ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200392

Algebraic Circuits for Decision Theoretic Inference and Learning

Vincent Derkinderen and **Luc De Raedt**[1]

Abstract. While it is well known that arithmetic circuits can be used for efficient probabilistic inference, arithmetic circuits can also be used for other tasks. In this paper, we show how arithmetic circuits in a semiring setting (i.e., algebraic circuits) can solve decision theoretic inference tasks and a utility learning task under partial observability. The former involves finding the set of decisions that maximises the expected utility. We introduce two approaches for this, both applying algebraic circuits. The learning task involves learning unknown utility values from partially observed interpretations of which the total utility is given. We provide the necessary theory and also perform an experimental evaluation of the approaches.

1 INTRODUCTION

Probabilistic models are ubiquitous in artificial intelligence [16, 28]. Inference and learning for such models is computationally hard. Nonetheless, there has been a steady progress in developing tractable representations and algorithms for supporting a wide range of tasks. Especially techniques of knowledge compilation [7] have been instrumental in speeding up inference in Bayesian networks and Statistical Relational AI [28]. Knowledge compilation is concerned with compiling a logical theory into a representation, a circuit, that allows to perform particular operations in polynomial time. Various classes of circuits have been produced, giving rise to the celebrated knowledge compilation map [7]. Central to probabilistic inference is the problem of weighted model counting (WMC), an extension of the well known satisfiability (SAT) problem. In SAT, one determines whether a propositional theory has a model, while in model counting (#SAT) the problem is to determine how many models it has. In WMC, every model has a weight (the product of the weight of its literals) and the task is to compute the weighted sum of its models. While WMC in general is #P-complete, once the theory has been compiled to an appropriate circuit representation (e.g. a Sentential Decision Diagram [6]), it can be performed in time linear in the size of the circuit. This is especially beneficial for repeated inference where the circuit is constructed once and only the weights and operations change (e.g. in learning). This is a state of the art approach for inference in Bayesian networks.

It is well-known that a wide range of algorithms can be generalised using semirings, for instance, belief propagation with the sum-product and max-product algorithms. The key idea is to replace the traditional $+$ and $\times$ by semiring operations $\oplus$ and $\otimes$. This has inspired work on Algebraic Model Counting (AMC) [14, 15] where rather than using the standard probabilistic semiring, a range of other semirings are used to solve a wide range of inference tasks, including max-product, sensitivity analysis, gradient computation, and even weighted model integration [33]. These techniques have also found their way in algebraic extensions of probabilistic programming languages such as aProbLog [14], and have been used in new learning frameworks such as DeepProbLog [22].

In this paper, rather than considering the standard probabilistic setting, we consider decision theoretic extensions and investigate how we can adapt and apply AMC techniques to cope with some of the resulting inference and learning problems. One way of viewing this, is as the transition from standard Bayesian networks to influence diagrams or from a probabilistic programming language (such as ProbLog [11]) to its decision theoretic extension (such as DT-ProbLog [3]). An additional inference task for such an extension is to find the decisions that maximise the expected utility. We focus on the one-shot setting where all decisions are made before there are any observations. Maximising the expected utility requires three operations – max sum product, which already indicates that it is unclear how to apply standard AMC techniques to this problem.

The key contributions of this paper are that we introduce modified AMC techniques to solve two problems. The first problem is the maximisation of the expected utility, for which we introduce two techniques (Section 3). The first technique constrains the variable ordering and is based on earlier work on the Same-Decision Probability task [27]. Constraining the circuit can lead to larger circuits. We therefore also consider an alternative approach which instead views the circuit as a function with unknown values (decisions) that have to be optimised. We optimise these decisions using gradient ascent and show how AMC can be used to compute the gradients on the circuit. The second problem we solve is a novel learning task for utility values (Section 4). In this setting, each example of the data set represents an instance – a possible world – of which only some variables are observed. In addition, we also observe the total utility of each instance, a linear sum of the utilities of the instance variables. The combination of both partially observed interpretations and total utilities makes this a novel learning setting. One way of tackling this problem is to minimise a loss function using gradient descent. We evaluate this approach and show how AMC can be used to compute the gradients. To validate all contributions, we implement them as an extension of DT-ProbLog. The code and data is available at https://github.com/VincentDerk/Paper-AC-Decisions-Learning.

2 BACKGROUND

In this section, we first provide background on circuits and the type of tasks they can solve (Section 2.1 and 2.2). Second, we clarify the relation with knowledge compilation and define the class of circuits which we will use, Sentential Decision Diagrams (Section 2.3).

[1] KU Leuven, Belgium, email: name.surname@cs.kuleuven.be

2.1 Weighted model counting

Weighted model counting (WMC) consists of counting all models that satisfy a propositional logic theory, each with a particular weight. More formally, the task consists of a propositional logic theory T over a set of variables V. A model of T is a set of positive and negative literals, one for each variable in V, assigning truth values (v or $\neg v$) such that T is satisfied. We denote by $\mathcal{M}(T)$ the set of all models that satisfy T. The weight of $m \in \mathcal{M}(T)$ is determined by its literals $l \in m$ and a weight function $w(l)$ assigning a real value to each positive and negative literal.

Definition 1. The weighted model count of a propositional logic theory T and a weight function w is equal to

$$WMC(T, w) = \sum_{m \in \mathcal{M}(T)} \prod_{l \in m} w(l) \tag{1}$$

Example 1. Consider $T = A \vee B$ over $V = \{A, B\}$ and $w = \{a \mapsto 1, \neg a \mapsto 2, b \mapsto 3, \neg b \mapsto 4\}$, then

$$\begin{aligned} WMC(T, w) &= w(a)w(b) + w(a)w(\neg b) + w(\neg a)w(b) \\ &= 3 + 4 + 6 = 13 \end{aligned}$$

2.2 Algebraic model counting

Algebraic model counting (AMC) generalises WMC by also supporting non-real weights and generalising $\sum$ and $\prod$ to the operations of a commutative semiring, $\oplus$ and $\otimes$ respectively. In AMC, the weight function is referred to as labeling function α.

Definition 2. A commutative semiring S is an algebraic structure $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ where

- $\mathcal{A}$ defines the domain of the values,
- $\oplus$ and $\otimes$ are associative, commutative binary operations over $\mathcal{A}$,
- $\otimes$ distributes over $\oplus$,
- $e^{\otimes} \in \mathcal{A}$ and for all $a \in \mathcal{A}$: $e^{\otimes} \otimes a = a$,
- $e^{\oplus} \in \mathcal{A}$ and for all $a \in A$: $e^{\oplus} \oplus a = a$ and $e^{\oplus} \otimes a = e^{\oplus}$.

Definition 3. Algebraic model counting [15] is the task of computing Equation 2 given a propositional logic theory T over variables V, a commutative semiring $S = (A, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ and a labeling function α mapping literals to elements of $\mathcal{A}$.

$$AMC(T, S, \alpha) = \bigoplus_{m \in \mathcal{M}(T)} \bigotimes_{l \in m} \alpha(l) \tag{2}$$

Example 2. Consider the semiring $S = (\mathbb{R}, max, \times, 0, 1)$, $T = A \vee B$ and $\alpha = \{a \mapsto 1, \neg a \mapsto 2, b \mapsto 3, \neg b \mapsto 4\}$ then $AMC(S, T, \alpha) = 6$, the highest weight out of all models of T.

Many problems can be transformed into an AMC problem by defining the appropriate semiring and labeling function, examples include sensitivity analysis and computing gradients [15].

2.3 Knowledge compilation: SDDs

A state-of-the-art approach to compute the AMC involves compiling the theory T into a representation that allows for tractable weighted (and algebraic) model counting. This process is studied in the domain of knowledge compilation [7]. One representation class that can be used for this is the Sentential Decision Diagrams class (Figure 1) [6].

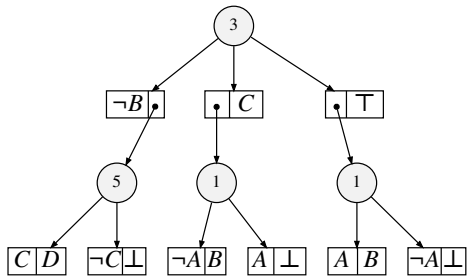

Figure 1. An SDD representing theory $(A \wedge B) \vee (C \wedge D) \vee (B \wedge C)$. A circle is an or-node, a paired-box is an and-node consisting of two children.

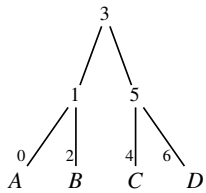

Figure 2. The vtree used for the SDD in Figure 1.

A Sentential Decision Diagram (SDD) represents a logic theory and is either a constant ($\top$ or $\bot$), a literal or a decomposition node $\{(p_1, s_1), \ldots, (p_n, s_n)\}$. The latter represents $\bigvee_{i=1}^{n} p_i \wedge s_i$ with p_i and s_i both SDDs. A decomposition node is an or-node which partitions the theory into disjoint children (p_i, s_i) called the elements of the decomposition node. Each element is graphically represented as a paired box where the left and right box are respectively the prime p_i and sub s_i. The pair (p_i, s_i) represents a conjunction of both (and-node). s_i represents the parent theory conditioned on p_i (Example 3).

Example 3. The root r in Figure 1 represents the theory $T = (A \wedge B) \vee (C \wedge D) \vee (B \wedge C)$. Call the children of r, from left to right, r_1, r_2 and r_3 and the theory they represent $\langle r_1 \rangle, \langle r_2 \rangle$ and $\langle r_3 \rangle$. The prime of r_1 represents $\neg B$ and the sub of r_1 represents T conditioned on $\neg B$ which equals $C \wedge D$. The conjunction of both the prime and sub of r_1 forms the theory $\langle r_1 \rangle = \neg B \wedge C \wedge D$. The theory of r is the disjunction of each of its children: $\langle r \rangle = T = \langle r_1 \rangle \vee \langle r_2 \rangle \vee \langle r_3 \rangle$.

A vtree is a full binary tree where each SDD variable appears in a leaf node (Figure 2). Denote with v^l and v^r the left and right subtree of vtree v. A vtree guides the construction of an SDD by determining the variables present in the primes and subs of each SDD node. When node $\{(p_1, s_1), \ldots, (p_n, s_n)\}$ respects vtree node v, the variables in each p_i and s_i are determined by respectively the variables in v^l and v^r, and each p_i (s_i) respects v^l (v^r). Graphically, the number in the decomposition node refers to the vtree node that it respects (Figure 1 and 2). The following is the formal SDD definition introduced by Darwiche [6]. The definition uses $\langle \alpha \rangle$ to denote the boolean function represented by the SDD α.

Definition 4. α is an SDD that respects vtree v iff:

- $\alpha = \bot$ or $\alpha = \top$.
  Semantics: $\langle \bot \rangle = false$ and $\langle \top \rangle = true$.
- $\alpha = X$ or $\alpha = \neg X$ and v contains variable X.
  Semantics: $\langle X \rangle = X$ and $\langle \neg X \rangle = \neg X$.

- $\alpha = \{(p_1, s_1), \dots (p_n, s_n)\}$, v is internal, $p_1, \dots, p_n$ are SDDs that respect the subtrees of v^l, $s_1, \dots, s_n$ are SDDs that respect the subtrees of v^r, and $\langle p_1 \rangle, \dots, \langle p_n \rangle$ is a partition.
  Semantics: $\langle \alpha \rangle = \bigvee_{i=1}^{n} \langle p_i \rangle \wedge \langle s_i \rangle$.

Example 4. We denote the root node in Figure 1 as r and the root of the vtree in Figure 2 as v. The primes of r only involve A and B since v^l only contains A and B. The subs of r only involve C and D since v^r only contains C and D. We say r respects v (node label 3).

When we replace the or-nodes (circle) and and-nodes (paired box) in an SDD with respectively $+$ and $\times$, we obtain what is called an arithmetic circuit. This circuit can be evaluated bottom-up to obtain the WMC at the root node [5]. Furthermore, when replacing the $+$ and $\times$ with the more general $\oplus$ and $\otimes$ operations, and the leaf values with semiring elements, we obtain what we refer to as an algebraic circuit. The latter circuit can be used to compute the AMC [15]. In a sense, this circuit represents the computations required to obtain the AMC in a tractable manner, by exploiting the theory T and the semiring properties of the $\oplus$ and $\otimes$ operations.

An SDD is an **X**-constrained SDD when the vtree satisfies certain conditions. The following definitions are taken from the work that introduced **X**-constrained SDDs [27].

Definition 5. A vtree node v is **X**-constrained iff v appears on the right-most path of the vtree and **X** is the set of variables outside v. A vtree is **X**-constrained iff it has an **X**-constrained node.

Definition 6. An SDD is **X**-constrained iff it respects an **X**-constrained vtree. An SDD node is **X**-constrained iff it respects an **X**-constrained vtree node.

Example 5. The vtree in Figure 2 is both $\{A, B\}$- and $\{A, B, C\}$-constrained because of respectively node 5 and 6. The SDD in Figure 1 is hence both $\{A, B\}$- and $\{A, B, C\}$-constrained.

We use the SDD package[2] to implement our work. The package supports both SDD and **X**-constrained SDDs.

3 MAXIMISING DECISIONS

We first explain how AMC can be used to compute a probability and an expected utility.

Probabilities Weighted model counting (and AMC) can be used to answer probabilistic queries and is a basis for inference in Bayesian networks and ProbLog programs. Probabilistic inference can be cast into a WMC (and the more general AMC) task as follows. The probability of a model is equal to multiplying the weights of the positive and negative literals of that model $\prod_{l \in m} w(l)$. By summing up all the models where query q holds, the WMC yields the probability of q[3]: $\sum_{m \in \mathcal{M}(T \wedge q)} \prod_{l \in m} w(l)$. This corresponds exactly to the weighted model counting task (and AMC).

Expected utility When each variable is associated with both a probability and a utility, then the expected utility can be computed using AMC, similar to the probability approach. We denote with $u(m)$ and $p(m)$ the utility and probability of a model m and with u_v ($u_{\neg v}$) the utility obtained from variable v being true (false). The expected utility of m is defined as $eu(m) = p(m) \times u(m)$. The expected utility of a theory T, $eu(T)$, is defined in Equation 3 and can be computed using the AMC framework.

$$eu(T) = \sum_{m \in \mathcal{M}(T)} \underbrace{(\overbrace{\prod_{l \in m} p_l}^{p(m)})(\overbrace{\sum_{l \in m} u_l}^{u(m)})}_{eu(m)} \tag{3}$$

Define the labeling function α such that $\alpha(v) = (p_v, p_v \times u_v)$, then the expected utility semiring can be used to compute $eu(T)$.

Definition 7. The expected utility semiring $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ with

- $\mathcal{A} = \{(p, eu) | p \in \mathbb{R}_{\geq 0}, eu \in \mathbb{R}\}$
- $(p_1, eu_1) \oplus (p_2, eu_2) = (p_1 + p_2, eu_1 + eu_2)$
- $(p_1, eu_1) \otimes (p_2, eu_2) = (p_1 p_2, p_1 eu_2 + p_2 eu_1)$
- $e^{\oplus} = (0, 0)$ and $e^{\otimes} = (1, 0)$

is an instance of the expectation semiring [10] and can be used to compute the expected utility of a theory.

Maximising expected utility A more complicated problem arises when we introduce decision variables and seek the assignment of truth values to those decisions that maximises the expected utility. We refer to this problem as the maximum expected utility (MEU) problem. We focus on its one-shot decision setting where all decisions are made before observing any stochastic variable.

Definition 8. The maximum expected utility (MEU) problem consists of a propositional logic theory T over a set of decision variables D and a set of stochastic variables S, each associated with a probability and utility. For decisions, the probability is either 0 or 1. For the first solving approach, we set $p(d) = 1 = p(\neg d)$ for all $d \in D$ such that the probability of a model is not impacted by its decision. We use D_v to refer to an assignment of a truth value for each decision variable $d \in D$ and $T[D : D_v]$ to refer to the theory T where the truth values of the variables D are set to D_v. The MEU task consists of finding the best truth assignment D_v such that the expected utility is maximised (Equation 4).

$$\underset{D_v}{argmax} \sum_{m \in \mathcal{M}(T[D:D_v])} (\prod_{l \in m} p_l)(\sum_{l \in m} u_l) \tag{4}$$

While this task consists of three operations (max, sum and product), a semiring is a structure of only two. It is therefore not obvious on how to apply AMC.

Example 6. Consider a very small problem, deciding whether to use machine A. Using A has a cost of -3 but, when there is no failure, it also yields a reward of 4.

$$\begin{gathered}(profit \wedge useA \wedge \neg failure) \vee \\ (\neg profit \wedge \neg useA) \vee \\ (\neg profit \wedge failure)\end{gathered}$$
$$p_{useA} = 1.0, \quad p_{\neg useA} = 1.0, \quad u_{useA} = -3, \quad u_{\neg useA} = 0$$
$$p_{profit} = 1.0, \quad p_{\neg profit} = 1.0, \quad u_{profit} = 4, \quad u_{\neg profit} = 0$$
$$p_{failure} = 0.6, \quad p_{\neg failure} = 0.4, \quad u_{failure} = u_{\neg failure} = 0$$

3.1 Constrained algebraic circuit

To obtain the expected utility of a set of decision assignments D_v, we need to sum all the models with the same set of decisions. This implies that there is an ordering that the circuit needs to adhere to in order to compare decisions in a valid manner. More specifically, when

[2] The SDD package is available at http://reasoning.cs.ucla.edu/sdd/.

[3] When computing the conditional probability or supporting constraints, a normalisation is required $WMC(T \wedge q)/WMC(T)$.

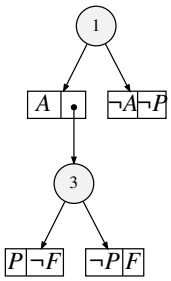

Figure 3. An {A}-Constrained SDD modelling Example 6. The variables are abbreviated: $A = useA$, $P = profit$ and $F = failure$.

using SDDs, the circuit must first condition on the decision variables before considering the rest. **X**-constrained SDDs with $\mathbf{X} = D$ have exactly this property (Figure 3). This can be seen as follows. By definition, the D-constrained SDD nodes in a D-constrained SDD represent the whole theory T conditioned on an assignment for each decision, $T[D : D_v]$. Such a node represents a disjunction of all the models with the same decisions. The circuit will thus first combine models with the same set of decisions before combining (comparing) with models of other decisions, as was required.

The vtree can be used to determine whether an or-node of the **X**-constrained SDD represents a summation or a maximisation. Another approach that can be used is to store the decision information in the semiring elements in the form of a decision set L and performing maximisation when the sets of decisions are different. To stay close to the algebraic framework, we choose the latter approach and use the following structure $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$, dynamically defining the $\oplus$-operation:

$$\{\alpha(v) = (p_v, p_v \times u_v, L_v) | v \in D \cup S\} \subset A \tag{5}$$

with $L_v = \{v\}$ and $L_{\neg v} = \{\neg v\}$ or $L_v = \emptyset = L_{\neg v}$ depending on whether v is a decision.

$$a \oplus b = \begin{cases} max(a,b), & \text{if } L_a \neq L_b \\ (p_a + p_b, eu_a + eu_b, L_a), & \text{otherwise} \end{cases} \tag{6}$$

$$a \otimes b = (p_a p_b, p_a eu_b + p_b eu_a, L_a \cup L_b) \tag{7}$$

$$e^{\oplus} = (0, 0, D \cup \neg D) \tag{8}$$

$$e^{\otimes} = (1, 0, \emptyset) \tag{9}$$

$$max(a,b) = \begin{cases} a, & \text{if } b = e^{\oplus} \\ b, & \text{else if } a = e^{\oplus} \\ a, & \text{else if } \frac{eu_a}{p_a} \geq \frac{eu_b}{p_b} \\ b, & \text{otherwise} \end{cases} \tag{10}$$

This structure (Equation 5 to 10) extends the expectation semiring to keep track of the decision sets L and to perform max when required. For a set of decision assignments, the probability of all models must sum up to one. This is not necessarily the case when constraints are present and we therefore normalise when comparing expected utilities: $\frac{eu_a}{p_a} \geq \frac{eu_b}{p_b}$ (Equation 10). Additional cases can be added to prevent division by 0 when $p_a = 0$ or $p_b = 0$. When such a case occurs, the other value must be chosen. Note that this structure is not a semiring as the associativity property is only satisfied within the **X**-constrained context. This is expected and the reason we require an **X**-constrained SDD. If associativity was satisfied in general, then any SDD would have been sufficient.

3.2 Unconstrained algebraic circuit

The vtree which yields the smallest circuit is not necessarily X-constrained. This means that constraining the vtree can lead to larger circuits. To avoid this problem, we introduce another approach which does the maximisation outside of the circuit and does not rely on a constrained ordering. Instead, it treats the circuit as a function where the decision values are unknown and have to be chosen such that the output of the function, the expected utility, is maximised. This works as follows. When the probability of each decision variable is set to either 0 or 1 (and its negation to 1 or 0), the output of the circuit is the expected utility of that decision set. Hence, optimising these parameters will maximise the expected utility. We solve this problem via gradient ascent and compute the gradient using the circuit and AMC. While the constrained approach is exact and ensures an optimal decision, the unconstrained approach does not ensure this when optimising with gradient ascent.

Implementation To keep the probability of a decision d in $[0, 1]$, we use the sigmoid function: $p_d = \sigma(z) = \frac{1}{1+e^{-z}}$ for which $\sigma'(z) = \sigma(z)(1 - \sigma(z))$. We optimise the function represented by the algebraic circuit (Equation 11) via gradient ascent and use algebraic circuits to obtain the required gradients (Equation 12 to 16).

$$MEU = \sum_{m \in \mathcal{M}(T)} U(m) \times P(m) \tag{11}$$

$$\frac{\partial MEU}{\partial d_i} = \sum_{m \in \mathcal{M}(T)} U(m) \times \frac{\partial P(m)}{\partial d_i} \tag{12}$$

$P(m)$ is a combination of stochastic variables S and decision variables D that are in m (positive literals) and not in m (negative literals, Equation 13),

$$P(m) = \prod_{\substack{s \in m, \\ s \in S}} p_s \prod_{\substack{s \notin m, \\ s \in S}} p_{\neg s} \prod_{\substack{d \in m, \\ d \in D}} p_d \prod_{\substack{d \notin m, \\ d \in D}} (1 - p_d) \tag{13}$$

thus, if $d_i \in m$

$$\frac{\partial P(m)}{\partial d_i} = (1 - \sigma(d_i))P(m) \tag{14}$$

and if $d_i \notin m$

$$\frac{\partial P(m)}{\partial d_i} = -(1 - \sigma(d_i))P(m) \tag{15}$$

This means the gradient with respect to d_i can be obtained by computing the sigmoid of d_i, the expected utility where d_i is true and where d_i is false (Equation 16). The last two each result in one (parallel) circuit evaluation.

$$\begin{aligned} \frac{\partial MEU}{\partial d_i} &= \Big(\sum_{\substack{m \in \mathcal{M}(T), \\ d_i \in m}} U(m) \frac{\partial P(m)}{\partial d_i}\Big) \\ &\quad + \Big(\sum_{\substack{m \in \mathcal{M}(T), \\ d_i \notin m}} U(m) \frac{\partial P(m)}{\partial d_i}\Big) \\ &= (1 - \sigma(d_i))\Big(\sum_{m \in \mathcal{M}(T \wedge d_i)} EU(m)\Big) \\ &\quad + (\sigma(d_i) - 1)\Big(\sum_{m \in \mathcal{M}(T \wedge \neg d_i)} EU(m)\Big) \end{aligned} \tag{16}$$

The decision values we find with this approach are in the range of [0,1] and affect the weight with which models contribute to the expected utility. The optimal value will always be either 0 or 1. However, insufficient measures to prevent local minima or insufficient time can cause decision values to be far from 0 or 1. To obtain the best decisions from the found values, several approaches can be investigated. Examples include rounding to the nearest integer, evaluating different decisions for which the value is far from 0 and 1, treating the results as a stochastic policy, etc. In our implementation, we choose to set each final parameter to the nearest integer in order to have a deterministic policy and use random restarts to counter local minima.

3.3 Experiments

In our experiments, rather than using the low level AMC encodings directly, we use the higher-level probabilistic programming language ProbLog to specify models and queries. These models are then compiled into AMC problems using the mechanics of aProbLog [14]. This lead to new abilities for ProbLog[4]. We implement our approaches for DT-ProbLog by using the mechanics of aProbLog [14]. The data set is constructed as follows. Well known Bayesian networks [30], Survey [31], Asia [18] and Earthquake [17] are first compiled into ProbLog programs using ProbLog's existing conversion script. Next, we add decisions to the programs by converting each parent node of the Bayesian network with two possible values into a decision. If this results in less than four decisions, any node of the Bayesian network not yet considered has a chance of 0.5 to introduce a new decision. Each value of the node has an equal probability of being affected by this new decision. Finally, we introduce utilities using two different approaches. The first approach considers each term t and adds a utility value for t with a probability of 0.8 and for $\neg t$ with a probability of 0.3. The utility values themselves are uniformly sampled from $[-50, 50]$. The second approach instead introduces five new separate terms with a positive and negative utility, and for each new term samples five interpretations from the program. The samples serve as rules for the new term to become satisfied. The second approach happens before adding the decisions. The first approach happens afterwards, to allow decisions to also have utilities. Using this process, we construct 60 DT-ProbLog models (20 for Asia, Earthquake and Survey), half of them constructed with the first utility approach and half of them with the second. The number of rules in the resulting models ranges from 38 up to 108, the number of utilities from 7 to 23 and the number of decisions from 1 to 6. The memory consumption of larger networks (e.g. Sachs [30]) was too high to consider here. This is due to the rather naive standard encoding of Bayesian networks as ProbLog programs, which could be optimized using more compact encodings, a better vtree heuristic or when configuring the SDD package more optimally. Our experiments are designed to answer two questions.

Q1) Does the unconstrained approach provide optimal solutions? We experimented on the 60 DT-ProbLog models comparing both approaches. For 85% of the models, the difference was less than 0.1. The average difference over the experiments is 1.472 and the average relative difference is 0.057. We conclude that overall, the unconstrained approach provides promising results. Further investigation into problems of a larger size would be interesting.

Q2) How does the constrained ordering impact the circuit size, compile- (CT) and runtime (RT) compared to the unconstrained approach? We report on the average compile-, runtime and SDD size in Table 1. It is clear that constraining the circuit can lead to larger circuits[5]. However, the unconstrained approach trades compile time for more evaluation time and currently becomes slower than the constrained approach. This part in the implementation can still be optimised, e.g. by using a better random restart configuration.

Table 1. Statistics on the executions for the constrained (c) and unconstrained (u) approach for each dataset D (earthquake (e), asia (a) and survey (s)) and for all datasets combined (g). The average compile time (CT), run time (RT) and SDD size is provided. The run time includes the compilation time.

D	Appr.	avg. CT (s)	avg. RT (s)	avg. SDD Size (# nodes)
g	c	4.0	4.3	1 480 603
	u	2.5	41.0	1 055 494
e	c	0.0	0.0	2065
	u	0.0	1.1	2244
a	c	0.0	0.1	12539
	u	0.0	3.3	5539
s	c	11.9	12.7	4 427 204
	u	7.5	118.6	3 158 698

We conclude that to scale to larger problem sizes, more work is required on the used encoding, vtree heuristics, random restart configuration, etc. Regardless, we have shown that AMC and algebraic circuits provide an expressive framework that can also solve decision theoretic tasks.

4 LEARNING UTILITY PARAMETERS

Several techniques have already been introduced to learn the probability parameters in ProbLog [12, 13, 22]. The most recent addition jointly learns the parameters of probabilistic facts and those of neural networks by optimising a loss-function using gradient descent [22]. This approach integrates well with ProbLog's inference as the gradient can also be computed using the algebraic circuit. We will use the same approach and define a loss function that we use as an approximate signal, allowing us to learn the utility parameters for each variable.

Setting The input of our learning task is based on a set of interpretations $\{m_1, \ldots, m_M\}$ called examples. These examples we only observe partially Q = $\{q_1, \ldots, q_M\}$. For each of the interpretations m_j, we also observe the total utility $\tilde{u}_j$, $\tilde{U} = \{\tilde{u}_1, \ldots, \tilde{u}_M\}$. The output of this learning task consists of the positive and negative utility, respectively $u_{i,p}$ and $u_{i,n}$, associated with each fact f_i. We focus on the utilities here and assume the probability parameter of each probabilistic fact is already known. This can be relaxed in two ways. On the one hand, we can learn the probabilities first, ignoring the utility values, after which we are in our described setting. On the other hand, the assumption can be relaxed when the loss function is extended with a component concerning the likelihood of the observations. Though it could be interesting to compare the performance of these different approaches empirically, this is left for further work. We also assume that of each example, all decisions are observed. Our approach works for both one-shot and sequential decision problems.

Approach To learn the utility parameters, we minimise the following loss function:

[4] ProbLog is available at https://dtai.cs.kuleuven.be/problog/

[5] The reported circuit sizes were obtained from the SDD package and can include dead nodes left over from the construction process.

$$MSE(Q,\tilde{U},T) = \frac{1}{M}\sum_{j=1}^{M}(ceu(q_j,T) - \tilde{u}_j)^2 \tag{17}$$

$$ceu(q_j,T) = \sum_{m\in\mathcal{M}(T)} P(m|q_j).u(m) \tag{18}$$

The intuition behind this equation is that we minimise the difference between the utility we expect $ceu(q_j)$ and the utility that was actually observed $\tilde{u}_i$. The former is defined by $u(m)$ and $P(m|q_j)$. Note that the assumption that all decisions are fully observed, simplifies the calculation of $P(m|q_j)$ and is justified as long as the decisions are not made by some unknown third party. Our approach also works for partially observed decisions when each decision has an associated probability. When assuming optimal behavior, the calculation of $P(m|q_j)$ and our minimisation approach becomes more complex.

To optimise Equation 17, we employ a gradient descent approach and use an algebraic circuit to compute the gradient $\frac{\partial MSE(Q,\tilde{U},T)}{\partial u_{i,p}}$ (Equation 19).

$$\frac{2}{M}\sum_{j=1}^{M}\underbrace{(ceu(q_j,T)}_{Part1} - \tilde{u}_j)\underbrace{\sum_{m\in\mathcal{M}(T)}\delta_{i,m,p}P(m|q_j)}_{Part2} \tag{19}$$

$$\delta_{i,m,p} = \begin{cases} 1 & \text{if } f_i \in m, \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

The gradient for the negative utility parameter $u_{i,n}$ is similar to Equation 19 except that we use $\delta_{i,m,n}$ instead of $\delta_{i,m,p}$.

$$\delta_{i,m,n} = \begin{cases} 1 & \text{if } f_i \notin m, \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

Part 1 of Equation 19 can be computed using the expected utility semiring querying for q_j conditioned on q_j. Part 2 of that equation can be computed using the probability semiring querying for $q_j \wedge f_i$ (or $q_j \wedge \neg f_i$ for the $u_{i,n}$ case) conditioned on q_j.

Experiments The correct DT-ProbLog programs are constructed using the process described for maximisation (Section 3). To simplify the experiment setup, we do not add decisions to the programs. To construct the partially observed examples, we sample from the programs and leave out each observed term with a probability of p_{drop}. The input model of the learning task is the original model with for each utility term, a chance of 0.5 that it is made unknown.

To evaluate our approach we answer the following questions. **Q1)** Is the MSE loss function a good indicator when our aim is to 1) predict the utility of an interpretation, 2) recover the correct values or 3) to make good decisions. **Q2)** How does the partial observability affect the results? We consider three metrics to answer both questions, mean squared sampled error (MSSE), mean relative error (MRE) and relative regret. The MSSE (Q1.1, Q2) is closest to what we are optimising and is evaluated by sampling $n_s = 100$ interpretations, comparing the total utility of the interpretation t_c with the total utility of the learned values t_l: $\frac{1}{n_s}\sum_{i=1}^{n_s}(t_c - t_l)^2$. The MRE (Q1.2) is used to compare how close the learned utility values x_i are to the actual values $\tilde{x}_i$, using the relative error to normalise for large (small) $\tilde{x}_i$, When there are n_v learned values, $MRE = \frac{1}{n_v}\sum_{i=0}^{n_v}|\frac{x_i-\tilde{x}_i}{\tilde{x}_i}|$. If we use the learned model for decision making, then the regret metric is more interesting, but also more complex to compute. The regret (Q1.3) is based on the utility to expect when taking the decisions based on the learned model. Denote D_l as the optimal decisions according to the learned model, D_t as the true optimal decisions and $eu(D)$ as the expected utility when taking decisions D. Then the relative regret is defined as $|\frac{eu(D_t)-eu(D_l)}{eu(D_t)}|$ and computed using the maximisation approaches described in this paper. This metric requires decisions which we add as described for the maximisation approach (Section 3).

Q1) We have tested the learning approach on five different Survey, Earthquake and Asia networks, each for varying values of p_{drop}[6]. Each experiment was given 80 epochs to converge and 150 partially observed examples to train on. Figure 4 shows that while optimising our loss function, the MSSE and MRE successfully decrease as well. This suggests the MSE can be used as an indicator to optimise MSSE and MRE. It is possible that when optimising too long, the MSSE and MRE can increase again due to overfitting to MSE. This is more noticeable for MRE (Figure 5) than for MSSE. The relative regret is low, even for high p_{drop} (Figure 6). **Q2)** We investigate the effect of p_{drop} on the MSSE (Figure 7). As expected, it generally becomes harder to learn with an increased p_{drop}.

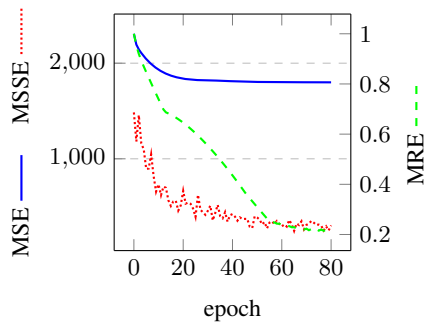

Figure 4. The learning progress of a Survey network with $p_{drop} = 0.8$.

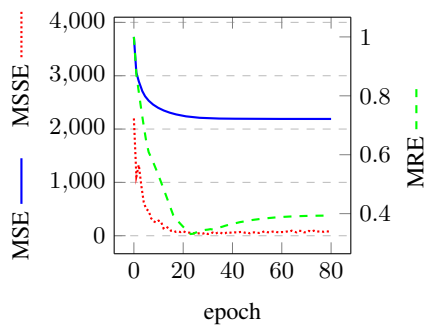

Figure 5. The learning progress of a Survey network with $p_{drop} = 0.7$.

We conclude that AMC techniques can be adapted to perform utility learning. The performed experiments show good results. To investigate larger problems with more parameters, we first need improvements to obtain smaller circuits.

[6] Due to a non-deterministic ordering originating in the ProbLog database, an increase in p_{drop} can cause previously unobserved terms to become observed. It is however still impossible for more terms to become observed.

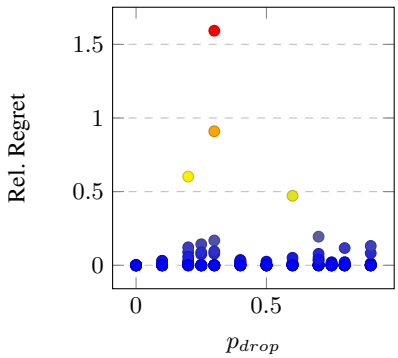

Figure 6. Relative regret for 180 survey networks.

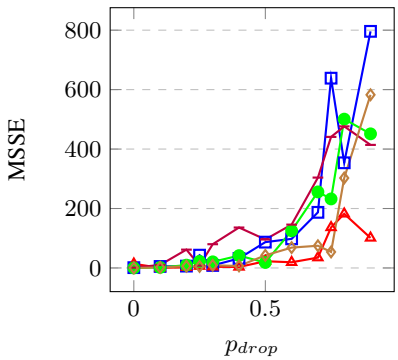

Figure 7. The MSSE for five different survey networks, over different degrees of partial observability (p_{drop}).

5 RELATED WORK

Maximisation Our maximisation approaches address the same problem as DT-ProbLog [3]. Their approach consists of manipulating Binary Decision Diagrams (BDD) while we use a more general AMC approach applied to SDDs. Our approaches also retain the circuit so that it can be used for other tasks (e.g. learning utilities) or an extension of this task (e.g. stochastic constraints). Finally, DT-ProbLog did not yet consider a learning setting. Sum-Product-Max Networks (SPMN) and Decision Circuits (DC) have a structure similar to our constrained circuit approach [2, 24]. They determine the operations of each node during the construction process. Our approach emphasises the power of an algebraic circuit and AMC, dynamically defining the operations. Due to the higher level definition, we can also reuse existing compilation tools. This AMC approach provides more flexibility towards extended or different tasks [20]. Finally, we have also introduced an approach that does not constrain the variable ordering and a learning task. A DPLL approach can also maximise the expected utility [1, 21]. This is related to our approach as the traces of DPLL can be used to form an arithmetic circuit. The advantage of a circuit is that it can be reused, significantly reducing the cost of re-evaluating the theory with different input weights. This is especially beneficial for the utility learning approach which requires multiple evaluations to obtain gradients. Another example of a task that requires multiple evaluations is sensitivity analysis [5]. AND-OR graphs are related to DPLL and algebraic circuits [8, 9]. Work in that domain is often used in a probabilistic setting but can also be applied to the maximum expected utility problem [19, 23]. Those approaches often start from an influence diagram while we start from an expressive DT-ProbLog program. Furthermore, as main difference to the work on AND-OR graphs, our contribution includes the application of algebraic circuits to utility learning and an unconstrained circuit approach. This has not been considered by those other approaches.

Utility learning There is a lot of work already performed in the context of utility learning. However, to the best of our knowledge there is none that is situated in our setting, that is, with partially observed interpretations and the total utility of that interpretation. In terms of data structure, the work on Sum-Product-Max networks [25] is the most similar but it considers a fully observed setting. Markov Decision Processes and Influence Diagrams (also known as Bayesian Decision Networks) are two alternatives for modelling a decision problem. We are not aware of any work for those models that considers our setting. In general, utility information is not provided and instead obtained indirectly for example by preference elicitation [4, 29] or based on interpretations with optimal behavior [26, 32]. The latter is the case in the domain of inverse reinforcement learning [26] where an unknown utility function, for example of an MDP, is learned from examples containing optimal behavior.

6 CONCLUSION

Algebraic circuits are versatile structures. We have shown at the level of AMC how maximising the expected utility and utility learning can be solved. Because we defined this at the high-level of AMC, we were able to reuse the algebraic circuit mechanics of the existing probabilistic languages (aProbLog and DTProbLog) without adding new constructs to them. We have shown two approaches for the maximisation problem and we introduced a novel learning setting where unknown utility values are learned from partially observed interpretations with observed utilities. This learning task can be tackled by a gradient descent approach, using algebraic circuits to compute the gradients. The circuit size and compilation time rapidly increase for larger problems and is currently an obstacle for scaling our approaches. In future work, we plan to investigate ways of improving this by adapting our methods (e.g. encodings) or improving knowledge compilation tools. Finally, we plan to extend the maximisation approaches to sequential problems.

ACKNOWLEDGEMENTS

VD is an SB PhD fellow at FWO (1SA5520N). This research was also partially funded by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme, from the Research Foundation - Flanders under the Data-driven logistics project (FWO-S007318N) and the VeriLearn project (EOS No. 30992574) and from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 694980) SYNTH: Synthesising Inductive Data Models.

REFERENCES

[1] Udi Apsel and Ronen I Brafman, 'Lifted meu by weighted model counting', in *Twenty-Sixth AAAI Conference on Artificial Intelligence*, (2012).

[2] Debarun Bhattacharjya and Ross D Shachter, 'Evaluating influence diagrams with decision circuits', *arXiv preprint arXiv:1206.5257*, (2012).

[3] Guy Van den Broeck, Ingo Thon, Martijn van Otterlo, and Luc De Raedt, 'Dtproblog: A decision-theoretic probabilistic prolog', in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pp. 1217–1222. AAAI Press, (2010).

[4] Urszula Chajewska, Daphne Koller, and Ronald Parr, 'Making rational decisions using adaptive utility elicitation', in *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, eds., Henry A. Kautz and Bruce W. Porter, pp. 363–369. AAAI Press / The MIT Press, (2000).

[5] Adnan Darwiche, 'A differential approach to inference in bayesian networks', in *UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, California, USA, June 30 - July 3, 2000*, eds., Craig Boutilier and Moisés Goldszmidt, pp. 123–132. Morgan Kaufmann, (2000).

[6] Adnan Darwiche, 'Sdd: A new canonical representation of propositional knowledge bases', in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Two*, IJCAI'11, pp. 819–826. AAAI Press, (2011).

[7] Adnan Darwiche and Pierre Marquis, 'A knowledge compilation map', *J. Artif. Int. Res.*, **17**(1), 229–264, (September 2002).

[8] Rina Dechter, 'Bucket elimination: A unifying framework for reasoning', *Artif. Intell.*, **113**(1-2), 41–85, (1999).

[9] Rina Dechter and Robert Mateescu, 'And/or search spaces for graphical models', *Artif. Intell.*, **171**(2-3), 73–106, (February 2007).

[10] Jason Eisner, 'Parameter estimation for probabilistic finite-state transducers', in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 1–8, (2002).

[11] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Sht. Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt, 'Inference and learning in probabilistic logic programs using weighted boolean formulas', *TPLP*, **15**(3), 358–401, (2015).

[12] Bernd Gutmann, Angelika Kimmig, Kristian Kersting, and Luc De Raedt, 'Parameter learning in probabilistic databases: A least squares approach', in *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I*, eds., Walter Daelemans, Bart Goethals, and Katharina Morik, volume 5211 of *Lecture Notes in Computer Science*, pp. 473–488. Springer, (2008).

[13] Bernd Gutmann, Ingo Thon, and Luc De Raedt, 'Learning the parameters of probabilistic logic programs from interpretations', in *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I*, eds., Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, volume 6911 of *Lecture Notes in Computer Science*, pp. 581–596. Springer, (2011).

[14] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt, 'An algebraic prolog for reasoning about possible worlds', in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, eds., Wolfram Burgard and Dan Roth. AAAI Press, (2011).

[15] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt, 'Algebraic model counting', *Journal of Applied Logic*, **22**, 46–62, (jul 2017).

[16] Daphne Koller and Nir Friedman, *Probabilistic graphical models: principles and techniques*, MIT press, 2009.

[17] Kevin B Korb and Ann E Nicholson, *Bayesian artificial intelligence*, CRC press, 2010.

[18] Steffen L Lauritzen and David J Spiegelhalter, 'Local computations with probabilities on graphical structures and their application to expert systems', *Journal of the Royal Statistical Society: Series B (Methodological)*, **50**(2), 157–194, (1988).

[19] Junkyu Lee, Radu Marinescu, Alexander T. Ihler, and Rina Dechter, 'A weighted mini-bucket bound for solving influence diagram', in *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, eds., Amir Globerson and Ricardo Silva, p. 432. AUAI Press, (2019).

[20] Zhifei Li and Jason Eisner, 'First- and second-order expectation semirings with applications to minimum-risk training on translation forests', in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pp. 40–51, Stroudsburg, PA, USA, (2009). Association for Computational Linguistics.

[21] Stephen M. Majercik and Michael L. Littman, 'Using caching to solve larger probabilistic planning problems', in *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pp. 954–959, Menlo Park, CA, USA, (1998). American Association for Artificial Intelligence.

[22] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt, 'Deepproblog: Neural probabilistic logic programming', in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, eds., Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, pp. 3753–3763, (2018).

[23] Radu Marinescu, 'A new approach to influence diagrams evaluation', in *Research and Development in Intelligent Systems XXVI, Incorporating Applications and Innovations in Intelligent Systems XVII, Peterhouse College, Cambridge, UK, 15-17 December 2009*, eds., Max Bramer, Richard Ellis, and Miltos Petridis, pp. 107–120. Springer, (2009).

[24] Mazen Melibari, Pascal Poupart, and Prashant Doshi, 'Sum-product-max networks for tractable decision making', in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, ed., Subbarao Kambhampati, pp. 1846–1852. IJCAI/AAAI Press, (2016).

[25] Mazen Melibari, Pascal Poupart, and Prashant Doshi, 'Sum-product-max networks for tractable decision making', in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1846–1852, New York, USA, (2016).

[26] Andrew Y Ng, Stuart J Russell, et al., 'Algorithms for inverse reinforcement learning', in *Icml*, volume 1, p. 2, (2000).

[27] Umut Oztok, Arthur Choi, and Adnan Darwiche, 'Solving pp^{pp}-complete problems using knowledge compilation', in *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, eds., Chitta Baral, James P. Delgrande, and Frank Wolter, pp. 94–103. AAAI Press, (2016).

[28] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole, 'Statistical relational artificial intelligence: Logic, probability, and computation', *Synthesis Lectures on Artificial Intelligence and Machine Learning*, **10**(2), 1–189, (2016).

[29] Constantin A. Rothkopf and Christos Dimitrakakis, 'Preference elicitation and inverse reinforcement learning', in *Machine Learning and Knowledge Discovery in Databases*, eds., Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, pp. 34–48, Berlin, Heidelberg, (2011). Springer Berlin Heidelberg.

[30] Marco Scutari. Bnlearn: Bayesian network repository. http://www.bnlearn.com/bnrepository/. Accessed: 2019-11-10.

[31] Marco Scutari and Jean-Baptiste Denis, *Bayesian networks: with examples in R*, Chapman and Hall/CRC, 2014.

[32] Dicky Suryadi and Piotr J. Gmytrasiewicz, 'Learning models of other agents using influence diagrams', in *Proceedings of the Seventh International Conference on User Modeling*, UM '99, pp. 223–232, Secaucus, NJ, USA, (1999). Springer-Verlag New York, Inc.

[33] Pedro Miguel Zuidberg Dos Martires, Anton Dries, and Luc De Raedt, 'Exact and approximate weighted model integration withprobability density functions using knowledge compilation', in *Proceedings of the 30th Conference on Artificial Intelligence*. AAAI Press, (2019).