Title
Compiling relational Bayesian networks for exact inference

Journal
International Journal of Approximate Reasoning, 42(1-2)

ISSN
0888-613X

Authors
Chavira, M D
Darwiche, A
Jaeger, M

Publication Date
2006-05-01

Peer reviewed

Compiling Relational Bayesian Networks for Exact Inference

Mark Chavira, Adnan Darwiche

Computer Science Department, UCLA, Los Angeles, CA 90095

Manfred Jaeger

Institut for Datalogi, Aalborg Universitet, Fredrik Bajers Vej 7 E, DK-9220 Aalborg Ø

Abstract

We describe in this paper 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 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.

Key words: Exact Inference, Relational Models, Bayesian Networks

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 applying classical algorithms, such as jointree [1], to compute answers to queries.

Email addresses: chavira@cs.ucla.edu (Mark Chavira), darwiche@cs.ucla.edu (Adnan Darwiche), jaeger@cs.aau.dk (Manfred Jaeger).

The propositional instance of a relational model includes one Boolean random variable for each ground relational atom. For example, if we have n domain objects $o_1, \ldots, o_n$, and a binary relation $R(.,.)$, we 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 sub-networks 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. 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 some relational models whose propositional instances may look quite formidable at first. Specifically, we employ the approach proposed by [2] to compile propositional instances of relational models into arithmetic circuits, and then perform online inference by evaluating and differentiating the compiled circuits in time linear in their size. As our experimental results illustrate, this approach can efficiently handle some relational models whose PRIMULA–generated propositional instances are quite massive. [1] We note here that the inference approach of [2] is applicable to any Bayesian network, but is especially effective on networks with local structure, including determinism. Hence, one of the main points of this paper is to illustrate the extent of local structure available in propositional instances of relational models, and the effectiveness in exploiting this local structure by the approach proposed in [2].

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

[1] Some may recall the technique of zero–compression which can be used to exploit determinism in the jointree framework [3]. This technique, however, requires that one perform inference on the original jointree before it is zero–compressed, making almost all of our data sets inaccessible to this method. For a more detailed relationship to jointree inference, the reader is referred to [4].

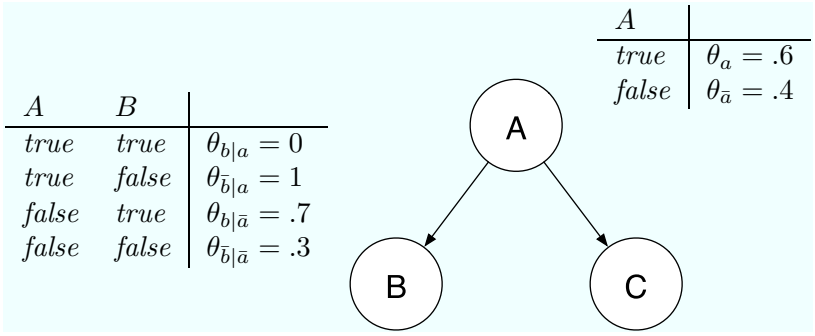

Fig. 1. A Bayesian net with two of its CPTs.

approach for compiling relational models. We provide experimental results in Section 5, and finally close with some concluding remarks in Section 6.

2 Relational Models

A Bayesian network is a compact representation of a probability distribution and has two parts: a directed acyclic graph and a set of conditional probability tables (CPTs). Each node in the graph represents a random variable, which we assume to be discrete in this paper. Each variable X has associated with it a CPT, which specifies the conditional probabilities $Pr(x \mid \mathbf{u})$, where $\mathbf{u}$ is a configuration of the parents $\mathbf{U}$ of X in the network.

A Bayesian network over a set of variables specifies a unique probability distribution over these variables. Probabilistic queries with respect to a Bayesian network are to be interpreted as queries with respect to the probability table the network specifies. The main goal of algorithms for Bayesian networks is to answer such queries without having to construct the table explicitly, since the table's size is exponential in the number of network variables. Figure 1 depicts a simple Bayesian network with two of its CPTs.

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. [5]), 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 [6–8], network fragments [9], frame–based representations [10,11], and probabilistic predicate logic formulas [12]. We review relational models with an example.

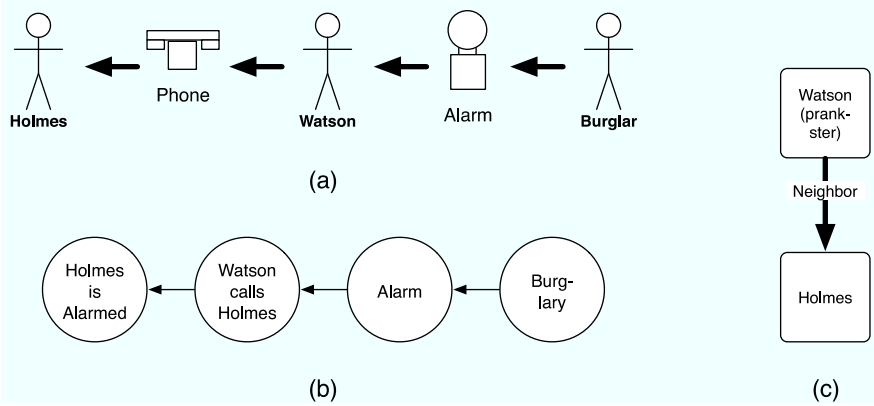

Fig. 2. (a) A simple alarm scenario, (b) the corresponding Bayesian network, and (c) a graph depicting the particulars of the situation, as opposed to what is common to all alarm situations.

2.1 An Example

Consider the well–known example depicted in Figure 2(a), in which Holmes becomes alarmed if he receives a call from his neighbor Watson. Watson will likely call if an alarm has sounded at Holmes' residence, which is more likely if a burglary occurs. However, Watson is a prankster, so Holmes may receive a call even if the alarm does not sound. We can model this example with a Bayesian network as shown in Figure 2(b). A query might be the probability that there is a burglary given that Holmes is alarmed. We could also consider similar scenarios. Holmes might have multiple neighbors (only some of whom are pranksters) and become alarmed if any of them calls. There might be multiple individuals who can receive calls, each with distinct neighbors. Or it might be that individuals share neighbors and individuals who receive calls can also make them. For each of these scenarios, we can construct a distinct Bayesian network. Moreover, we can imagine needing to deal with many of these situations, and hence needing to construct many different networks.

Each of the situations described represents a combination of various themes, such as the theme of an alarm compelling a neighbor to call or an individual becoming alarmed when some neighbor calls. Relational models address domains involving themes by separating the model construction process into two phases. We first describe a set of general rules that apply to all situations. For example, in the alarm domain described, we need four rules:

(1) At a given residence, the probability of burglary is 0.005.
(2) A particular alarm sounds with probability 0.95 if a burglary occurs at the corresponding residence, and with probability 0.01 otherwise.
(3) If an alarm sounds at an individual's residence, then each of the individual's neighbors will call with probability 0.9; otherwise, if the neighbor is a prankster, then the neighbor will call with probability 0.05; otherwise,

the neighbor will not call.
(4) An individual is alarmed if one or more neighbors call.

We highlight here that whether an individual is alarmed depends on the number of the individual's neighbors, which makes this domain difficult represent with a template–based language.

Once we have specified what is common to all situations, in order to specify a particular situation, we only need specify a small amount of additional information. In the alarm example, that information consists of which individuals are involved (other than burglars), who are neighbors of whom, and who are pranksters. We specify a graph where nodes represent individuals, edges capture the neighbor relationship, and each node is marked if the corresponding individual is a pranktser. Figure 2(c) depicts the graph corresponding to the situation in Figure 2(a).

One of the main advantages of using a relational model is that a relational model describes a situation involving themes succinctly. This advantage often makes constructing a relational model much easier and less error–prone than constructing a Bayesian network. For example, it is not uncommon for a relational model with a dozen or so general rules to correspond to a Bayesian network that involves hundreds of thousands of CPT parameters. Another advantage is that much of the work performed in constructing a relational model can be directly re-used in describing variations of the model, whereas creating another Bayesian network can involve much more work.

2.2 Relational Bayesian Networks

We use in this paper the language of *relational Bayesian networks* [12] to represent relational models, as implemented in the PRIMULA system available at *http://www.cs.aau.dk/∼jaeger/Primula*. The formal semantics of the language is based on *Random Relational Structure Models (*RRSM*s)*, 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 $s(d)$ ($s \in S$, $d \subseteq D$) 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 *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 [11]. For the alarm example above, the set D of objects is the set of individuals. 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 $\texttt{calls}(v, w)$: whether v calls w in order to warn w that his alarm went off. We also have another probabilistic relation $\texttt{alarmed}(v)$: whether v has been alarmed (called by at least one neighbor). A third is the relation $\texttt{alarm}(v)$: whether v's alarm went off. The last probabilistic relation is $\texttt{burglary}(v)$: whether v's home has been burglarized. The RRSM is the set of four generic rules described previously.

We now describe four RRSMs used in our experiments. These models have been implemented in PRIMULA, which provides a syntax for specifying RRSM.

Random Blocks. This model describes the random placement of blocks (obstacles) on the locations of a map. 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 3 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 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 experiment with different versions of this relational model, blockmap–l–b, where l is the number of locations and b the number of blocks.

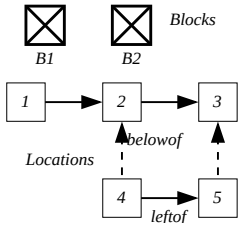

Fig. 3. Input S^D–structure.

Mastermind. In the game of Mastermind, Player 1 arranges a hidden sequence of colored pegs. Player 2 guesses the exact sequence of colors by ar-

ranging guessed sequences of colored pegs. To each guessed sequence, Player 1 responds by stating 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). 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: $\texttt{true-color}(p, c)$ represents that c is the color of the hidden peg p; $\texttt{guessed-color}(p, c, r)$ 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. A query might be the most probable color configuration of the hidden pegs, given the observed query and feedback pegs. We 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. This domain was used by [13] to investigate methods for approximate inference for relational models. We have two types of objects in this model: *students* and *professors* and two corresponding attributes in the set S. Professors have two probabilistic attributes in R: `fame` (yes/no) and `funding_level` (high/low). Students have one probabilistic attribute in R: `success` (yes/no). Students and professors are related via the binary probabilistic relation $\texttt{advisor}(s, p)$ in R. According to the model, students use the *softmax* rule, and choose advisor i with funding level y_i with probability $e^{y_i} / \sum_k e^{y_k}$. With the funding level discretized into two categories high and low, this reduces to choosing any given rich (poor) professor with probability $z_h/(Kz_h + Lz_l)$ $(z_l/(Kz_h + Lz_l))$, where K is the number of rich professors, L is the number of poor professors, and z_h, z_l are the (exponentials of) the funding levels of rich, respectively poor, professors. The probability of success of a student is defined conditional on the funding level. A query for this model can be the probabilities for a professor's funding level, given the success of his students. Inference in this model becomes hard very quickly with increasing numbers of professors and students in the domain [13]. We will experiment with different versions of this relational model, students–p–s, where p is the number of professors and s is the number of students.

Friends and Smokers. This domain was introduced in [14]. It involves a number of individuals, with relations in R, such as $\texttt{smokes}(v)$, which indicates whether a person smokes, $\texttt{cancer}(v)$, which indicates whether a person has cancer, and $\texttt{friends}(u, v)$, which indicates who are friends of whom. There are no relations in S for this model. The probabilistic model over R is defined by assigning weights to logical constraints, such as $\texttt{friends}(u, v) \wedge \texttt{smokes}(u) \rightarrow \texttt{smokes}(v)$. A query for this model might be the probability that a person has

cancer given information about others who have cancer. The PRIMULA encoding of this model utilizes auxiliary probabilistic relations corresponding to the logical constraints. In ground instances of the model these auxiliary variables manifest themselves as variables in the Bayesian network, on which evidence should be asserted to indicate that they are always *true*. We experiment with different versions of this relational model, fr&sm–n, where n is the number of people in the domain.

3 The PRIMULA System

The RRSM is an abstract semantics 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 [12], and outputs the distribution on R^D–structures in the form of a standard Bayesian network.

3.1 Specifying RRSM*s using* PRIMULA

We now provide an example of specifying an RRSM using PRIMULA. Consider again the alarm example from Section 2.1 and recall that for this example, the domain is the set of individuals, the set of predefined relations is $S = \{\texttt{prankster}(v), \texttt{neighbor}(v, w)\}$, and the set of probabilistic relations is $R = \{\texttt{calls}(v, w), \texttt{alarm}(v), \texttt{alarmed}(v), \texttt{burglary}(v)\}$. The probability of $\texttt{calls}(v, w)$ is defined conditional on the predefined *neighbor* and *prankster* relations (it is 0 if v and w are not neighbors), and on the probabilistic $\texttt{alarm}(v)$ relation: whether the alarm of v went off.

This RRSM is specified in PRIMULA as given in Figure 4, 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 depends on the predefined relations in the input structure), and the exact conditional probabilities, given the truth values of parent atoms.

The specification of the RRSM provides some intuition for why a logic–based approach might work well when applied to PRIMULA generated networks. In addition to certain numbers, we also see in this specification a number of logical constructs. For example, each of the occurrences of $(x : y, z)$ is essentially an application of an *if–then–else*, and the *noisy-or* construct is essentially an existential quantification, which can be converted into a disjunction over a

```
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)}
```

Fig. 4. Specifying an RRSM using PRIMULA.

```
DOMAIN: Holmes, Watson, Gibbon;
RELATION: prankster/1 {2};
RELATION: neighbor/2 {(0,1) (0,2) (1,0) (2,0)};
```

Fig. 5. Specifying an S^D structure using PRIMULA.

set of auxiliary variables. The utilization of these logical constructs is quite common in relational models.

3.2 From relational to propositional networks

To instantiate a generic relational model in PRIMULA, one must provide a definition of an input S^D–structure. For the RRSM defined in Figure 4, one must define the set of individuals in domain D, and then one must define which of these individuals are pranksters (by defining the attribute *prankster*), and who are neighbors of whom (by defining the relation *neighbor*). PRIMULA provides a GUI for this purpose, but one can also supply a file–based definition of the domain and corresponding S relations. Figure 5 presents what one of these files might look like. This file defines the domain to be $D = \{Holmes, Watson, Gibbon\}$ and specifies that *Gibbon* is a prankster, that $Holmes$ is a neighbor of $Watson$ and *Gibbon* and that $Watson$ and *Gibbon* are neighbors of $Holmes$.

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 `alarmed`($Holmes$), for instance, depends on `calls`($w, Holmes$) for all of Holmes's neighbors w (of which there might be arbitrarily many).

The PRIMULA system employs the general method described in [15] 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 prob-

ability of the original node conditional only on the new auxiliary nodes. This method can be applied to any relational Bayesian network that only contains multi-linear combination functions (including *noisy-or* and *mean*), and yields a Bayesian network where the number of parents is bounded by three 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 [16] 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.

Recent work has described high-level inference techniques that aim at achieving such gains in average-case performance [17,18]. The potential advantage of this and similar 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 considering here, 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 3 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. 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 1 depicts the relational models with which we experimented, together with the size of corresponding propositional Bayesian networks generated by PRIMULA. The table also reports the size of the largest cluster for the jointree

we constructed for these networks. Obviously, most of these networks are inaccessible to mainstream, structure–based algorithms for exact inference. Yet, we will show later that all of these particular 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 [2]. 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. 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 1 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 [2] for details.

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 6. 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

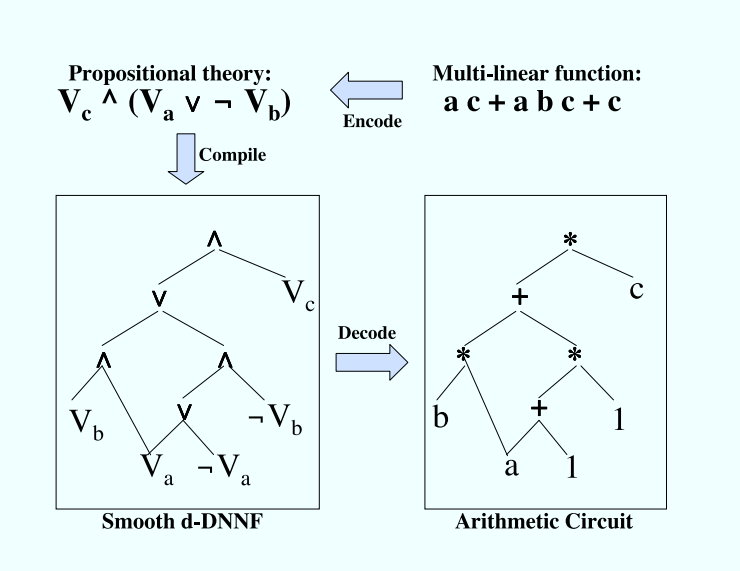

Fig. 6. Factoring multi–linear functions into arithmetic circuits.

the circuit size [2].

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 [19]. This approach is based on three conceptual steps, as shown in Figure 6. 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. [2]

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

[2] A similar approach has been recently proposed in [20], which calls for encoding Bayesian networks into CNFs, and reducing probabilistic inference to weighted model counting on the generated CNFs. The approach is similar in two senses. First, the weighted model counting algorithm applied in [20] is powerful enough to factor the CNF as suggested by Step 2 below—see [21]. Second, the factored logical form we generate from the CNF in Step 2 is tractable enough to allow weighted model counting in time linear in the form size [22,23].

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, model σ encodes term t since $\sigma(V_j) = true$ precisely when term t contains the real–valued variable j. 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 labeled with literals, and where internal nodes are labeled with conjunctions or disjunctions; see Figure 6. 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 6 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) [24].

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. First, replace and–nodes in Δ_f by multiplications; then replace or–nodes by additions; and finally, replace each leaf node labeled with V_x by x and each node labeled with $\neg V_x$ by 1. The resulting arithmetic circuit is then guaranteed to correspond to polynomial f [19]. Figure 6 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 attempting to minimize the size of generated arithmetic circuit.

The encoding step described above is semantic; that is, it describes the theory Δ_f which encodes a multi–linear function by describing its models. As mentioned earlier, the PRIMULA system generates propositional instances of relational models in the form of classical Bayesian networks. We now turn to the question of how to syntactically represent in CNF the multi–linear function of a network so generated. We start with the baseline encoding defined in [19], which applies to any Bayesian network. The CNF has one Boolean variable I_λ for each indicator variable λ, and one Boolean variable P_θ for each parameter variable θ. CNF clauses fall into three sets. First, for each network variable X with domain $x_1, x_2, \ldots, x_n$, we have:

$$\begin{aligned} \underline{\text{Indicator clauses}} : & \; I_{\lambda_{x_1}} \vee I_{\lambda_{x_2}} \vee \ldots \vee I_{\lambda_{x_n}} \\ & \; \neg I_{\lambda_{x_i}} \vee \neg I_{\lambda_{x_j}}, \text{ for } i < j \end{aligned}$$

For example, variable B from Figure 1 generates the following clauses:

$$I_{\lambda_b} \vee I_{\lambda_{\bar{b}}}, \quad \neg I_{\lambda_b} \vee \neg I_{\lambda_{\bar{b}}} \tag{1}$$

These clauses ensure that exactly one indicator variable for B appears in every term of the multi–linear function. The second two sets of clauses correspond to network parameters. In particular, for each parameter $\theta_{x_n|x_1,x_2,\ldots,x_{n-1}}$, we have:

$$\begin{aligned} &\underline{\text{IP clause}} : I_{\lambda_{x_1}} \wedge I_{\lambda_{x_2}} \wedge \ldots \wedge I_{\lambda_{x_n}} \Rightarrow P_{\theta_{x_n|x_1,x_2,\ldots,x_{n-1}}} \\ &\underline{\text{PI clauses}} : P_{\theta_{x_n|x_1,x_2,\ldots,x_{n-1}}} \Rightarrow I_{\lambda_{x_i}}, \text{ for each } i \end{aligned}$$

For example, parameter $\theta_{b|a}$ in Figure 1 generates the following clauses:

$$I_{\lambda_a} \wedge I_{\lambda_b} \Rightarrow P_{\theta_{b|a}}, \quad P_{\theta_{b|a}} \Rightarrow I_{\lambda_a}, \quad P_{\theta_{b|a}} \Rightarrow I_{\lambda_b} \tag{2}$$

These clauses ensure that $\theta_{b|a}$ appears in a term iff the λ_a and λ_b appear. The encoding as discussed does not capture information about parameter values (local structure). However, it is quite easy to encode information about determinism within this encoding. Consider again Figure 1 and the parameter $\theta_{b|a} = 0$, which generates the clauses in Equation 2. Given that this parameter is known to be 0, all multi–linear terms that contain this parameter must vanish. Therefore, we can suppress the generation of a Boolean variable for this parameter, and then replace the above clauses by the single clause: $\neg I_{\lambda_a} \vee \neg I_{\lambda_b}$.

This clause has the effect of eliminating all CNF models which correspond to vanishing terms, those containing the parameter $\theta_{b|a}$.

To this basic encoding we apply some optimizations:

- PRIMULA generated networks contain only binary variables. Therefore, 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 indicator clauses. For example, without the enhancement, variable B in Figure 1 generates Boolean variables I_{λ_b} and $I_{\lambda_{\bar{b}}}$ and the two clauses in Equation 1. With the optimization, B generates only a single Boolean variable I_B and no clauses. This optimization requires a corresponding modification to the decoding step as indicated below.
- Another enhancement results from the observation that the Boolean indicators and parameters corresponding to the same state of a network root variable are logically equivalent, making it possible to delete the parameter variables and the corresponding IP and PI clauses, which establish the equivalence. The Boolean indicator thus represents both an indicator and a parameter. For example, without the enhancement, parameter θ_a in Figure 1 generates one Boolean variable P_{θ_a} and two clauses, $I_A \Rightarrow P_{\theta_a}$ and $P_{\theta_a} \Rightarrow I_A$. With the enhancement, the variable and clauses are omitted. This optimization requires a corresponding modification to the decoding step as indicated below.
- Variables and clauses generated by parameters equal to 1 are redundant and therefore omitted.

Applying these enhancements allows us to create the CNF as follows. For each network variable X, we create propositional variable I_X. If X is not a root, then we perform three more steps. (1) For each network parameter $\theta_{x|\mathbf{u}}$ not equal to 0 or 1, create a propositional variable $P_{\theta_{x|\mathbf{u}}}$. (2) For each parameter $\theta_{x|u_1,u_2\ldots,u_n}$ equal to 0, create clause $\neg L_{U_1} \vee \neg L_{U_2} \vee \ldots \vee \neg L_{U_n} \vee \neg L_X$, where L_{U_i} is a literal over variable I_{U_i} whose sign is the the same as u_i, and similarly for L_X with respect to x. (3) For each parameter $\theta_{x|u_1,u_2,\ldots,u_n}$ not equal to 0 and not equal to 1, create clauses, $L_{U_1} \wedge L_{U_2} \wedge \ldots \wedge L_{U_n} \wedge L_x \Rightarrow P_{\theta_{x|u_1,\ldots,u_n}}$, $P_{\theta_{x|u_1,\ldots,u_n}} \Rightarrow L_{U_1}$, $P_{\theta_{x|u_1,\ldots,u_n}} \Rightarrow L_{U_2}$, $\ldots$, $P_{\theta_{x|u_1,\ldots,u_n}} \Rightarrow L_{U_n}$, $P_{\theta_{x|u_1,\ldots,u_n}} \Rightarrow L_X$, where L_{U_i} and L_X are as defined earlier. As an example, the CPT for variable B in Figure 1 generates the following clauses:

1st CPT row: $\neg I_A \vee \neg I_B$

3nd CPT 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 CPT row: $\neg I_A \wedge \neg I_B \Rightarrow P_{\theta_{\bar{b}|\bar{a}}}, P_{\theta_{\bar{b}|\bar{a}}} \Rightarrow \neg I_A, P_{\theta_{\bar{b}|\bar{a}}} \Rightarrow \neg I_B.$

Because PRIMULA generates networks with binary variables and nodes with at most three parents, this encoding leads to a CNF whose size is linear in the number of network variables. Table 1 depicts the size of CNF encodings for the relational models with which we experimented.

The special encoding used above calls for a slightly different decoding scheme for transforming a smooth d-DNNF into an arithmetic circuit. Specifically, if X is not a root, then literals I_X and $\neg I_X$ are replaced with evidence indicators λ_x and $\lambda_{\bar{x}}$, respectively. If X is a root, then literals I_X and $\neg I_X$ are replaced with $\lambda_x * \theta_x$ and $\lambda_{\bar{x}} * \theta_{\bar{x}}$, 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.

We close this section by pointing the reader to [25], which discusses more recent and sophisticated encodings to handle Bayesian networks with context–specific–independence [26], multi–valued variables, large CPTs, and lesser amounts of determinism.

5 Experimental Results

We ran our experiments on a 1.6GHz Pentium M with 2GB of RAM using a system available for download at *http://reasoning.cs.ucla.edu/ace*. Table 1 lists for each relational model a number of instances, and for each instance a number of measurements. First is the size and connectivity of the Bayesian network that PRIMULA generated. PRIMULA generates networks in formats acceptable by general purpose tools such as Hugin and Netica, but exact inference in these tools cannot handle most of these networks. Next is the number of variables and clauses in the CNF encodings. Clauses have at most five literals since the networks have at most three parents per node.

Table 1 shows additional findings. First, 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 randomly generated evidence sets. By evaluating and differentiating the circuit, one obtains marginals over all network families, in addition to other probabilities discussed in [2].

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 1 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.

Relational Model	Bayesian Network			CNF Encoding		Arithmetic Circuit			AC Time		JT
	Vars	CPT Params	Max Clst	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	0.029	1	8.25
04–08–03	1418	9802	26	1580	5252	71666	541356	19.0	0.052	1	57.48
05–08–03	1616	11278	32	1832	6125	149982	942167	19.8	0.093	1	
06–08–03	1814	12754	37	2084	6998	258228	1523888	20.5	0.152	1	
10–08–03	2606	18658	54	3092	10490	1293323	4315566	22.0	0.684	3	
03–08–04	2288	16008	31	2432	8292	186351	4859201	22.2	0.300	2	
04–08–04	2616	18488	39	2832	9712	932355	19457308	24.2	1.734	5	
03–08–05	3692	26186	40	3872	13453	1359391	55417639	25.7	4.325	10	
students											
P–S											
03–02	376	2616	25	618	2131	7927	37281	15.2	0.005	1	6.14
03–06	764	5512	50	1454	5147	110196	595737	19.2	0.056	1	
03–12	1346	9856	59	2708	9671	24219	113876	16.8	0.018	1	
04–08	1571	11566	72	3099	11099	95649	445410	18.8	0.053	2	
04–16	2827	21070	101	5859	21115	181166	815461	19.6	0.093	3	
05–10	2774	20688	128	5624	20279	630092	2531230	21.3	0.289	3	
05–20	5064	38168	148	10734	38889	1319834	5236257	22.3	1.844	7	
06–12	4445	33454	176	9209	33353	4586368	16936504	24.0	3.212	14	
06–24	8201	62302	233	17693	64325	9922233	36450231	25.1	12.966	33	
blockmap											
L–B											
05–01	700	4784	18	708	2412	1255	3364	11.7	0.005	1	2.70
05–02	855	5898	21	875	2999	1751	12306	13.6	0.006	1	6.36
05–03	1005	6972	23	1035	3561	2833	20636	14.3	0.007	1	27.39
10–01	5650	40070	52	5670	20083	10147	56998	15.8	0.014	1	
10–02	6252	44444	53	6292	22318	11978	309176	18.2	0.026	1	
10–03	6848	48758	52	6908	24529	17749	974817	19.9	0.058	2	
15–01	16497	116048	68	16525	58094	29347	224826	17.8	0.035	2	
15–02	17649	124298	70	17709	62299	33011	1798085	20.8	0.109	3	
15–03	18787	132436	68	18877	66443	47475	7643307	22.9	0.380	6	
20–01	39297	278138	90	39335	139164	69208	726787	19.5	0.094	6	
20–02	41337	292760	90	41413	146570	75299	6989375	22.7	0.376	10	
20–03	43356	307220	92	43476	153910	105602	40172434	25.3	2.453	30	
22–01	54318	386842	104	54360	193526	96424	1103074	20.1	0.141	10	
22–02	56873	405240	103	56957	202830	103980	11707536	23.5	0.823	20	
22–03	59404	423452	104	59536	212056	44136	76649302	26.2	4.665	61	
fr&sm											
N											
1	10	46	3	16	44	21	22	4.5	0.001	1	0.00
4	262	1600	13	430	1442	327	380	8.6	0.004	1	0.08
7	1225	7798	36	2023	7007	1,404	1,686	10.7	0.005	1	
19	3385	21880	70	5605	19655	3,686	4,211	12.0	0.007	1	
13	7228	47086	118	11986	42302	7,689	8,616	13.1	0.013	1	
16	13240	86656	172	21976	77864	13,919	15,394	13.9	0.016	1	
19	21907	143830	244	36385	129257	22,824	25,214	14.6	0.028	1	
22	33715	221848	316	56023	199397	34,877	38,210	15.2	0.040	2	
25	49150	323950	412	81700	291200	50,651	55,249	15.8	0.063	3	
28	68698	453376	528	114226	407582	70,541	76,273	16.2	0.082	3	
29	76212	503150	560	126730	452342	78,203	84,874	16.4	0.107	6	

Table 1
Relational Bayesian networks, their corresponding propositional instances, and the sizes of their CNF encodings.

Table 1 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. Yet 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.

Friends and smokers produces networks with particularly high connectivity. We mentioned previously that logical constraints in this model give rise to grounded Bayesian networks with evidence that applies to all queries. One might hope that classical pruning techniques—such as deleting leaf nodes not part of the query or evidence [27] and deleting edges exiting evidence nodes [28]—might reduce the connectivity of these networks, making them accessible to classical inference algorithms. This possibility is not realized though since all of the evidence occur on leaf nodes. However, we can use the method of [29] to place this evidence into the CNF encoding and compile *with* the evidence. In particular, if we know that network variable A corresponds to a logical constraint that must be *true*, then we simply add a unit clause λ_a to the CNF encoding. In fact, injecting these unit clauses into the CNF encoding prior to compilation has a critical effect on both compilation time and AC size, as most of these networks could not be compiled otherwise.

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 determinism in relational models, allowing us to reason efficiently with some relational models whose Primula–generated propositional instances contain thousands 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. It is also equally applicable and effective to any Bayesian network that exhibits similar properties (e.g., determinism), regardless of whether it is synthesized from a relational model.

Acknowledgments

This work has been partially supported by NSF grant IIS-9988543 and MURI grant N00014-00-1-0617.

References

[1] F. V. Jensen, S. Lauritzen, K. Olesen, Bayesian updating in recursive graphical models by local computation, Computational Statistics Quarterly 4 (1990) 269–282.

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

[3] F. Jensen, S. K. Andersen, Approximations in Bayesian belief universes for knowledge based systems, in: Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence (UAI), Cambridge, MA, 1990, pp. 162–169.

[4] J. Park, A. Darwiche, A differential semantics for jointree algorithms, Artificial Intelligence 156 (2004) 197–216.

[5] J. S. Breese, R. P. Goldman, M. P. Wellman, Introduction to the special section on knowledge-based construction of probabilistic and decision models, IEEE Transactions on Systems, Man, and Cybernetics 24 (11) (1994) 1577–1579.

[6] T. Sato, A statistical learning method for logic programs with distribution semantics, in: Proceedings of the 12th International Conference on Logic Programming (ICLP'95), 1995, pp. 715–729.

[7] S. Muggleton, Stochastic logic programs, in: L. de Raedt (Ed.), Advances in Inductive Logic Programming, IOS Press, 1996, pp. 254–264.

[8] K. Kersting, L. de Raedt, 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, 2001.

[9] K. B. Laskey, S. M. Mahoney, Network fragments: Representing knowledge for constructing probabilistic models, in: Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97), Morgan Kaufmann Publishers, San Francisco, CA, 1997, pp. 334–341.

[10] D. Koller, A. Pfeffer, Probabilistic frame-based systems, in: Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), 1998, pp. 580–587.

[11] N. Friedman, L. Getoor, D. Koller, A. Pfeffer, Learning probabilistic relational models, in: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 1999.

[12] M. Jaeger, Relational Bayesian networks, in: D. Geiger, P. P. Shenoy (Eds.), Proceedings of the 13th Conference of Uncertainty in Artificial Intelligence (UAI-13), Morgan Kaufmann, Providence, USA, 1997, pp. 266–273.

[13] H. Pasula, S. Russell, Approximate inference for first-order probabilistic languages, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2001, pp. 741–748.

[14] M. Richardson, P. Domingos, Markov logic networks, Tech. rep., Department of Computer Science and Engineering, University of Washington, Seattle, WA, to appear in the special issue of the Machine Learning Journal on Statistical Relational Learning and Multi-Relational Data Mining (in press).

[15] M. Jaeger, Complex probabilistic modeling with recursive relational Bayesian networks, Annals of Mathematics and Artificial Intelligence 32 (2001) 179–220.

[16] M. Jaeger, On the complexity of inference about probabilistic relational models, Artificial Intelligence 117 (2000) 297–308.

[17] D. Poole, First-order probabilistic inference, in: Proceedings of IJCAI-2003, 2003.

[18] R. de Salvo Braz, E. Amir, D. Roth, Lifted first-order probabilistic inference, in: Proceedings of the Nineteenth Int. Joint Conf. on Artificial Intelligence (IJCAI-05), 2005, pp. 1319–1325.

[19] A. Darwiche, A logical approach to factoring belief networks, in: Proceedings of KR, 2002, pp. 409–420.

[20] T. Sang, P. Beame, H. Kautz, Solving Bayesian networks by weighted model counting, in: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), Vol. 1, AAAI Press, 2005, pp. 475–482.

[21] J. Huang, A. Darwiche, DPLL with a trace: From sat to knowledge compilation, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), 2005, pp. 156–162.

[22] A. Darwiche, P. Marquis, A knowlege compilation map, Journal of Artificial Intelligence Research 17 (2002) 229–264.

[23] A. Darwiche, P. Marquis, Compiling propositional weighted bases, Artificial Intelligence 157 (1-2) (2004) 81–113.

[24] A. Darwiche, New advances in compiling CNF to decomposable negational normal form, Tech. Rep. D–141, Computer Science Department, UCLA, Los Angeles, Ca 90095, to appear in ECAI'04 (2004).

[25] M. Chavira, A. Darwiche, Compiling Bayesian networks with local structure, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), 2005, pp. 1306–1312.

[26] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context–specific independence in Bayesian networks, in: Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI), 1996, pp. 115–123.

[27] R. D. Shachter, Evaluating influence diagrams, Operations Research 34 (6) (1986) 871–882.

[28] S. Ross, Evidence absorption and propagation through evidence reversals, in: Proceedings of the 5th Annual Conference on Uncertainty in Artificial Intelligence (UAI-90), Elsevier Science Publishing Company, Inc., New York, NY, 1990.

[29] M. Chavira, D. Allen, A. Darwiche, Exploiting evidence in probabilistic inference, in: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI), 2005, pp. 112–119.