	8,031	37,853	58,429	1,656,304	-	-	-	-	-
DFBnB+FDP	805	8,000	28,507	54,547	1,577,515	-	-	-	-	-
DFBnB+IS	172	4,032	1,320	7,468	41,117	189,844	148,398	340,016	413,312	471,054
DFBnB+ISP	195	2,953	1,366	8,015	59,461	230,078	114,320	314,967	539,609	557,640

Table 3: As a test of scalability, these results show the running times of DFBnB using full and incremental jointree bounds in solving MAP problems for benchmark Bayesian networks with an increasing number of MAP variables. The symbol ”-” indicates the algorithm exceeded the time limit of 30 minutes.

[Huang *et al.*, 2006] Jinbo Huang, Mark Chavira, and Adnan Darwiche. Solving MAP exactly by searching on compiled arithmetic circuits. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, page 143148, 2006.

[Lauritzen and Spiegelhalter, 1988] S. L. Lauritzen and D. J. Spiegelhalter. 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, 1988.

[Park and Darwiche, 2001] J. D. Park and A. Darwiche. Approximating MAP using local search. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI–01)*, pages 403–410, Morgan Kaufmann Publishers San Francisco, California, 2001.

[Park and Darwiche, 2003] J. D. Park and A. Darwiche. Solving MAP exactly using systematic search. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI–03)*, pages 459–468, Morgan Kaufmann Publishers San Francisco, California, 2003.

[Park, 2002] J. D. Park. MAP complexity results and approximation methods. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI–02)*, pages 388–396, Morgan Kaufmann Publishers San Francisco, California, 2002.

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

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

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

[Yuan and Hansen, 2008] Changhe Yuan and Eric Hansen. MAP search in bayesian networks using joint bounds. In *Proceedings of AAAI-08 Workshop on Search Techniques in Artificial Intelligence and Robotics*, pages 140–146, 2008.

[Yuan *et al.*, 2004] C. Yuan, T. Lu, and M. J. Druzdzel. Annealed MAP. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI–04)*, pages 628–635, AUAI Press, Arlington, Virginia, 2004.