Compiling Relational Bayesian Networks for Exact Inference

Jaeger, Manfred; Chavira, Mark; Darwiche, Adnan

Published in:
Proceedings of the Second European Workshop on Probabilistic Graphical Models

Publication date:
2004

Document Version
Publisher's PDF, also known as Version of record

Compiling Relational Bayesian Networks for Exact Inference

Mark Chavira, Adnan Darwiche
Computer Science Department
UCLA, CA 90095
{*chavira,darwiche*}*@cs.ucla.edu*

Manfred Jaeger
Institut for Datalogi, Aalborg Universitet
Fredrik Bajers Vej 7 E, DK-9220 Aalborg
jaeger@cs.auc.dk

Abstract

We describe a system for exact inference with relational Bayesian networks as defined in the publicly available PRIMULA tool. The system is based on compiling propositional instances of relational Bayesian networks into arithmetic circuits and then performing online inference by evaluating and differentiating these circuits in time linear in their size. We report on experimental results showing the successful compilation, and efficient inference, on relational Bayesian networks whose PRIMULA–generated propositional instances have thousands of variables, and whose jointrees have clusters with hundreds of variables.

1 Introduction

Relational probabilistic models extend Bayesian network models by representing objects, their attributes and their relations with other objects. The standard approach for inference with a relational model is based on the generation of a propositional instance of the model in the form of a classical Bayesian network, and then using classical algorithms, such as the jointree (Jensen et al., 1990), to compute answers to queries.

The propositional instance of a relational model includes one Boolean random variable for each ground relational atom. For example, if we have n objects $o_1, \ldots, o_n$ in the domain, and a binary relation $R(.,.)$, we will generate a propositional variable for each instance of the relation: $R(o_1, o_1), R(o_1, o_2), \ldots, R(o_n, o_n)$. The first task in making Bayesian networks over these random variables tractable for inference is to ensure that the size of the Bayesian network representation does not show exponential growth in the number n of domain objects (as can easily happen due to nodes whose in-degree grows as a function of n). This can often be achieved by decomposing nodes with high in-degree into suitable, sparsely connected subnetworks using a number of new, auxiliary nodes. This approach is systematically employed in the PRIMULA system. Even when a reasonably compact Bayesian network representation (i.e. polynomial in the number of objects) has been constructed for a propositional instance, this model will often be inaccessible to standard algorithms for exact inference, because its global structure does not lead to tractable jointrees.

Even though the constructed networks may lack the global structure that would make them accessible to standard inference techniques, they may very well exhibit abundant local structure in the form of determinism and context–specific independence (Boutilier et al., 1996). The objective of this paper is to describe a system for inference with propositional instances of relational models which can exploit this local structure, allowing us to reason very efficiently with relational models whose propositional instances may look quite formidable at first. Specifically, we will employ the approach proposed by (Darwiche, 2003) to compile propositional instances of relational models into arithmetic circuits, and then perform online inference by simply evaluating and differentiating the compiled circuits in time linear in their size. As our experimental results illustrate, this approach efficiently handles relational models whose PRIMULA–generated propositional instances are quite massive.[1]

[1] Some may recall the technique of zero–compression which can be used to exploit determinism in the jointree framework (Jensen and Andersen, 1990). This technique, however, requires that one do inference on the original jointree before it is zero–compressed, making almost all of our data sets inaccessible to this method.

This paper is structured as follows. We start in Section 2 with a review of relational models in general and the specific formalization used in this paper. We then discuss in Section 3 the PRIMULA system, which implements this formalization together with a method for generating propositional instances in the form of Bayesian networks. Section 4 is then dedicated to our proposed approach for compiling relational models. We then provide experimental results in Section 5, and finally close with some concluding remarks in Section 6.

2 Relational Models

Relational or first–order probabilistic models extend propositional modeling supported by Bayesian networks by allowing one to represent objects explicitly, and to define relations over these objects. Most of the early work on such generic models, which has been subsumed under the title *knowledge-based model construction* (see e.g. (Breese et al., 1994)), combines elements of logic–programming with Bayesian networks. Today one can distinguish several distinct representation paradigms for relational and first–order models: (inductive) logic-programming based approaches (Sato, 1995; Muggleton, 1996; Kersting and de Raedt, 2001), network fragments (Laskey and Mahoney, 1997), frame–based representations (Koller and Pfeffer, 1998; Friedman et al., 1999), and probabilistic predicate logic formulas (Jaeger, 1997).

We will use in this paper the language of *relational Bayesian networks* (Jaeger, 1997) to represent relational models, as implemented in the PRIMULA system (http://www.cs.auc.dk/∼jaeger/Primula). The formal semantics of the language is based on *Random Relational Structure Models (*RRSMs*)*, which we define next.

Definition 1 *Given (1) a set of relational symbols S, called predefined relations; (2) a set of relational symbols R, called probabilistic relations; and (3) a finite set D, called the domain; we define an S^D-structure to be an interpretation of relations S over domain D, that is, a function which maps every ground atom $s(\boldsymbol{d})$ ($s \in S$, $\boldsymbol{d} \subseteq D$) to either $true$ or $false$. We also define a Random Relational Structure Model (*RRSM*) as a partial function which takes an S^D-structure as input, and returns a probability distribution over all R^D-structures as output.*

Intuitively, members of domain D represent *objects*, and members of S and R represent relations that can hold on these objects. These relations can be unary in which case they are called object *attributes*. A user would typically define the relations in S (by providing an S^D–structure), and then use an RRSM to induce a probability distribution over the possible definitions of relations in R (R^D–structures). We note here that S^D–structures correspond to *skeleton structures* in (Friedman et al., 1999).

We now describe three RRSMs that will be used in our experiments. These models have been implemented in PRIMULA, which provides a syntax for specifying RRSMs, and can be obtained from *http://www.cs.ucla.edu/∼chavira/pgm04*.

Random Blocks. This is a model for the random placement of blocks, which serve as obstacles, on the locations of a map, and the resulting accessibility relation among the locations. The input structures consist of a particular gridmap, and a set of blocks. This is represented using a set of predefined relations $S = \{$*location,block,leftof,belowof*$\}$ where *location* and *block* are attributes that partition the domain into the two types of objects, and *leftof* and *belowof* are binary relations that determine the spatial relationship among locations. Figure 1 shows an input S^D–structure.

One of the probabilistic relations in R for this model is the binary relation $\texttt{blocks}(b, l)$ which represents the random placement of a block b on some location l. Another is $\texttt{connected}(l_1, l_2)$ between pairs of locations which describes whether, after the placement of the blocks, there is an unblocked path between l_1 and l_2. A probabilistic query might be the probability that there is an unblocked path between two locations l_1 and l_2, given the observed locations of some blocks (but uncertainty about the placement of the remaining ones).

We will experiment with different versions of this relational model, blockmap–*l*–*b*, where l is the number of locations and b the number of blocks.

Mastermind. In the game of Mastermind, Player 1 arranges a hidden sequence of colored pegs. Player 2 guesses the exact sequence of colors by arranging guessed sequences of colored pegs. To each guessed sequence, Player 1 responds by stating

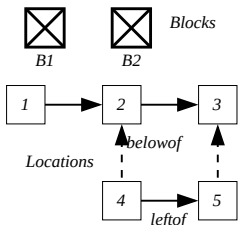


Figure 1: Input S^D–structure.

how many pegs in the guess match pegs in his hidden sequence both in color and position (white feedback), and how many pegs in the guess match pegs in the hidden sequence only in color (black feedback). This feedback is provided by placing white and black feedback pegs in a sequence. Player 2 wins if he guesses the hidden sequence within a certain number of rounds.

The game can be represented as an RRSM where the domain D consists of objects of types *peg*, *color*, and *round* specified by corresponding unary relations in S, as well as binary relations *peg-ord* and *round-ord* in S that impose orders on the peg and round objects, respectively. The probabilistic relations R in the model represent the game configurations after a number of rounds: `true-color`(p, c) represents that c is the color of the hidden peg p; `guessed-color`(p, c, r) represents that in round r color c was placed in position p in the guess. Similarly, the arrangement of the feedback pegs can be encoded. The probabilistic model specifies that all color sequences are equiprobable, but can also specify a distribution on the choice of hidden colors.

We will experiment with different versions of this model, mastermind–c–g–p, where c is the number of colors, g is the number of guesses, and p is the number of pegs.

Students and Professors. We refer the reader to *http://www.cs.ucla.edu/∼chavira/pgm04* for the definition of this last model used in our experiments, which is a variation on the model used by (Pasula and Russell, 2001) to investigate approximate inference for relational models.

3 The PRIMULA System

The RRSM is an abstraction of probabilistic relational models. For a practical system, one needs a specific syntax for specifying an RRSM. PRIMULA allows users to encode RRSMs using the language of relational Bayesian networks (Jaeger, 1997), and outputs the distribution on R^D–structures in the form of a standard Bayesian network.

3.1 Specifying RRSMs using PRIMULA

We will now provide an example of specifying an RRSM using PRIMULA. Consider a relational version of the well known domain involving individuals, their alarms, neighbors (who can be pranksters), and whether they receive calls from these neighbors when their alarm is set off. The domain D contains individuals, and the set of predefined relations S contains a unary relation, *prankster*, in addition to a binary relation *neighbor*. There are four probabilistic relations in R for this domain. The first is `calls`(v, w): whether v calls w in order to warn w that his alarm went off. The probability of `calls`(v, w) is defined conditional on the predefined *neighbor* and *prankster* relations (it is 0 if v and w are not neighbors), and on the probabilistic `alarm`(v) relation: whether the alarm of v went off. We also have another probabilistic relation `alarmed`(v): whether v has been alarmed (called by at least one neighbor). The last probabilistic relation is `burglary`(v): whether v's home has been burglarized.

This RRSM is specified in PRIMULA as given in Table 1, which provides the probability distribution on probabilistic relations using *probability formulas*. These formulas can be seen either as probabilistic analogues of predicate logic formulas, or as expressions in a functional programming language. A probability formula defines both the dependency structure between ground probabilistic atoms (which will depend on the predefined relations in the input structure), and the exact conditional probabilities, given the truth values of parent atoms.

```
burglary(v) = 0.005;
alarm(v) = (burglary(v):0.95,0.01);
calls(v,w) = (neighbor(v,w):
                    (prankster(v)):
                        (alarm(w):0.9,0.05),
                        (alarm(w):0.9,0)),0);
alarmed(v)=n-or{ calls(w,v)|w:neighbor(w,v)}
```

Table 1: Specifying an RRSM using PRIMULA.

3.2 From relational to propositional networks

To instantiate a generic relational model as in Table 1 in PRIMULA, one must provide a definition of an input S^D–structure. That is, one must define the set of individuals in domain D, and then one must define who of these individuals are pranksters (by defining the attribute *pranskter*), and who are neighbors of whom (by defining the relation *neighbor*). Given the above inputs, the distribution over probabilistic relations can be represented, as described in Section 1, using a standard Bayesian network with a node for each ground probabilistic atom. Our example also illustrates how the in-degree of a node can grow as a function of the number of domain objects: the node $\texttt{alarmed}(\textit{Holmes})$, for instance, will depend on $\texttt{calls}(w, \textit{Holmes})$ for all of Holmes's neighbors w (of which there might be arbitrarily many).

The PRIMULA system employs the general method described in (Jaeger, 2001) to decompose the dependency of a node on multiple parents. This method consists of an iterative algorithm that takes the probability formula defining the distribution of a node, decomposes it into its top–level subformulas—by introducing one new auxiliary node for each of these subformulas—and defines the probability of the original node conditional only on the new auxiliary nodes. This method can be applied to any relational Bayesian network that only contains mulit-linear combination functions (which include *noisy-or* and *mean*), and then yields a Bayesian network in which the number of parents is bounded by 3 for all nodes.

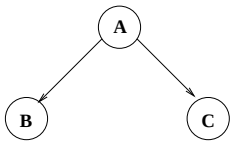
Even when one succeeds in constructing a standard Bayesian network of a manageable representation size, inference in this network may be computationally very hard. It is a long-standing open problem in first-order and relational modeling whether one might not design inference techniques that avoid these complexities of inference in the ground propositional instances by performing inference directly on the level of the relational representation, perhaps employing techniques of first-order logical inference. Complexity results derived in (Jaeger, 2000) show that one cannot hope for a better worst-case performance with such inference techniques. This still leaves the possibility that they could often lead to substantial gains in practice.

A high-level inference technique that aims at achieving such gains in average-case performance has recently been described by Poole (Poole, 2003). The potential advantage of this and similar inference techniques seems to be restricted, however, to relational models where individual model instances are given by relatively unstructured input structures, i.e. input structures containing large numbers of indistinguishable objects. The potential of high-level inference techniques lies in their ability to deal with such sets of objects without explicitly naming each object individually. However, in the type of relational models we are here considering, the input structures consist of mostly unique objects (in Random Blocks, for instance, the block objects are indistinguishable, but all location objects have unique properties defined by the *belowof* and *leftof* relations). We can identify an input structure with the complete ground propositional theory that defines it (for the structure of Figure 1 this would be the theory $\texttt{block}(B1) \wedge \neg\texttt{location}(B1) \wedge \ldots \wedge \texttt{leftof}(2,3) \wedge \ldots \wedge \neg\texttt{belowof}(5,5)$), and, informally, characterize highly structured input structures as those for which this propositional theory admits no simple first-order abstraction.[2] When a relational model instance, now, is given by an input structure that cannot be succinctly encoded in an abstract, first-order style representation, chances are very small that probabilistic inference for this model instance can gain much efficiency by operating on a non-propositional level.

It thus appears that at least for a fairly large class of interesting models more advantages might be gained by optimizing inference techniques for ground propositional models, than by non-propositional inference techniques.

Table 2 depicts the relational models we experimented with, together with the size of corresponding propositional Bayesian networks generated by PRIMULA's decomposition method. The table also reports the size of largest cluster in the jointree constructed for these networks. Obviously, most of these networks are inaccessible to mainstream, structure–based algorithms for exact inference. Yet,

[2] Formally one would define the 'structuredness' of an input structure in terms of the number of its automorphisms.



A	B	
$true$	$true$	$\theta_{b\|a} = 0$
$true$	$false$	$\theta_{\bar{b}\|a} = 1$
$false$	$true$	$\theta_{b\|\bar{a}} = .7$
$false$	$false$	$\theta_{\bar{b}\|\bar{a}} = .3$

Figure 2: A Bayesian net with one of its CPTs.

we will show later that all of these models can be handled efficiently using the compilation approach we propose in this paper.

4 Compiling Relational Models

We describe in this section the approach we use to perform exact inference on propositional instances of relational models, which is based on compiling Bayesian networks into arithmetic circuits (Darwiche, 2003). Inference can then be performed using a simple two–pass procedure in which the circuit is evaluated and differentiated given evidence.

4.1 Bayesian networks as polynomials

The compilation approach we adopt is based on viewing each Bayesian network as a very large polynomial (multi–linear function in particular), which may be compactly represented using an arithmetic circuit. The function itself contains two types of variables.[3] For each value x of each variable X in the network, we have a variable λ_x called an evidence indicator. For each instantiation $x, \mathbf{u}$ of each variable X and its parents $\mathbf{U}$ in the network, we have a variable $\theta_{x|\mathbf{u}}$ called a network parameter. The multi–linear function has a term for each instantiation of the network variables, which is constructed by multiplying all evidence indicators and network parameters that are consistent with that instantiation. For example, the multi–linear function of the network in Figure 2 has 8 terms corresponding to the 8 instantiations of variables A, B, C: $f = \lambda_a\lambda_b\lambda_c\theta_a\theta_{b|a}\theta_{c|a} + \lambda_a\lambda_b\lambda_{\bar{c}}\theta_a\theta_{b|a}\theta_{\bar{c}|a} + \ldots +$ $\lambda_{\bar{a}}\lambda_{\bar{b}}\lambda_{\bar{c}}\theta_{\bar{a}}\theta_{\bar{b}|\bar{a}}\theta_{\bar{c}|\bar{a}}$. Given this multi–linear function f, we can answer standard queries with respect to its corresponding Bayesian network by simply evaluating and differentiating this function; see (Darwiche, 2003) for details.

[3] We are using the standard notation: variables are denoted by upper–case letters (A) and their values by lower–case letters (a). Sets of variables are denoted by bold–face upper–case letters ($\mathbf{A}$) and their instantiations are denoted by bold–face lower–case letters ($\mathbf{a}$). For a variable A with values $true$ and $false$, we use a to denote $A{=}\ true$ and $\bar{a}$ to denote $A{=}\ false$. Finally, for a variable X and its parents $\mathbf{U}$, we use $\theta_{x|\mathbf{u}}$ to denote the CPT entry corresponding to $Pr(x \mid \mathbf{u})$.

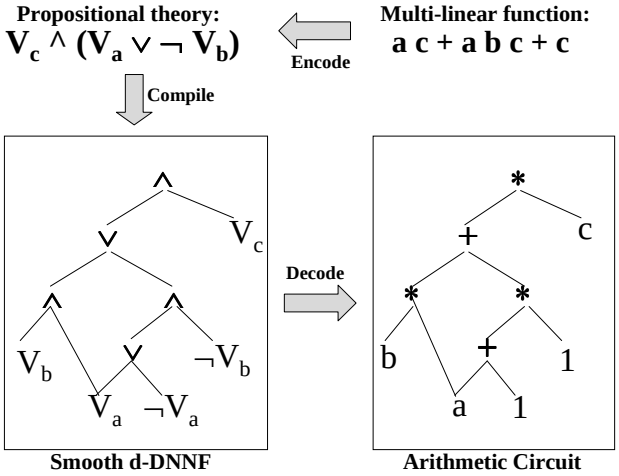


Figure 3: Factoring multi–linear functions into arithmetic circuits.

The ability to compute answers to probabilistic queries directly from the derivatives of f is interesting semantically, but one must realize that the size of function f is exponential in the number of network variables. Yet, one may be able to factor this function and represent it more compactly using an arithmetic circuit. *An arithmetic circuit* is a rooted DAG, in which each leaf represents a variable or constant and each internal node represents the product or sum of its children; see Figure 3. If we can represent the network polynomial efficiently using an arithmetic circuit, then inference can be done in time linear in the size of such circuits, since the (first) partial derivatives of an arithmetic circuit can all be computed simultaneously in time linear in the circuit size (Darwiche, 2003).

4.2 Compiling the network polynomial into an arithmetic circuit

We now turn to the approach for compiling/factoring network polynomials into arithmetic circuits, which is based on reducing the factoring problem to one of logical reasoning (Darwiche, 2002). This approach is based on three conceptual steps, as shown in Figure 3. First, the network polynomial is encoded using a propositional theory.

Next, the propositional theory is factored by converting it to a special logical form. Finally, an arithmetic circuit is extracted from the factored propositional theory.

Step 1: Encoding a multi–linear function using a propositional theory. The purpose of this step is to specify the network polynomial using a propositional theory. To illustrate how a multi–linear function can be specified using a propositional theory, consider the following function $f = ac + abc + c$ over real–valued variables a, b, c. The basic idea is to specify this multi–linear function using a propositional theory that has exactly three models, where each model encodes one of the terms in the function. Specifically, suppose we have the Boolean variables V_a, V_b, V_c. Then the propositional theory $\Delta_f = (V_a \vee \neg V_b) \wedge V_c$ encodes the multi–linear function f as follows:

Model	V_a	V_b	V_c	*encoded term*
σ_1	$true$	$false$	$true$	ac
σ_2	$true$	$true$	$true$	abc
σ_3	$false$	$false$	$true$	c

That is, if model σ_i encodes term t_i, then σ_i sets Boolean variable V_j to true iff t_i contains the corresponding real–valued variable. This method of specifying network polynomials allows one to easily capture local structure; that is, to declare certain information about values of polynomial variables. For example, if we know that parameter $a = 0$, then we can exclude all terms that contain a by conjoining $\neg V_a$ with our encoding.

Step 2: Factoring the propositional encoding. If we view the conversion of a network polynomial into an arithmetic circuit as a factoring process, then the purpose of this second step is to accomplish a similar task but at the logical level. Instead of starting with a polynomial (set of terms), we start with a propositional theory (set of models). And instead of building an arithmetic circuit, we build a Boolean circuit that satisfies certain properties. Specifically, the circuit must be in Negation Normal Form (NNF): a rooted DAG where leaves are labelled with literals, and where internal nodes are labelled with conjunctions or disjunctions; see Figure 3. The NNF must satisfy three properties: (1) conjuncts cannot share variables (decomposability), (2) disjuncts must be logically exclusive (determinism), and (3) disjuncts must be over the same variables (smoothness). The NNF in Figure 3 satisfies the above properties, and encodes the multi–linear function shown in the same figure. In our experimental results, we use a second generation compiler for converting CNFs to NNFs that are decomposable, deterministic and smooth (smooth d-DNNF) (Darwiche, 2004).

Step 3: Extracting an arithmetic circuit. The purpose of this last step is to extract an arithmetic circuit for the polynomial encoded by an NNF. If Δ_f is an NNF that encodes a network polynomial f, and if Δ_f is a smooth d-DNNF, then an arithmetic circuit for the polynomial f can be obtained easily as follows. First, replace and–nodes in Δ_f by multiplications; then replace or–nodes by additions; and finally, replace each leaf node labelled with V_x by x and each node labelled with $\neg V_x$ by 1. The resulting arithmetic circuit is then guaranteed to correspond to polynomial f (Darwiche, 2002). Figure 3 depicts an NNF and its corresponding arithmetic circuit. Note that the generated arithmetic circuit is no larger than the NNF. Hence, if we attempt to minimize the size of NNF, we are also minimizing the size of generated arithmetic circuit.

4.3 Encoding PRIMULA's networks

As mentioned earlier, the PRIMULA system generates propositional instances of relational models in the form of classical Bayesian networks. These networks, however, have a specific structure that we exploit when encoding the network polynomial as CNF; we do not use the exact encodings proposed in (Darwiche, 2002). In particular, the networks generated by PRIMULA will only have binary variables, and each node is restricted to have no more than 3 parents.

These specific properties of the generated networks allow one to use a tailored, efficient encoding of the network polynomial. Specifically, instead of using one propositional variable for each evidence indicator λ_x—which would be needed in general—we use one propositional variable I_X for each Bayesian network variable X, where the positive literal I_X represents indicator λ_x, and the negative literal $\neg I_X$ represents indicator $\lambda_{\bar{x}}$. Not only does this cut the number of indicator variables by half, but it also relieves the need for clauses of

the form, $\lambda_x \vee \lambda_{\bar{x}}$ and $\neg\lambda_x \vee \neg\lambda_{\bar{x}}$ which would be needed to ensure that exactly one indicator for variable X can appear in any polynomial term; see (Darwiche, 2002). The CNF encoding will also include one variable $P_{\theta_{x|\mathbf{u}}}$ for each network parameter $\theta_{x|\mathbf{u}}$ which is not equal to 0 or 1. Given these variables, the CNF will then only include the following clauses: If $\theta_{x|u_1,\ldots,u_n} = 0$, include clause $L_{U_1} \vee \ldots \vee L_{U_n}$, where L_{U_i} is a literal over variable I_{U_i} whose sign is opposite to the sign of u_i. If $\theta_{x|u_1,\ldots,u_n} \neq 0, \neq 1$, include clauses $L_{U_1} \vee \ldots \vee L_{U_n} \Rightarrow P_{\theta_{x|u_1,\ldots,u_n}}$, $P_{\theta_{x|u_1,\ldots,u_n}} \Rightarrow L_{U_1}$, $\ldots$, $P_{\theta_{x|u_1,\ldots,u_n}} \Rightarrow L_{U_n}$, where L_{U_i} is a literal as defined earlier.

For example, the CPT for variable B in Figure 2 will generate the following clauses: (1st row) $\neg I_A \vee \neg I_B$, (3nd row) $\neg I_A \wedge I_B \Rightarrow P_{\theta_{b|\bar{a}}}$, $P_{\theta_{b|\bar{a}}} \Rightarrow \neg I_A$, $P_{\theta_{b|\bar{a}}} \Rightarrow I_B$, (4th row) $\neg I_A \wedge \neg I_B \Rightarrow P_{\theta_{\bar{b}|\bar{a}}}$, $P_{\theta_{\bar{b}|\bar{a}}} \Rightarrow \neg I_A$, and $P_{\theta_{\bar{b}|\bar{a}}} \Rightarrow \neg I_B$.

Given that all network variables are binary, and given that each node has at most three parents, this encoding leads to a CNF whose size is linear in the number of network variables. Table 2 depicts the size of CNF encodings for the relational models we experimented with. The number of clauses for these encodings is usually smaller than the number of parameters in the corresponding Bayesian networks.

The special encoding used above calls for a slightly different decoding scheme for transforming a smooth d-DNNF into an arithmetic circuit. Specifically, literals I_X and $\neg I_X$ are replaced with evidence indicators λ_x and $\lambda_{\bar{x}}$, respectively. Moreover, literals $P_{\theta_{x|\mathbf{u}}}$ and $\neg P_{\theta_{x|\mathbf{u}}}$ are replaced by $\theta_{x|\mathbf{u}}$ and 1, respectively. Finally, conjunctions and disjunctions are replaced by multiplications and additions.

5 Experimental Results

Our experimental results were obtained on a 1.6GHz Pentium M with 2GB of RAM. Table 2 lists for each relational model a number of instances, and for each instance, a number of findings. First is the size and connectivity of the Bayesian network that PRIMULA generated. Next is the size of CNF encoding in terms of the number of Boolean variables and clauses. Third, the table shows the size of the compiled arithmetic circuit in terms of both number of nodes and edges (count and log base 2). We also show the time it takes to evaluate and differentiate the circuit, averaged over 31 different pieces of evidence. Note that by evaluating and differentiating the circuit, one obtains the marginals over all network families, in addition to other probabilities discussed in (Darwiche, 2003).

The main points to observe are the efficiency of online inference on compiled circuits and the size of these circuits compared to the size and connectivity of the Bayesian networks. Table 2 also shows the time for jointree propagation using the SamIam inference engine (http://reasoning.cs.ucla.edu/samiam) on instances whose cluster size was manageable. One can see the big difference between online inference using the compiled AC and corresponding jointrees.

Table 2 finally shows the compile time to generate the arithmetic circuits. The compile times range from less than a minute to about 60 minutes for the largest model (blockmap-22-03). Yet note that the time for online inference ranges from milliseconds to about 13 seconds for these models. This clearly shows the benefit of offline compilation in this case, whose time can be amortized over online queries.

6 Conclusion

We described in this paper an inference system for relational Bayesian networks as defined by PRIMULA. The proposed inference approach is based on compiling propositional instances of these models into arithmetic circuits. The approach exploits local structure in relational models, allowing us to reason efficiently with relational models whose PRIMULA–generated propositional instances contain thousand of variables, and whose jointrees contain hundreds of variables. The described system appears to significantly expand the scale of PRIMULA–based relational models that can be handled efficiently by exact inference algorithms.

References

C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context–specific independence in bayesian networks. *UAI'96*.

J. S. Breese, R. P. Goldman, and M. P. Wellman. 1994. Introduction to the special section on knowledge-based construction of probabilistic decision models. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(11).

Relational Model	Bayesian Network			CNF Encoding		Arithmetic Circuit			AC Time		JT
	Vars	CPT Parameters	Largest Cluster	Vars	Clauses	Nodes	Edges Count	Log	Inf (sec)	Comp (min)	Inf (sec)
mastermind C–R–P											
03–08–03	1220	8326	23	1328	4379	26021	339505	18.4	0.0291	1	8.25
04–08–03	1418	9802	26	1580	5252	71666	541356	19.0	0.0516	1	57.48
05–08–03	1616	11278	32	1832	6125	149982	942167	19.8	0.0930	1	
06–08–03	1814	12754	37	2084	6998	258228	1523888	20.5	0.1518	1	
10–08–03	2606	18658	54	3092	10490	1293323	4315566	22.0	0.6835	3	
03–08–04	2288	16008	31	2432	8292	186351	4859201	22.2	0.2997	2	
04–08–04	2616	18488	39	2832	9712	932355	19457308	24.2	1.7341	5	
03–08–05	3692	26186	40	3872	13453	1359391	55417639	25.7	4.3253	10	
students P–S											
03–02	376	2616	25	618	2131	7927	37281	15.2	0.0052	1	6.14
03–06	764	5512	50	1454	5147	110196	595737	19.2	0.0588	1	
03–12	1346	9856	59	2708	9671	24219	113876	16.8	0.0175	1	
04–08	1571	11566	72	3099	11099	95649	445410	18.8	0.0530	2	
04–16	2827	21070	101	5859	21115	181166	815461	19.6	0.0930	3	
05–10	2774	20688	128	5624	20279	630092	2531230	21.3	0.2885	3	
05–20	5064	38168	148	10734	38889	1319834	5236257	22.3	1.8439	7	
06–12	4445	33454	176	9209	33353	4586368	16936504	24.0	3.2120	14	
06–24	8201	62302	233	17693	64325	9922233	36450231	25.1	12.9663	33	
blockmap L–B											
05–01	700	4784	18	708	2412	1255	3364	11.7	0.0052	1	2.70
05–02	855	5898	21	875	2999	1751	12306	13.6	0.0058	1	6.36
05–03	1005	6972	23	1035	3561	2833	20636	14.3	0.0068	1	27.39
10–01	5650	40070	52	5670	20083	10147	56998	15.8	0.0136	1	
10–02	6252	44444	53	6292	22318	11978	309176	18.2	0.0255	1	
10–03	6848	48758	52	6908	24529	17749	974817	19.9	0.0582	2	
15–01	16497	116048	68	16525	58094	29347	224826	17.8	0.0349	2	
15–02	17649	124298	70	17709	62299	33011	1798085	20.8	0.1085	3	
15–03	18787	132436	68	18877	66443	47475	7643307	22.9	0.3799	6	
20–01	39297	278138	90	39335	139164	69208	726787	19.5	0.0940	6	
20–02	41337	292760	90	41413	146570	75299	6989375	22.7	0.3757	10	
20–03	43356	307220	92	43476	153910	105602	40172434	25.3	2.4529	30	
22–01	54318	386842	104	54360	193526	96424	1103074	20.1	0.1408	10	
22–02	56873	405240	103	56957	202830	103980	11707536	23.5	0.8227	20	
22–03	59404	423452	104	59536	212056	144136	76649302	26.2	4.6651	61	

Table 2: Relational Bayesian networks, their corresponding propositional instances, the sizes of their CNFs encodings, the sizes of their ACs, and times for online inference and compilation.

A. Darwiche. 2002. A logical approach to factoring belief networks. In *Proceedings of KR*, pages 409–420.

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

A. Darwiche. 2004. New advances in compiling CNF to decomposable negational normal form. Technical Report D–141, Computer Science Department, UCLA, Los Angeles, Ca 90095. To appear in ECAI'04.

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. *IJCAI'99*.

M. Jaeger. Relational bayesian networks. *UAI'97*.

M. Jaeger. 2000. On the complexity of inference about probabilistic relational models. *Artificial Intelligence*, 117:297–308.

M. Jaeger. 2001. Complex probabilistic modeling with recursive relational Bayesian networks. *Annals of Mathematics and Artificial Intelligence*, 32:179–220.

F. Jensen and S. K. Andersen. Approximations in Bayesian belief universes for knowledge based systems. *UAI'90*.

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

K. Kersting and L. de Raedt. 2001. Towards combining inductive logic programming and bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP-2001)*, Springer Lecture Notes in AI 2157.

D. Koller and A. Pfeffer. Probabilistic frame-based systems. *AAAI'98*.

K. B. Laskey and S. M. Mahoney. Network fragments: Representing knowledge for constructing probabilistic models. *UAI'97*.

S. Muggleton. 1996. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press.

H. Pasula and S. Russell. 2001. Approximate inference for first-order probabilistic languages. *IJCAI'2001*.

D. Poole. 2003. First-order probabilistic inference. *IJCAI'2003*.

T. Sato. 1995. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)*, pages 715–729.