Solving MAP Exactly by Searching on Compiled Arithmetic Circuits*

Jinbo Huang
Logic and Computation Program
National ICT Australia
Canberra, ACT 0200 Australia
jinbo.huang@nicta.com.au

Mark Chavira and **Adnan Darwiche**
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095 USA
{chavira, darwiche}@cs.ucla.edu

Abstract

The MAP (maximum a posteriori hypothesis) problem in Bayesian networks is to find the most likely states of a set of variables given partial evidence on the complement of that set. Standard structure-based inference methods for finding exact solutions to MAP, such as variable elimination and jointree algorithms, have complexities that are exponential in the *constrained* treewidth of the network. A more recent algorithm, proposed by Park and Darwiche, is exponential only in the treewidth and has been shown to handle networks whose constrained treewidth is quite high. In this paper we present a new algorithm for exact MAP that is not necessarily limited in scalability even by the treewidth. This is achieved by leveraging recent advances in compilation of Bayesian networks into arithmetic circuits, which can circumvent treewidth-imposed limits by exploiting the local structure present in the network. Specifically, we implement a branch-and-bound search where the bounds are computed using linear-time operations on the compiled arithmetic circuit. On networks with local structure, we observe orders-of-magnitude improvements over the algorithm of Park and Darwiche. In particular, we are able to efficiently solve many problems where the latter algorithm runs out of memory because of high treewidth.

Introduction

The MAP (maximum a posteriori hypothesis) problem in Bayesian networks is to find the most likely states of a set of variables (which we call the MAP variables) given partial evidence on the complement of that set. In a diagnostic setting, for example, one may be interested in knowing the most likely values of the variables modeling the health of a system, after observing a certain set of symptoms.

MAP appears to be much more difficult in practice than other typical tasks in probabilistic inference, such as computing posteriors and MPE (most probable explanation). In particular, MPE is a special case of MAP where one is interested in finding the most likely states of a set of variables given *full* evidence on the complement of that set. MPE is generally easier than MAP for standard structure-based inference methods. In variable elimination (Zhang & Poole 1996; Dechter 1996), for example, one can use any elimination order to solve the former, but can only choose among orders that put the MAP variables last to solve the latter. Consequently, MPE can be solved in time and space exponential in the treewidth of the network, while the corresponding algorithm for MAP requires time and space exponential in the *constrained* treewidth, which can be significantly higher. The same gap exists for other structure-based methods as well, such as jointree algorithms (Shenoy & Shafer 1986; Jensen, Lauritzen, & Olesen 1990).

A recent algorithm proposed in (Park & Darwiche 2003) represents a significant advance of the state of the art in solving MAP exactly. Instead of directly computing a MAP solution, it runs a depth-first search in the space of all instantiations of the MAP variables to find one with the highest probability. The key is that the search can be very effectively pruned using upper bounds that can be computed by a standard jointree algorithm (Shenoy & Shafer 1986), which is exponential only in the treewidth, *not* the constrained treewidth. The overall space requirements are therefore exponential in the treewidth only, allowing the algorithm to scale up to problems where standard methods fail because the constrained treewidth is too high (e.g., over 40).

In this paper we present a new algorithm for solving MAP exactly that is not necessarily limited in scalability even by the (unconstrained) treewidth. This is achieved by leveraging the latest advances in compilation of Bayesian networks into arithmetic circuits (Chavira & Darwiche 2005), which can circumvent treewidth-imposed limits by exploiting the local structure present in the network. Specifically, we implement a depth-first search as in (Park & Darwiche 2003), but compute upper bounds for pruning using linear-time operations on the arithmetic circuit that has been compiled from the network. The replacement of a jointree by an arithmetic circuit provides both new opportunities and challenges with respect to computational efficiency. On the one hand, many high-treewidth networks for which jointree and other structure-based inference algorithms are not feasible can be successfully compiled into arithmetic circuits of tractable size. On the other hand, one needs a new method for computing upper bounds based on circuits. Moreover, some of the techniques employed by (Park & Darwiche 2003) for

*This work has been partially supported by Air Force grant #FA9550-05-1-0075-P00002 and JPL/NASA grant #1272258. National ICT Australia is funded by the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

computing multiple upper bounds simultaneously on join-trees, in order to enable effective dynamic variable ordering, fail to carry over to arithmetic circuits, calling for new innovations in this regard. We address all of these issues in this paper, and provide empirical evidence of our ability to solve problems beyond the reach of previous methods.

In what follows, we review some background and previous work on MAP; describe the compilation of Bayesian networks into arithmetic circuits; present our new algorithm for finding exact solutions to MAP; present and discuss experimental results; and finally present our conclusions.

Background and Previous Work

We start with some notation and a formal definition of MAP. Given a Bayesian network that induces a joint probability distribution $\Pr$, let the network variables be partitioned into three sets: $\mathbf{E}$, $\mathbf{S}$, and $\mathbf{M}$ (we refer to $\mathbf{M}$ as the MAP variables). Given some evidence $\mathbf{e}$, which is an instantiation of variables $\mathbf{E}$, the MAP problem MAP($\mathbf{M}$, $\mathbf{e}$) is to find an instantiation $\mathbf{m}$ of variables $\mathbf{M}$ that maximizes $\Pr(\mathbf{m}, \mathbf{e})$ (or $\Pr(\mathbf{m}|\mathbf{e})$, equivalently). Note that the $\mathbf{S}$ variables here are those whose values we neither know nor care about.

Let $\mathbf{\Phi}$ denote the set of CPTs (conditional probability tables) of the network, and for each CPT $\phi \in \mathbf{\Phi}$, let $\phi_{\mathbf{e}}$ denote its restriction under evidence $\mathbf{e}$. $\Pr(\mathbf{m}, \mathbf{e})$ for all $\mathbf{m}$ is then given by the following potential ψ over variables $\mathbf{M}$:

$$\psi = \sum_{\mathbf{S}} \prod_{\phi \in \mathbf{\Phi}} \phi_{\mathbf{e}}. \quad (1)$$

Hence the probability of a MAP solution is given by:

$$\psi^* = \max_{\mathbf{M}} \sum_{\mathbf{S}} \prod_{\phi \in \mathbf{\Phi}} \phi_{\mathbf{e}}. \quad (2)$$

Note that when $\mathbf{S}$ is empty, we have an MPE problem, as a special case of MAP, where we only have maximizations; in the general case, we have both maximizations and summations, which cannot be swapped or mixed. Consequently, to solve MPE we can use any elimination order, the ones with lower widths in particular, but to solve MAP we can use only those orders that put the MAP variables last. The complexity of solving MAP by variable elimination is thus exponential in the *constrained* treewidth of the network, that is, the minimum width among all elimination orders satisfying the constraint that the MAP variables come last.

One way to avoid the complexity of MAP is to use an approximate algorithm. For example, one can solve the MPE problem for $\mathbf{M} \cup \mathbf{S}$ and project the solution on $\mathbf{M}$, or assemble the most likely state of each individual variable in $\mathbf{M}$ into an approximate MAP solution. Other methods include a genetic algorithm (de Campos, Gámez, & Moral 1999), hill climbing and taboo search (Park & Darwiche 2001), and simulated annealing (Yuan, Lu, & Druzdzel 2004).

For finding exact solutions to MAP, the most recent advance has been the depth-first branch-and-bound search algorithm of (Park & Darwiche 2003). As discussed earlier, the significance of this algorithm lies in that it expands the range of MAP problems for which exact solutions can be computed. Specifically, it allows problems to be solved where the treewidth is manageable but the constrained treewidth is too high for structure-based methods.

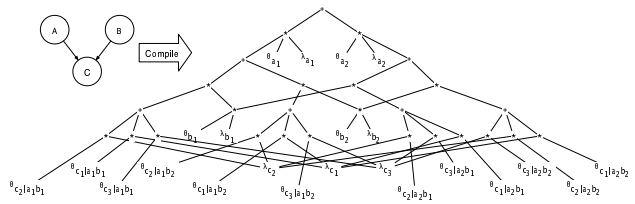

Figure 1: A Bayesian network and a corresponding AC.

The key observation underlying this search algorithm is that if we commute and mix the maximizations and summations in Equation 2, we will obtain a value that, although not the exact probability of a MAP solution, cannot be less than it. Since these values can be computed without any constraint on the elimination order, the space complexity of the algorithm drops down to exponential in the treewidth (we note that the search itself, being depth-first, uses only linear space). Moreover, with the help of several optimizations, these upper bounds have been shown to be very effective in pruning the search, allowing it to complete in reasonable time for many otherwise challenging problems.

In the present paper we aim to further expand the range of MAP problems accessible to exact methods. In particular, we aim to solve problems whose treewidth is too high for existing methods including that of (Park & Darwiche 2003), but which have local structure that allows the networks to be compiled into an arithmetic circuit of tractable size (Chavira & Darwiche 2005). We briefly review the compilation process next, followed by our proposed new algorithm.

Compiling Networks into Arithmetic Circuits

The notion of using arithmetic circuits (ACs) to perform probabilistic inference was introduced in (Darwiche 2003). With each Bayesian network, we associate a corresponding multi-linear function (MLF) that computes the probability of evidence. For example, the network in Figure 1, in which all variables are binary, induces the following MLF:

$$\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\theta_{a_1}\theta_{b_1}\theta_{c_1|a_1,b_1} + \lambda_{a_1}\lambda_{b_1}\lambda_{c_2}\theta_{a_1}\theta_{b_1}\theta_{c_2|a_1,b_1} + \ldots + \lambda_{a_2}\lambda_{b_2}\lambda_{c_2}\theta_{a_2}\theta_{b_2}\theta_{c_2|a_2,b_2} + \lambda_{a_2}\lambda_{b_2}\lambda_{c_3}\theta_{a_2}\theta_{b_2}\theta_{c_3|a_2,b_2}$$

The terms in the MLF are in one-to-one correspondence with the rows of the network's joint distribution. Assume that all *indicator variables* λ_x have value 1. Each term will then be a product of *parameter variables* $\theta_{x|\mathbf{u}}$ which evaluates to the probability of the corresponding row from the joint. The MLF will add all probabilities from the joint, for a sum of 1.0. To compute the probability of evidence, we need a way to exclude the certain terms from the sum. This removal of terms is accomplished by carefully setting certain indicators to 0 instead of 1, according to the evidence.

The fact that a network's MLF computes the probability of evidence is interesting, but the network MLF has exponential size. However, if we can factor the MLF into some-

thing small enough to fit within memory, then we can compute $\Pr(\mathbf{e})$ in time that is linear in the size of the factorization. The factorization will take the form of an AC, which is a rooted DAG (directed acyclic graph), where an internal node represents the sum or product of its children, and a leaf represents a constant or variable. In this context, those variables will be indicator and parameter variables. An example AC is depicted in Figure 1. We refer to this process of producing an AC from a network as *compiling* the network.

Once we have an AC for a network, we can compute $\Pr(\mathbf{e})$ by assigning appropriate values to leaves and then computing a value for each internal node in bottom-up fashion. The value for the root is then the answer to the query. A main point is that this process may then be repeated for as many probability of evidence queries as desired. Because computing $\Pr(\mathbf{e})$ is linear in the size of the AC, if we are able to generate an AC that is sufficiently small, computing answers to many $\Pr(\mathbf{e})$ queries will be extremely efficient.

For compiling networks into an ACs, we use the publicly available ACE system (*http://reasoning.cs.ucla.edu/ace*). ACE works by encoding the MLF into a propositional theory, compiling the theory into a logical form called d-DNNF, and then extracting the AC from the d-DNNF (Chavira & Darwiche 2005). There are several advantages to the approach. The most important of these is the ability to capture and effectively utilize local structure in the form of determinism and context-specific independence (CSI) (Boutilier *et al.* 1996). This ability is the key feature that allows some networks to be compiled in spite of large treewidth.

ACEMAP: A New Algorithm for Exact MAP

Given an arithmetic circuit C for the Bayesian network in question and a MAP query MAP($\mathbf{M}, \mathbf{e}$), the top-level procedure of our new algorithm, called ACEMAP, is a depth-first branch-and-bound search in the space of all instantiations of $\mathbf{M}$, as shown in Algorithm 1. After calling ACEMAP($C|_{\mathbf{e}}, \mathbf{M}, \{\}$), we will find a MAP solution stored in the global variable $\mathbf{m}$ and its probability stored in the global variable lb. Note that we write $C|_{\mathbf{e}}$ to denote the incorporation of evidence $\mathbf{e}$ into the arithmetic circuit C (by setting appropriate leaves of C to constants).

The depth-first search part of the algorithm works as follows: At each search node, it selects a MAP variable X from $\mathbf{M}$ (Line 2), and recursively searches each branch that is created by setting X to each of of its values (Lines 3–5). When all MAP variables have been set (Line 6), the probability of the instantiation is computed (Line 7), and compared with the current best probability (Line 8). In case the former wins, the latter is updated (Line 9) and the instantiation is stored as the current best (Line 10).

The key to making ACEMAP efficient is the pruning method employed on Line 4, whereby the algorithm will skip branches without sacrificing optimality. This is achieved by means of a function call $eval(C|_{X=x}, \mathbf{M}\backslash\{X\})$, to be defined later, which is guaranteed to return an upper bound on the probability of any instantiation of the global MAP variables that completes the current partial instantiation (which is $\mathbf{path}$). Unless this upper bound is greater than the current best probability (lb), the branch cannot contain an improvement over the current best instantiation ($\mathbf{m}$) and can therefore be pruned.

Algorithm 1 ACEMAP($C, \mathbf{M}, \mathbf{path}$)

global variables: lb (lower bound on probability of MAP solution); $\mathbf{m}$ (best instantiation of MAP variables so far). **return value**: none. **parameters** to top-level call: C (with evidence incorporated); $\mathbf{M}$ (set of MAP variables); $\mathbf{path}$ (empty set). **precondition** for top-level call: $lb < 0$. **postcondition** for top-level call: $\mathbf{m}$ is a MAP solution; $lb = \Pr(\mathbf{m})$.

```
1: if M is not empty then
2:    select variable X ∈ M
3:    for each value x of variable X do
4:       if eval(C|_{X=x}, M\{X}) > lb then
5:          ACEMAP(C|_{X=x}, M\{X}, path ∪ {X = x})
6: else
7:    p = eval(C, ∅)
8:    if p > lb then
9:       lb = p
10:      m = path
```

Algorithm 2 $eval(C, \mathbf{M})$

parameters: C is the root of the AC with original evidence and partial MAP instantiation incorporated. $\mathbf{M}$ is the set of uninstantiated MAP variables.

$$
\begin{cases}
k, & \text{if } C \text{ is leaf node representing constant } k \\
1, & \text{if } C \text{ is a leaf node representing a variable} \\
\prod_i eval(C_i, \mathbf{M}), & \text{if } C = \prod_i C_i; \\
\max_i eval(C_i, \mathbf{M}), & \text{if } C = \sum_i C_i \text{ and } C \text{ is} \\
 & \text{associated with a MAP variable} \\
\sum_i eval(C_i, \mathbf{M}), & \text{if } C = \sum_i C_i \text{ and } C \text{ is } \textbf{not} \\
 & \text{associated with a MAP variable.}
\end{cases}
$$

Computing Upper Bounds Using ACs

We now describe the key component of this algorithm, the $eval(C|_{\mathbf{e}}, \mathbf{M})$ function given in Algorithm 2, which serves two purposes:

- When $\mathbf{M}$ is nonempty, computes an upper bound on $\Pr(\mathbf{e}, \mathbf{m}')$, where $\mathbf{e}$ is the evidence asserted into the AC so far (including original evidence and partial MAP instantiation) and $\mathbf{m}'$ is the best completion of the MAP instantiation. This enables pruning (Line 4 of Algorithm 1).
- When $\mathbf{M}$ is empty, computes the probability of the evidence that has been set in the AC, which is effectively the probability of the original evidence and the instantiation of the MAP variables at a leaf of the search tree (Line 7).

Algorithm 2 performs standard circuit evaluation when the set of MAP variables $\mathbf{M}$ is empty. It is known that the circuit value in this case is simply the probability of evidence on which the circuit is conditioned (Darwiche 2003), which allows us to efficiently compute the probability of an instantiation of the MAP variables (Line 7 of Algorithm 1). When the set of MAP variables is not empty, Algorithm 2

will perform a maximization (instead of addition) on some of the addition nodes which correspond to MAP variables. Therefore, Algorithm 2 requires a labeled arithmetic circuit in the sense that some of its addition nodes are associated with MAP variables. Indeed, the arithmetic circuits generated by ACE have the following property: every addition node is of the form $\alpha + \beta$, where the two sub-circuits α and β correspond to different values of some variable X. Moreover, ACE can provide information on the identity of variables associated with addition nodes and we use this information to label nodes as required by Algorithm 2. Here is the guarantee provided by this algorithm.

Theorem 1 (Upper Bound) *Algorithm 2 computes an upper bound on the probability of the best instantiation of variables* $\mathbf{M}$*:* $eval(C|_{\mathbf{e}}, \mathbf{M}) \geq \max_{\mathbf{m}} \Pr(\mathbf{m}, \mathbf{e})$.

Note that the AC passed to $eval$ is not the initial AC for the Bayesian network, but that restricted under the initial evidence $\mathbf{e}$ and any partial instantiation of the initial MAP variables that has been made prior to the call to $eval$. The value computed by $eval$ is hence effectively an upper bound on the probability of the best instantiation of the initial MAP variables that extends the current partial instantiation, serving the purposes of pruning nicely.

While we omit a formal proof, the intuition behind Theorem 1 is as follows. Suppose that we replace every addition node in an arithmetic circuit by a maximization node instead. It is well known that the value of the circuit in this case will represent the probability of the most likely variable instantiation. That is, it will represent the value of MAP assuming that all network variables are MAP variables. Suppose now that only a subset of network variables are MAP variables. The algorithm will also compute the probability of MAP in this case, assuming that no addition node which is associated with a MAP variable can be a descendent of an addition node that is not associated with a MAP variable. One can indeed construct circuits that satisfy this property, but their construction can no longer be guaranteed to be exponential only in network treewidth. By relaxing this requirement, one can construct circuits bounded in the worst case by treewidth, while allowing some maximizations to be performed too early, leading to a final value that can be greater (but never less) than the exact MAP value.

Static vs. Dynamic Variable Ordering

It is well known that variable ordering (Line 2 of Algorithm 1) can have a great effect on the efficiency of a search algorithm. We experimented with both static and dynamic ordering heuristics, with a view to increasing the tightness of the upper bounds and hence the amount of pruning.

The static ordering heuristic we found to have the best overall performance works as follows: For each MAP variable X and each of its values x, we locate the leaf λ_x of the AC that corresponds to $X = x$ (which is unique if it exists in the AC at all), and count the AC nodes that are ancestors of this leaf—we will refer to this set of nodes as the *cone* of λ_x. Each variable is thus associated with multiple cones corresponding to its different values. We then select variables in decreasing order of their average cone size.

The intuition is that a larger cone represents a greater contribution to the looseness of the bound when the corresponding MAP variable is left uninstantiated. In this case, maximizations over the variable inflate the bound through more nodes up the circuit. Variables with larger cones should therefore be set earlier to help produce tighter bounds for subsequent search.

Our investigation into dynamic ordering revealed an interesting trade-off between the effectiveness of a variable ordering heuristic on tightening the bounds and the cost of its computation. In particular, we found that the amount of pruning could be greatly increased and the number of search nodes greatly reduced if the variable was chosen dynamically, on Line 2 of Algorithm 1, in the following way: For each variable X, compute $eval(AC|_{X=x}, \mathbf{M}\backslash\{X\})$ for each value x of X and call the highest result the *score* of X; select a variable with the *lowest* score.

The intuition here is that the score of a variable is an upper bound on the best probability that can be achieved after setting that variable, and hence picking a variable that has the lowest score represents an attempt to minimize (tighten) this upper bound and hence increase the amount of subsequent pruning. Our experiments indicate that this dynamic ordering heuristic almost always leads to significantly fewer search nodes than the static ordering heuristic described above. However, the actual search time increases in most cases owing to the large number of calls to $eval()$ required.

We note that with a jointree algorithm, as has been employed in (Park & Darwiche 2003), all these calls to $eval()$ at each search node can be replaced by a single run of jointree propagation, which makes dynamic ordering a very good choice for the MAP algorithm of (Park & Darwiche 2003). We view our current inability to achieve the same efficiency in this regard a potential opportunity to further enhance our AC-based approach in future work. Our current experimental results are based on using the static ordering.

Initialization

We initialize the search with an "educated guess" of a good approximate MAP solution, as follows: For each MAP variable in the order determined by the static ordering heuristic described above, we choose the value that has the highest partial derivative (for space constraints we refer the reader to (Darwiche 2003) for details on the semantics of AC differentiation); the variable is then set to the chosen value and the AC differentiated again so that all the partial derivatives are updated before the next variable is processed.

It is known that the partial derivative of the AC with respect to a leaf λ_x of the AC gives the updated probability of evidence when X is set to x (Darwiche 2003). Our heuristic for an initial approximate solution can therefore be viewed as attempting to greedily maximize its probability. Note also that differentiation of an AC is a linear-time operation and hence our initialization heuristic is generally very efficient.

Experimental Results

Table 1 shows the networks with which we experimented. The blockmap, mastermind, and student networks represent

Table 1: Performance of ACEMAP vs. PD03 ("–" indicates failure to solve all instances using available memory).

Network	MAP Vars	Instances	Approximate		PD03	Compilation into Arithmetic Circuit				ACEMAP
			TW	Cons. TW	Time	Vars	Nodes	Edges	Time	Time
blockmap-5-1	26	10	19	36	186.42	1411	2479	4793	0.33	0.01
blockmap-5-2	27	10	19	37	306.25	1738	3251	11789	0.48	0.12
blockmap-5-3	28	10	22	38	357.18	2055	4550	22209	0.64	0.14
blockmap-10-1	100	10	51	122	–	11326	20214	69078	14.22	0.34
blockmap-10-2	100	10	48	122	–	12558	23183	264933	18.05	0.52
blockmap-10-3	100	10	50	122	–	13786	30115	586379	23.72	0.84
mastermind-3-8-3	27	10	21	33	2088.19	2494	24207	461060	1.70	44.45
students-3-2	14	10	23	29	212.79	1004	5925	53352	0.28	0.09
students-3-6	14	10	40	45	–	2260	387883	1596976	10.00	4.47
students-3-12	14	10	53	55	–	4144	10494185	54725407	368.25	128.01
grid-50-10-1	99	10	15	15	99.32	600	59298	309948	0.63	54.52
grid-50-10-2	99	10	15	15	80.91	636	49660	243306	0.53	10.98
grid-50-10-3	99	10	15	15	72.80	592	115465	562329	1.17	29.42
grid-50-12-1	100	10	18	23	1419.62	756	54295	274787	0.61	305.33
grid-50-12-2	100	10	18	35	2098.53	808	275020	1614714	3.41	72143.73
grid-50-12-3	100	10	18	27	1511.30	836	152067	733874	1.67	18499.28
grid-75-10-1	99	10	15	15	34.34	364	2052	6684	0.08	0.19
grid-75-10-2	99	10	15	14	41.47	400	2778	9371	0.09	0.11
grid-75-10-3	99	10	15	15	56.44	396	5207	20705	0.11	1.16
grid-75-12-1	100	10	18	29	609.28	560	13486	60889	0.23	11.63
grid-75-12-2	100	10	18	33	255.91	548	9190	44764	0.17	1.00
grid-75-12-3	100	10	18	31	595.36	600	15673	67132	0.30	3.33
grid-75-14-1	100	10	21	79	2197.01	652	6396	31843	0.19	1.48
grid-75-14-2	100	10	21	64	6317.88	708	21139	107317	0.36	25.70
grid-75-14-3	100	10	21	84	–	732	31493	169774	0.50	33.11
grid-75-16-1	100	10	25	99	–	1000	228585	1155910	3.06	16811.04
grid-75-16-2	100	10	24	102	–	1056	79401	355094	1.16	25437.23
grid-75-16-3	100	10	25	101	–	992	120859	592151	1.70	1534.51
grid-90-10-1	99	10	15	15	26.69	300	1794	6387	0.06	0.09
grid-90-10-2	99	10	15	15	25.67	276	788	2241	0.05	0.05
grid-90-10-3	99	10	15	15	17.70	300	1060	4107	0.05	0.06
grid-90-12-1	100	10	18	29	124.95	404	1666	5569	0.08	0.09
grid-90-12-2	100	10	18	34	168.02	408	1116	2377	0.05	0.05
grid-90-12-3	100	10	18	29	203.22	436	1437	3830	0.08	0.08
grid-90-14-1	100	10	22	73	1666.81	520	2080	7318	0.11	0.22
grid-90-14-2	100	10	21	88	2701.21	540	1885	6260	0.09	0.34
grid-90-14-3	100	10	22	92	1422.71	520	1432	4186	0.20	0.13
grid-90-16-1	100	10	25	101	–	736	10022	54785	0.39	5.08
grid-90-16-2	100	10	25	102	–	620	1985	8517	0.12	0.16
grid-90-16-3	100	10	25	103	–	728	2415	7600	0.16	0.16
grid-90-18-1	100	10	28	107	–	912	7612	49639	0.31	3.69
grid-90-18-2	100	10	29	107	–	896	7824	41192	0.30	4.06
grid-90-18-3	100	10	28	106	–	904	5664	24771	0.25	9.31
grid-90-22-1	100	10	35	113	–	1304	16397	59780	0.72	23.91
grid-90-22-2	100	10	36	113	–	1324	21997	111851	0.92	138.21
grid-90-22-3	100	10	36	115	–	1320	6544	23426	0.36	1.63
grid-90-26-1	100	10	43	119	–	1832	24830	103103	1.44	108.08
grid-90-26-2	100	10	44	120	–	1918	139143	628461	7.06	7462.13
grid-90-26-3	100	10	42	120	–	1828	44109	218879	1.72	368.22
alarm	12	10	7	16	5.84	264	3026	8551	0.06	0.19
hailfinder	17	10	12	36	41.67	1030	18513	109385	0.38	681.55
pathfinder	1	10	15	15	22.97	2679	33082	207207	7.83	1.00
pigs	20	10	16	33	12.90	2211	1154490	6136011	30.69	7905.22
tcc4f.obfuscated	36	10	10	10	7.67	1290	17499	192859	0.55	0.55
water	8	10	21	21	324.132	3659	108164	1242839	3.47	4.94

ground instances of relational Bayesian networks (Jaeger 1997). These networks have many binary variables, small CPTs, large amounts of determinism, and large treewidth. The second set of networks, grid-x-y-z, are similar in that they all arrange their nodes in a grid-like pattern. Again, each has a large degree of determinism. They were used in (Sang, Beame, & Kautz 2005) to demonstrate probabilistic inference using weighted model counting, which shares much in common with the compilation approach used here. The last set consists of benchmark networks that have long been used to evaluate inference algorithms. These networks do not necessarily exhibit a large amount of determinism, but the lesser amounts of determinism that exist, along with any CSI, will be utilized by ACE.

For each network other than the grid networks, we generated ten problems randomly based on the method in (Yuan, Lu, & Druzdzel 2004). For each problem, we randomly selected n MAP variables from the roots or all roots if their number was smaller than n. We then randomly selected n evidence variables from the leaves or all leaves if their number was smaller than n. Finally, we generated evidence randomly, making sure that the probability of evidence was nonzero. For pigs, $n = 20$; otherwise, $n = 100$. The grid networks each have a single root. As a result, we generated the ten problems for these networks based on the method in (Park & Darwiche 2003). For each problem, we made all leaves into evidence variables and randomly chose 100 MAP variables from non-leaves.

For each network, we compiled the network into an AC and then used ACEMAP to solve the generated problems. For comparison, we also solved the problems using the exact MAP algorithm of (Park & Darwiche 2003), which we refer to as PD03 (as implemented in SAMIAM available at *http://reasoning.cs.ucla.edu/samiam*). All experiments were conducted on a 3.2GHz Pentium 4 with 1GB of RAM running Windows XP Professional.

Table 1 shows a number of results. The first observation we make relates to the treewidths and constrained treewidths of these networks. We computed an approximation of these metrics by pruning the network according to the evidence and query, applying a minfill heuristic, and reporting the largest value obtained across all problems. PD03 is able to solve many problems where the constrained treewidth is high, but runs out of memory when treewidth is high. Despite high treewidth, ACEMAP was able to solve all of these problems. Our second observation relates to the compile times for these networks. The vast majority of networks compiled in a matter of seconds or less, and compile times are very small compared to the time for executing PD03. One prominent exception is pigs. Recall that compilation needs to be performed only once per network, no matter how many MAP problems need to be solved. Compilation sizes are measured in number of AC edges. Most of these sizes are very small compared to what will fit within available RAM, and the largest of them correspond to networks on which PD03 failed. Finally, we report total search times across all ten problems for both algorithms. ACEMAP outperformed PD03 on 51 out of the 55 networks with which we experimented, often by orders of magnitude. For 21 of these networks, PD03 ran out of memory and failed to solve the problems. We believe that ACEMAP's inferior performance on the four networks, all of which have relatively low treewidth, is due to the dynamic variable ordering used by PD03 being particularly superior to ACEMAP's static ordering for these networks.

Conclusions

We presented a new algorithm for finding exact solutions to MAP, which significantly expands the range of MAP problems that can be solved exactly. Moreover, the proposed algorithm is more efficient on problems that can already be solved by previous methods. The algorithm is based on static variable ordering as the implementation of dynamic variable ordering proved too expensive in the context of ACs. We expect the development of efficient dynamic variable ordering heuristics to lead to further significant improvements in our ability to solve MAP problems exactly.

References

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in bayesian networks. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, 115–123.

Chavira, M., and Darwiche, A. 2005. Compiling Bayesian networks with local structure. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 1306–1312.

Darwiche, A. 2003. A differential approach to inference in Bayesian networks. *Journal of the ACM* 50(3):280–305.

de Campos, L. M.; Gámez, J. A.; and Moral, S. 1999. Partial abductive inference in Bayesian belief networks using a genetic algorithm. *Pattern Recognition Letters* 20(11-13):1211–1217.

Dechter, R. 1996. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, 211–219.

Jaeger, M. 1997. Relational Bayesian networks. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI)*, 266–273.

Jensen, F. V.; Lauritzen, S.; and Olesen, K. 1990. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly* 4:269–282.

Park, J. D., and Darwiche, A. 2001. Approximating MAP using local search. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI)*, 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)*, 459–468.

Sang, T.; Beame, P.; and Kautz, H. A. 2005. Performing Bayesian inference by weighted model counting. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 475–482.

Shenoy, P. P., and Shafer, G. 1986. Propagating belief functions with local computations. *IEEE Expert* 1(3):43–52.

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

Zhang, N. L., and Poole, D. 1996. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research* 5:301–328.