Join Tree Propagation Utilizing Both Arc Reversal and Variable Elimination

C. J. Butz and **K. Konkel**
Department of Computer Science
University of Regina
Regina, SK, S4S 0A2
{butz,konkel1k}@cs.uregina.ca

P. Lingras
Department of Mathematics and Computing Science
Saint Mary's University
Halifax, NS, B3H 3C3
pawan.lingras@stmarys.ca

Abstract

In this paper, we put forth the first join tree propagation algorithm that selectively applies either *arc reversal* (AR) or *variable elimination* (VE) to build the propagated messages. Our approach utilizes a recent method for identifying the propagated join tree messages à priori. When it is determined that precisely one message is to be constructed at a join tree node, VE is utilized to build this distribution; otherwise, AR is applied as it is better suited to construct multiple distributions passed between neighboring join tree nodes. Experimental results, involving evidence processing in seven real-world and one benchmark Bayesian network, empirically demonstrate that selectively applying VE and AR is faster than applying one of these methods exclusively on the entire network.

Introduction

Bayesian networks (Pearl 1988) provide a rigorous foundation for uncertainty management by combining probability theory and graph theory, and have been successfully applied in practice to a wide variety of problem domains. A Bayesian network consists of a *directed acyclic graph* (Pearl 1988) and a corresponding set of *conditional probability tables* (Shafer 1996). The vertices in the directed acyclic graph represent the random variables in the real-world problem, while the edges in the graph represent probabilistic dependencies amongst the variables. More specifically, the *probabilistic conditional independencies* encoded in the directed acyclic graph indicate that the product of the conditional probability tables is a joint probability distribution. Therefore, Bayesian networks continue to provide a robust framework for designing expert systems (Kjaerulff and Madsen 2008). Although Cooper (1990) has shown that the complexity of exact inference in discrete Bayesian networks is NP-hard, various approaches have been developed that seem to work quite well in practice. All of these methods center around eliminating variables from the networks to produce posterior probability distributions and can be broadly classified into two categories.

The first category of Bayesian network inference is *direct computation*. The two leading direct computation algorithms are *variable elimination* (VE) (Pearl 1988; Hájek, Havránek, and Jiroušek 1992; Shafer 1996) and *arc reversal* (AR) (Olmsted 1983; Shachter 1986). VE removes a variable by multiplying together all of the distributions involving the variable and then summing the variable out of the obtained product. AR removes a variable by reversing the edges between the variable and its children giving a modified directed acyclic graph and then building the conditional probability tables corresponding to the modified graph. The second category is *join tree propagation*, which Shafer (1996) emphasizes is central to the theory and practice of probabilistic expert systems. Join tree propagation first builds a secondary network, called a join tree, from the directed acyclic graph of the Bayesian network and then performs inference by propagating probabilities in the join tree.

Recently, Madsen (2004) introduced *Lazy-AR* and empirically demonstrated a computational advantage over its predecessor, *Lazy-VE* (Madsen and Jensen 1999). The only difference between Lazy-AR and Lazy-VE is that the former utilizes AR when eliminating variables during join tree propagation, whereas the latter uses VE. Both Lazy-AR and Lazy-VE, however, are too rigid to exploit various kinds of structures found within real-world Bayesian networks. We explicitly demonstrate that during propagation in one join tree, AR can be the best choice for eliminating variables at one join tree node, yet VE is the most suitable method for eliminating variables at another join tree node. Neither Lazy-AR nor Lazy-VE are flexible enough to take advantage of these situations as they both apply a single technique for eliminating variables throughout the entire join tree.

In this paper, we suggest selectively applying either AR or VE to build the messages propagated in a join tree. A key difference between our system, called *DataBayes*, and all other join tree propagation algorithms are two analytical preprocessing steps. The first step uses the method in (Butz, Yao, and Hua 2008) to determine à priori those messages that will be propagated in the join tree. The second step uses this information as follows. When it is known that a join tree node will construct a single distribution to be sent to a neighboring node, VE is applied to build this message. On the other hand, AR is applied when a node is to pass more than one distribution to a neighboring node. The efficiency improvement offered by DataBayes is shown through empirical evaluations involving seven real-world and one benchmark Bayesian network. As is usually done, in each network, inference is performed with varying amounts of evidence, namely, 0%, 9% and 18%. DataBayes finished inference faster than LAZY-AR in all cases without exception. Since Lazy-AR tends not to be slower than Lazy-VE (Madsen 2004), our results empirically demonstrate that selectively applying VE and AR is faster than applying one of these methods exclusively.

This paper is organized as follows. Background information is first reviewed. We then propose a more sophisticated approach to join tree propagation. Experimental results are subsequently provided. The manuscript ends with our conclusions.

Definitions

Let $U = \{v_1, v_2, \dots, v_n\}$ denote a finite set of discrete random variables. Each variable v_i is associated with a finite domain, denoted $dom(v_i)$, representing the values v_i can take on. For a subset $X \subseteq U$, we write $dom(X)$ for the Cartesian product of the domains of the individual variables in X. Each element $x \in dom(X)$ is called a *configuration* of X. A *potential* (Hájek, Havránek, and Jiroušek 1992) on $dom(X)$ is a function ϕ on $dom(X)$ such that $\phi(x) \geq 0$, for each configuration $x \in dom(X)$, and at least one $\phi(x)$ is positive. For brevity, we refer to a potential as a probability distribution on X rather than $dom(X)$, and we call X, not $dom(X)$, its domain (Shafer 1996). A *joint probability distribution* (Shafer 1996) on U, denoted $p(U)$, is a potential on U that sums to one.

Let X and Y be two disjoint subsets of U. A *conditional probability table* (Shafer 1996) for Y given X, denoted $p(Y|X)$, is a nonnegative function on $X \cup Y$, satisfying the following condition: for each configuration $x \in dom(X)$, $\sum_{y \in dom(Y)} p(Y = y|X = x) = 1.0$.

The *heading* of a probability distribution is the label appearing above the probability column.

A discrete *Bayesian network* on U is a pair (D, C). D is a *directed acyclic graph* on U. C is a set of *conditional probability tables* defined as: for each variable $v_i \in D$, there is a conditional probability table for v_i given its set of parents P_i in D.

Note that the set U of vertices in D represent the random variables in the real-world problem domain, and the edges in D represent probabilistic dependencies amongst the variables. We shall use vertex and variable interchangeably, as well as the terms Bayesian network and directed acyclic graph, if no confusion arises.

The *family* of a variable v_i, denoted as F_i, in a directed acyclic graph is $\{v_i\} \cup P_i$. Let $An(X \cup Y)$ be the *ancestral set* of $X \cup Y$, where X and Y are subsets of U, i.e., the set of variables in $X \cup Y$ and the ancestors of those variables. A numbering $\prec$ of the variables in a directed acyclic graph is called *ancestral* (Castillo, Gutierrez, and Hadi 1997), if the number corresponding to any variable v_i is lower than the number corresponding to each of its children v_j, denoted $v_i \prec v_j$.

Once a Bayesian network has been obtained, the next task is to answer queries such as $p(E = e)$ or $p(X|E = e)$, where E and X are disjoint sets of variables in the Bayesian network, and e are the observed values of E. One common approach to performing Bayesian network inference is to build a secondary structure, called a *join tree*, and then conduct Bayesian network inference via join tree propagation.

A *join tree* is a tree with sets of variables as nodes, with the property that any variable in two nodes is also in any node on the path between the two. By definition of join tree, one node can be a subset of another. The *separator* between any two neighboring nodes N_i and N_j is $N_i \cap N_j$. We refer the reader to (Kjaerulff and Madsen 2008) for a discussion on building Bayesian networks and join trees.

For example, one possible join tree for a real-world Bayesian network, called *Water*, is depicted in Figure 1. The join tree nodes are $A = \{0, 4, 5, 20, 24\}$, $B = \{4, 16, 20, 24, 25\}$, $C = \{5, 10, 16, 17, 21, 25, 26, 31\}$, $D = \{5, 10, 17, 21, 25, 26, 31\}$, $E = \{8, 9, 10, 12, 16, 17, 26, 31\}$, $F = \{9, 10, 12, 13, 16, 17, 26\}$, $G = \{9, 10, 13, 17, 26\}$, The CPTs of the Water Bayesian network are assigned to the join tree nodes as shown. For instance, $\{p(16), p(25|4, 16, 20, 24\}$ are assigned to join tree node B.

A preprocessing step (Butz, Yao, and Hua 2008) is applied to identify the headings of all messages to be passed in the join tree à priori. It must be made clear that the step is not computing the actual probability distributions with rows and probability values; instead, it is only computing the headings appearing above the probability columns (the schema of each message).

For example, given the assignment of CPTs to the Water join tree nodes in Figure 1, it can be determined à priori that node A will pass messages $\{p(4), p(5|4, 20, 21, 24), p(20), p(21|20, 24), p(24)\}$ to its neighboring node B when join tree propagation is performed.

Bayesian Network Inference With DataBayes

In this section, we propose a new join tree propagation algorithm for Bayesian network inference. This algorithm has been implemented in our probabilistic reasoning system, called DataBayes.

Following the probabilistic inference competition held at UAI 2006, our objective is to answer a query $p(E = e)$ posed to a Bayesian network, where E is a randomly selected set of variables taking value e. As variables irrelevant to the query are pruned from the Bayesian network as part of query optimization, the constructed join tree only contains the variables required to compute $p(E = e)$. Lastly, our method only involves the inward phase of join tree propagation.

The key difference between DataBayes and all previous join tree propagation algorithms is that DataBayes applies a preprocessing step, as discussed next.

The following algorithm, called PickARorVE, determines whether DataBayes should apply AR or VE at each node in the join tree before propagation begins.

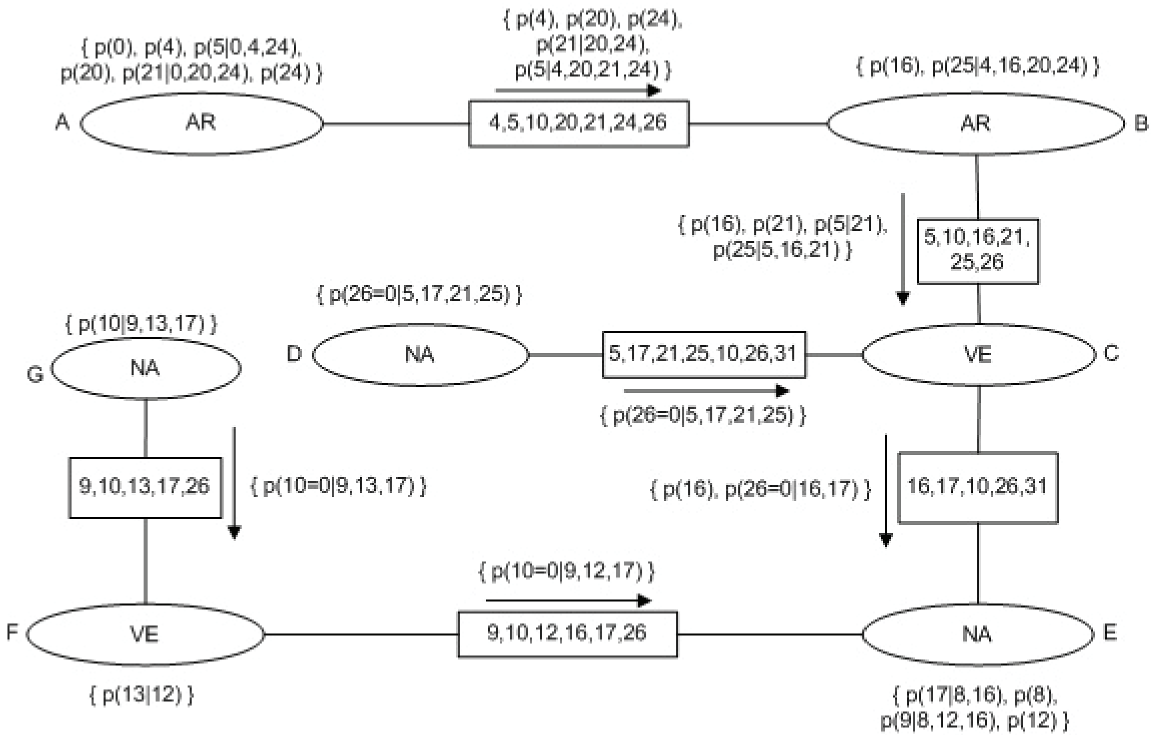

Figure 1: Given the Water join tree with identified messages, the PickARorVE algorithm labels all nodes as shown.

```
Algorithm 1 PickARorVE ($J$)
Input: a join tree $J$ with messages to be propagated identified.
Output: each join tree node marked with one of AR, VE or NA.
begin
for each join tree node $N$
        count the number $n$ of messages to be constructed at $N$.
        if $n > 1$
                Mark $N$ as AR
        else if $n == 1$
                Mark $N$ as VE
        else
                Mark $N$ as NA
end
```

We now illustrate the PickARorVE algorithm. In the Water join tree of Figure 1 with identified messages, the PickARorVE algorithm labels all nodes as shown. Consider node D. Node D does not need to build any distributions. Instead, it is simply forwarding its only distribution $p(26 = 0|5, 17, 21, 25)$ to node C. Thus, PickARorVE marks node D as NA. Now consider node A. It is forwarding to node B three distributions $p(4)$, $p(20)$ and $p(24)$, but needs to construct two distributions $p(5|4, 20, 21, 24)$ and $p(21|20, 24)$. Consequently, PickARorVE marks node A as AR. Lastly, consider node C. It is forwarding to node E the distribution $p(16)$, but needs to build the one distribution $p(26 = 0|16, 17)$. Thereby, PickARorVE marks node C as VE.

The important point is that the PickARorVE labelling indicates which technique, if any, should be applied to eliminate variables at each join tree node. As shown in Figure 1, AR will be applied at nodes A and B, while VE will be applied at nodes C and F. Nodes D and G simply forward distributions, while node E is the root.

DataBayes Applying VE

DataBayes uses VE to eliminate variables when precisely one message needs to be constructed at a join tree node. It must be emphasized that VE is not simply applied when a single distribution is passed between neighboring nodes. It is entirely possible that more than one distribution is passed from a node N to a neighbor, but only one of these distributions need to be constructed at N. For example, consider the two distributions $p(16)$ and $p(26 = 0|16, 17)$ to be passed from node C to node E in Figure 1. Observe that $p(16)$ does not need to be built at node C, since $p(16)$ was constructed at node B, passed to node C, and node C will simply forward it to node E. On the other hand, $p(26 = 0|16, 17)$ needs to be constructed at node C.

Due to lack of space, we refer the reader to the literature for a formal description of the VE algorithm and use the following example to illustrate it.

DataBayes applies VE to construct the single distribution $p(26 = 0|16, 17)$, sent from node C to node E in Figure 1, as follows. Variables $\{5, 21, 25\}$ need to be eliminated from: $p(5|21) \cdot p(16) \cdot p(21) \cdot p(25|5, 16, 21) \cdot p(26 = 0|5, 17, 21, 25)$. Note that the elimination ordering $21, 5, 25$ is computed by the DataBayes system. VE marginalizes variable 21 out from the product $p(5|21) \cdot p(21) \cdot p(25|5, 16, 21) \cdot p(26 = 0|5, 17, 21, 25)$. This process yields $p(5, 25, 26 = 0|16, 17)$. Thus, we obtain

$$\sum_{25} \sum_{5} p(5, 25, 26 = 0|16, 17) \cdot p(16). \qquad (1)$$

Variable 5 is then removed by marginalization yielding

$$\sum_{25} p(16) \cdot p(25, 26 = 0|16, 17). \qquad (2)$$

The last variable 25 is then marginalizaed away leaving

$$p(16) \cdot p(26 = 0|16, 17), \qquad (3)$$

which are the messages passed from node C to node E in Figure 1.

DataBayes Applying AR

Arc reversal (AR) (Olmsted 1983; Shachter 1986) is an algorithm that eliminates a variable by making it barren via a sequence of directed edge (arc) reversals in the directed acyclic graph of a Bayesian network prior to eliminating it. AR uses an ancestral numbering $\prec$ of the original Bayesian network to avoid creating directed cycles, thereby maintaining a directed acyclic graph structure after eliminating a variable by applying arc reversal (Madsen 2004).

We illustrate the AR algorithm with the following example. In the Water join tree of Figure 1, DataBayes applies AR to construct the distributions sent from node B to node C as follows. Variables $\{4, 20, 24\}$ must be eliminated from: $p(4) \cdot p(5|4, 20, 21, 24) \cdot p(16) \cdot p(20) \cdot p(21|20, 24) \cdot p(24) \cdot p(25|4, 16, 20, 24)$. Note the elimination ordering $4, 20, 24$ is generated by the DataBayes system. Here variable 4 is a parent of two children, namely, variables 5 and 25. With respect to variable 5, AR computes:

$$p(5|20, 21, 24) = \sum_{4} p(4) \cdot p(5|4, 20, 21, 24)$$

and

$$p(4|5, 20, 21, 24) = \frac{p(4) \cdot p(5|4, 20, 21, 24)}{p(5|20, 21, 24)}.$$

Next, with respect to the second and final child 25, AR calculates $p(25|5, 16, 20, 21, 24)$ by

$$\sum_{4} p(4|5, 20, 21, 24) \cdot p(25|4, 16, 20, 24).$$

Now that variable 4 has been eliminated, variables 20 and 24 will be eliminated from: $p(5|20, 21, 24) \cdot p(16) \cdot p(20) \cdot p(21|20, 24) \cdot p(24) \cdot p(25|16, 20, 24)$. Variable 20 is a parent of three children: $21, 5$ and 25. Similar to the case when eliminating variable 4, AR eliminates variable 20 as follows. For the first child 25,

$$p(21|24) = \sum_{20} p(20) \cdot p(21|20, 24)$$

and

$$p(20|21, 24) = \frac{p(20) \cdot p(21|20, 24)}{p(21|24)},$$

while for the second child 5

$$p(5|21, 24) = \sum_{20} p(20|21, 24) \cdot p(5|20, 21, 24)$$

and

$$p(20|5, 21, 24) = \frac{p(20|21, 24) \cdot p(5|20, 21, 24)}{p(5|21, 24)}.$$

For the last child 25,

$$p(25|16, 24) = \sum_{20} p(20|5, 21, 24) \cdot p(25|16, 20, 24).$$

Now that variable 20 is eliminated, variable 24 will be eliminated from: $p(5|21, 24) \cdot p(16) \cdot p(21|24) \cdot p(24) \cdot p(25|16, 24)$. As variable 24 has three children 21, 5 and 25, AR computes.

$$\begin{aligned}
p(21) &= \sum_{24} p(24) \cdot p(21|24), \\
p(24|21) &= \frac{p(24) \cdot p(21|24)}{p(21)}, \\
p(5|21) &= \sum_{24} p(24|21) \cdot p(5|21, 24), \\
p(24|5, 21) &= \frac{p(24|21) \cdot p(5|21, 24)}{p(5|21)}, \\
p(25|5, 16, 21) &= \sum_{24} p(24|5, 21) \cdot p(25|16, 24).
\end{aligned}$$

After variable 24 has been eliminated, the messages passed from node B to node C in Figure 1 are

$$p(5|21) \cdot p(16) \cdot p(21) \cdot p(25|5, 16, 21). \qquad (4)$$

ADVANTAGES

We first explain our choice of whether to apply VE or AR when building messages. Our preprocessing step (Butz, Yao, and Hua 2008) identifies the messages to be passed in a join tree before propagation begins. The messages we identify are precisely those that will be propagated, provided that AR is chosen as the algorithm for building messages. However, (Butz and Hua 2006) pointed out that during its execution distributions can be built that will not be passed as messages, nor will they be needed in the construction of the distributions to be passed as messages. The reason is that in the elimination of a variable v_i, AR constructs $3k - 1$ distributions and outputs k of them, where k is the number of children of v_i with respect to the distributions at the sending join tree node. On the contrary, VE is more streamlined in that it only outputs a single distribution. Therefore, if our preprocessing step indicates that a join tree node will pass a single distribution, then we select VE to build it. Otherwise, we pick AR to build the multiple distributions to be passed. The next example demonstrates that it can be wasteful to apply AR when only one distribution needs to be built.

Recall the distribution $p(26 = 0|16, 17)$ passed from node C to node E in the Water join tree of Figure 1. Variables $25, 5$ and 21 need to be eliminated at node C from $p(16) \cdot$

$p(21) \cdot p(5|21) \cdot p(25|21,5,16) \cdot p(26=0|21,5,17,25)$. The distinction to be made is whether to apply VE, as DataBayes did earlier, or to apply AR, as Lazy-AR does in the next example.

Lazy-AR applies AR to eliminate variable 21 by computing the following equations.

$$\begin{aligned} p(5) &= \sum_{21} p(21) \cdot p(5|21) \\ p(21|5) &= \frac{p(21) \cdot p(5|21)}{p(5)} \\ p(25|5,16) &= \sum_{21} p(21|5) \cdot p(25|21,5,16) \\ p(21|5,25,16) &= \frac{p(21|5) \cdot p(25|21,5,16)}{p(25|5,16)}, \end{aligned}$$

and

$$\begin{aligned} & p(26=0|5,16,17,25) \\ &= \sum_{21} p(21|5,25,16) \cdot p(26=0|21,5,17,25) \end{aligned}$$

AR eliminates variable 5 via:

$$\begin{aligned} p(25|16) &= \sum_{5} p(5) \cdot p(25|5,16) \\ p(5|25,16) &= \frac{p(5) \cdot p(25|5,16)}{p(25|16)}, \end{aligned}$$

and

$$\begin{aligned} & p(26=0|16,17,25) \\ &= \sum_{5} p(5|25,16) \cdot p(26=0|5,16,17,25). \end{aligned}$$

Finally, AR eliminates variable 25 through the following computation:

$$p(26=0|16,17) = \sum_{25} p(25|16) \cdot p(26=0|16,17,25).$$

The important point in the above example is that Lazy-AR eliminates variables 21, 5 and 25 using AR, which requires six multiplication operations, six marginalization operations and three division operations. On the contrary, DataBayes eliminates these same variables using VE, which only requires three multiplication operations and three marginalization operations.

The above discussion clearly demonstrates how Lazy-AR is unable to recognize those situations when VE should be applied for eliminating variables because it does not perform preprocessing steps like DataBayes to determine which messages need to be built and which techique for eliminating variables is most appropriate. Instead, Lazy-AR exclusively applies AR, which is more suitable for constructing multiple distributions and less so for building a single distribution. It is acknowledged, however, that VE and AR will perform exactly the same computation if the variable being eliminated only appears in two distributions at the time it is removed.

Experimental Results

In this section, we report an empirical comparison of our DataBayes join tree propagation approach and Lazy-AR. Both methods were implemented in the C++ programming language. The experiments were conducted on a 24-processor SGI Onyx2 graphics supercomputer with two processors allocated for our sole usage. The evaluation was carried out on seven real-world Bayesian networks: Alarm, Barley, CHD, Diabetes, Hailfinder, Mildew, and Water, and a benchmark Bayesian network Insurance. Table 1 describes the Bayesian network used, the number of variables in each Bayesian network, the number of rows in the CPTs of the BN, the number of nodes in each join tree, and the number of messages passed in the JT when no evidence is involved.

Bayesian network	# vars	# CPT rows	# JT nodes	# JT msgs
Alarm	37	1192	27	51
Barley	48	473406	36	123
CHD	11	60	5	6
Diabetes	413	1185160	336	1093
Hailfinder	56	9048	43	104
Insurance	27	232	18	61
Mildew	35	1167440	29	99
Water	32	17984	19	81

Table 1: Description of real-world or benchmark Bayesian networks and the constructed join trees

Table 2 reports on Bayesian inference not involving evidence processing. Running times for the PickARorVE algorithm are in microseconds, and are included in the running times for DataBayes. Running times for Lazy-AR and for DataBayes are listed in seconds. The last column shows the speed-up percentage of DataBayes over Lazy-AR and possessing an average percentage of 32%.

Bayesian network	Pick-ARorVE	Data-Bayes	Lazy-AR	Net %
Alarm	$2\mu s$	143	233	39%
Barley	$2\mu s$	3890	7409	47%
CHD	$0\mu s$	32	53	40%
Diabetes	$101\mu s$	7250	9126	21%
Hailfinder	$1\mu s$	338	444	24%
Insurance	$0\mu s$	152	210	28%
Mildew	$1\mu s$	547	670	18%
Water	$1\mu s$	148	247	40%

Table 2: The performance of Lazy-AR and DataBayes not involving evidence processing in real-world or benchmark Bayesian networks

Next, we measure the runtime of inference involving evidence. In Tables 3 and 4, approximately nine percent and eighteen percent of the variables in each Bayesian network are randomly instantiated as evidence variables, respectively. Note that once again, DataBayes is faster than

Lazy-AR in all Bayesian networks. In Table 3, the percentage of time saved ranged from 8% to 30% with an average percentage of 23%. In Table 4, the percentage of time saved ranged from 2% to 58% with an average percentage of 24%.

Bayesian network	Pick-ARorVE	Data-Bayes	Lazy-AR	Net %
Alarm	$1\mu s$	163	223	27%
Barley	$0\mu s$	93	128	27%
CHD	$0\mu s$	54	59	8%
Diabetes	$13\mu s$	57656	68144	15%
Hailfinder	$0\mu s$	404	564	28%
Insurance	$0\mu s$	63	80	22%
Mildew	$0\mu s$	31	41	24%
Water	$0\mu s$	340	488	30%

Table 3: The performance of Lazy-AR and DataBayes involving 9% evidence in real-world or benchmark Bayesian networks

Bayesian network	Pick ARorVE	Data-Bayes	Lazy-AR	Net %
Alarm	$1\mu s$	199	268	26%
Barley	$0\mu s$	1585	3738	58%
CHD	$0\mu s$	39	40	2%
Diabetes	$19\mu s$	118485	131231	10%
Hailfinder	$0\mu s$	280	392	29%
Insurance	$0\mu s$	152	186	18%
Mildew	$1\mu s$	177	257	31%
Water	$0\mu s$	169	213	21%

Table 4: The performance of Lazy-AR and DataBayes involving 18% evidence in real-world or benchmark Bayesian networks

The results in Tables 2, 3 and 4 empirically demonstrate that by selectively applying VE and AR, join tree propagation can be performed faster. For inference in all eight real-world or benchmark Bayesian networks, we consider three different cases: zero percent, nine percent and eighteen percent of variables randomly instantiated as evidence. In each case, DataBayes is faster than Lazy-AR without exception.

Conclusions

Current join tree propagation algorithms, such as Lazy-AR (Madsen 2004) and Lazy-VE (Madsen and Jensen 1999), utilize a single inference technique for constructing the messages throughout the entire join tree of a given Bayesian network. In this paper, we have proposed a new join tree propagation approach to probabilistic inference in Bayesian networks. In our probabilistic expert system, called *DataBayes*, we selectively apply either VE or AR to build the messages at each node in the JT. As was demonstrated, the motivation for our approach is that, during inference in real-world Bayesian networks, it is often the case that one of VE and AR is best suited to construct the messages at a particular node. In these cases, selecting the most appropriate algorithm will provide better performance than exclusively applying a single method. Therefore, DataBayes is an improvement over Lazy-VE and Lazy-AR. Our empirical comparison of DataBayes and Lazy-AR was conducted on seven real-world and one benchmark Bayesian networks. As is usually done, in each network, we performed inference with varying amounts of evidence, namely, 0%, 9% and 18%. As reported in Tables 2, 3 and 4, DataBayes finished inference sooner than Lazy-AR in all cases with no exceptions.

The novelty of our method lies in the uncoupling of semantic modelling and physical computation. By performing semantic modelling first, our flexible approach can identify the most efficient inference algorithm to apply for each separator. On the contrary, by treating these two independent tasks as dependent, Lazy-AR cannot recognize those situations when it is more efficient to employ VE as the inference algorithm. Future work will examine a more precise measure for determining whether to use AR or VE than simply the number of distributions to be constructed at a join tree node.

References

Butz, C., and Hua, S. 2006. An improved lazy-ar approach to bayesian network inference. In *Proceedings of the Nineteenth Canadian Conference on Artificial Intelligence*, 183–194.

Butz, C.; Yao, H.; and Hua, S. 2008. A join tree probability propagation architecture for semantic modeling. *Journal of Intelligent Information Systems* doi: 10.1007/s10844-008-0073-4.

Castillo, E.; Gutierrez, J.; and Hadi, A. 1997. *Expert Systems and Probabilistic Network Models*. Springer.

Cooper, G. F. 1990. The Computational Complexity of Probabilistic Inference using Bayesian Belief Networks. *Artificial Intelligence* 42:393–405.

Hájek, P.; Havránek, T.; and Jiroušek, R. 1992. *Uncertain Information Processing in Expert Systems*. CRC Press.

Kjaerulff, U., and Madsen, A. 2008. *Bayesian networks and Influence Diagrams*. Springer.

Madsen, A., and Jensen, F. 1999. LAZY propagation: A junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence* 113(1-2):203–245.

Madsen, A. 2004. An empirical evaluation of possible variations of lazy propagation. In *Proceedings of the Twentyth Conference on Uncertainty in Artificial Intelligence*, 366–373.

Olmsted, S. 1983. *On representing and solving decision problems*. Ph.D. Dissertation, Stanford University, Department of Engineering Economic Systems.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Shachter, R. D. 1986. Evaluating influence diagrams. *Operations Research* 34(6):871–882.

Shafer, G. 1996. *Probabilistic Expert Systems*. SIAM.