International Journal of Approximate Reasoning 90 (2017) 411–432

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Weighted positive binary decision diagrams for exact probabilistic inference

Giso H. Dal [*], Peter J.F. Lucas

Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands

A R T I C L E   I N F O

Article history:
Received 17 October 2016
Received in revised form 5 August 2017
Accepted 10 August 2017
Available online 18 August 2017

Keywords:
Knowledge compilation
Probabilistic inference
Weighted model counting
Bayesian networks
Binary decision diagrams

A B S T R A C T

Recent work on weighted model counting has been very successfully applied to the problem of probabilistic inference in Bayesian networks. The probability distribution is encoded into a Boolean normal form and compiled to a target language. This results in a more efficient representation of local structure expressed among conditional probabilities. We show that further improvements are possible, by exploiting the knowledge that is lost during the encoding phase and by incorporating it into a compiler inspired by Satisfiability Modulo Theories. Constraints among variables are used as a background theory, which allows us to optimize the Shannon decomposition. We propose a new language, called *Weighted Positive Binary Decision Diagrams*, that reduces the cost of probabilistic inference by using this decomposition variant to induce an arithmetic circuit of reduced size.

1. Introduction

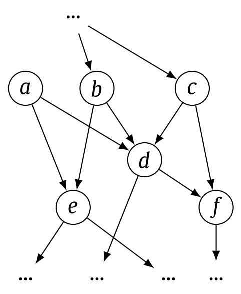
Bayesian networks, BNs for short, have been a subject of great interest, partly due to their contribution in solving real-life problems involving uncertainty. Bayesian networks are probabilistic graphical models that concisely represent joint probability distributions by factoring them into conditional probabilities based on independence assumptions. This will perform inference more efficiently [1]. Further representational and computational advances have been made by exploiting causal independence [2], as well as contextual independence [3] and determinism [4] expressed in conditional probability tables (CPTs). In order to represent these additional independencies more concisely, Bayesian networks have been represented as weighted Boolean formulas [5,6], reducing inference to *Weighted Model Counting* (WMC), or *weighted #SAT* [5]. By representing a Bayesian network as a Boolean formula f in *conjunctive normal form* (CNF), it can be compiled into a more concise normal form, or language, that renders inference a polytime operation in the size of the representation [7].

A joint probability space with n Boolean variables has 2^n interpretations. It is therefore necessary to be able to reason with sets of interpretations, requiring a symbolic representation [8]. Symbolic inference unifies the work of probabilistic inference and the extensive research done in the field of model checking, verification and satisfiability [9]. *Ordered Binary Decision Diagrams* (OBDDs) are based on Shannon decomposition and have been a very influential symbolic representation that reduces compilation to the problem of finding the variable ordering resulting in an optimal factoring.

By exploiting the often ignored knowledge that is lost in the translation, we improve in this paper on methods in recent work that encode a BN as an equivalent weighted Boolean formula, while still maintaining the ability to use off-the-shelf SAT-solvers.

[*] Corresponding author.
E-mail addresses: gdal@cs.ru.nl (G.H. Dal), peterl@cs.ru.nl (P.J.F. Lucas).

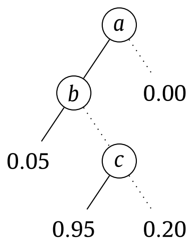
http://dx.doi.org/10.1016/j.ijar.2017.08.003
0888-613X/© 2017 Elsevier Inc. All rights reserved.



(**a**) Bayesian network

a	b	c	$P(d \mid a, b, c)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0.95
1	0	1	0.20
1	1	0	0.05
1	1	1	0.05

(**b**) d's CPT



(**c**) Decision tree

Fig. 1. Bayesian network with context-specific independence and determinism.

Our contributions are the following. We propose a weighted variant of OBDDs, called *Weighted Positive Binary Decision Diagrams* (WPBDDs), which are based on *positive Shannon decomposition*, allowing constraints in BNs to be represented more concisely. We use probabilities as *symbolic* edge weights, reducing the search space exponentially. An optimized compilation algorithm is introduced, inspired by the field of Satisfiability Modulo Theories (SMT), known as a *lazy SMT-solver* [10]. It provides the means to view constraints among variables in the encoding as background theory $\mathcal{T}$ which supports the SAT-solver, allowing *constant time conditioning*. We compile the conditional probability tables of a BN explicitly, but leave out the domain closure implied by the encoding. This approach allows us to remove up to a third of the clauses in the encoding.

A comparison is provided with the state-of-the-art CUDD (CU Decision Diagram [11]) and SDD (Sentential Decision Diagram [12]) compilers and we show that WPBDDs induce arithmetic circuits that are 60% reduced in size *on average* compared to corresponding OBDD circuits while representing over 30 publicly available BNs. We show an inference speedup of over 2.6 times on average compared to Weighted Model Counting with OBDDs, and a speedup up to several orders of magnitude compared to Ace and different implementations of the Junction Tree algorithm, making WPBDDs a valuable addition to the field of exact probabilistic inference.

After preliminaries and background (Section 4), we introduce WPBDDs (Section 5). The process of using a BN to perform exact inference by WMC is explained (Section 6) in addition to its optimization (Section 7). We conclude with experimental results (Section 8), and review achievements (Section 9). We start however with summarizing related work.

2. Motivation

Probabilistic inference is an important computational problem in Artificial Intelligence. The size of a full joint probability distribution is exponential in the number of variables. Finding more concise representations of probability distributions is essential to making probabilistic inference tractable.

Bayesian networks (BN) take advantage of *conditional independence* to represent probability distributions concisely, yet generally leave a substantial number of independencies of other types unexploited. This is demonstrated in Fig. 1, which partially shows a BN, where one of its conditional probability tables (CPTs) could be represented more concisely. It shows probabilities for $d = 0$. Remember that $P(d = 1 \mid a, b, c) = 1 - P(d = 0 \mid a, b, c)$. Fig. 1c shows how to represent the same CPT as a decision tree using 4 probabilities instead of 8 for example.

Recent work has improved the cost of inference by reducing probabilistic inference to Weighted Model Counting (WMC). Inference by WMC has a linear time complexity in the size of the representation [13]. Finding more concise representations by exploiting additional independencies has therefore direct influence on inference costs. We improve upon related work by proposing a suitable representation and providing the methods to obtain it.

3. Related work

Probabilistic inference is a hard computational problem that can be achieved by marginalizing out non-evidence variables from a joint distribution, requiring an exponential number of operations in the worst case. Efforts toward efficient exact probabilistic inference attempt to find a concise factorization; BNs for instance represent a joint probability distribution as a multiplicative factorization, exploiting conditional independencies among variables. Further improvements to this factorization have been made by using propositional and first order logic [14–16]. Representing probability distributions as Boolean functions empowers probabilistic inference with the tools developed for VLSI-CAD design, by using symbolic representations and Boolean algebra for minimization. *Symbolic Probabilistic Inference* (SPI) [9] is a good example of this, which is currently more commonly referred to as inference by WMC or #SAT [17].

A BN can be translated into a Satisfiability (SAT) instance in *conjunctive normal form* (CNF), a form commonly used in satisfiability solving. Various encodings have been proposed: log, direct [18], order [19], compact order [20], log-support encoding [21], etc. In the context of BNs, a probability distribution can be considered a pseudo Boolean function $f : \{0, 1\}^n \to \mathbb{R}$, with arity n, which can be uniquely written as an exponentially sized multi-linear polynomial [22,23]. Others have used the direct encoding [6], or a combination of the direct and order encoding [24], where a BN is viewed as a set of discrete real valued functions, and each function represents a distinct CPT.

Inference by WMC is motivated by linear time complexity in the size of the representation [13], where the common goal is to exploit local structure [25]. Choosing a representation or *language* to compile to is therefore a critical task, as one must deal with the balance between the functions a language can represent concisely on the one hand and its algorithmic properties on the other. Initial attempts include probability trees [3,26,27] and recursive (factored) probability trees [28], which focus on concisely representing each CPT independently, allowing their usage in inference algorithms directly. Probabilistic Decision Graphs (PDG) have even shown that the smallest PDG is at least as small as the smallest Junction tree for the same distribution [29].

Current WMC approaches to inference divide into *search* and *compilation* methods [30]. Typical search algorithms are based on DPLL-style SAT solvers that do an exhaustive run to count all satisfying models [24]. Recording SAT evaluation paths (i.e. resolution steps) as a compiled structure (e.g. an OBDD), yields one possible factoring. We refer to finding the optimal factoring given all variable orderings, as *exact* compilation.

Compilation performance has been improved by clause learning [31], formula caching [32], bounding [33], and using canonical languages. Representational advances include symmetry detection [34,35], support for causal independence [36] and using *read-once functions* [37].

Representations relevant in the context of BN compilation are *AND/OR Multi-Valued Decision Diagrams* (AOMDD) [38], *Sentential Decision Diagrams* (SDD) [12], deterministic-DNNF [39], Zero-suppressed Binary decision diagrams (ZBDD) [22] and Ordered Binary Decision Diagrams (OBDD) [40], which view probabilities as auxiliary literals, resulting in an intractably large search space. Multi-Terminal BDDs [41] represent multi-valued functions, but would also require many terminal nodes considering the size of a probability distribution. Variants of Edge-Value BDDs [42,43] focus on real valued functions. When multiple CPTs have probabilities in common, we lose the ability to distinguish from which CPT the probabilities originate. We therefore cannot determine on which variables they depend, resulting in an inconsistent model count with regard to the distribution. Our approach to maintain consistency is to represent probabilistic edge weights *symbolically*. This differentiates our approach from Multi-Terminal BDDs [41] and Edge-Value BDDs [42]. And unlike SDDs [44], we are not obligated to view probabilities as auxiliary literals, reducing the search space to a fraction of its former size. A common characteristic with ZBDDs, is the ability to represent mutual-exclusive constraints more concisely [45]. The intuitive difference is that ZBDD optimize only the positive cofactor, while we optimize both the positive and negative cofactor of decomposition nodes, a matter we will elaborate on in the upcoming discussion.

4. Preliminaries and background

We provide here a description of what Bayesian networks are and introduce a running example (Section 4.1). We then show how to encode a BN onto the Boolean domain (Section 4.2), and describe an influential representation that will serve for comparison with ours (Section 4.3).

4.1. Bayesian networks

A Bayesian Network (BN) is a graphical representation that is used to compactly represent a joint distribution as a product of factors, by taking advantage of *conditional independence* (CI). A BN is a directed acyclic graph (DAG) that models variables X as nodes, the dependencies among them as edges, and their joint probability distribution as

$$P(X) = P(x^1, ..., x^n) = \prod_{i=1}^{n} P(x^i \mid pa(x^i)),$$

where $P(x^i \mid pa(x^i))$ represents the conditional probability of variable x^i given its parents $pa(x^i)$. Conditional probability tables (CPTs) are associated with edges and capture the degree to which variables are related. BNs reduce the size of representing a probability distribution to $\mathcal{O}(n2^k)$, where k is the maximum number of parents of any node.

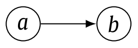
Example 1. Fig. 2 shows a BN $\mathcal{B}$ defined over variables $X = \{a, b\}$ (Fig. 2b), its CPTs (Fig. 2c) and corresponding full joint probability distribution (Fig. 2a).

Fig. 2b includes a factored form that can greatly be improved when CPTs exhibit *local structure*, which comes in two forms. *Context-specific independence* (CSI) is expressed when probabilities in a CPT show uniformity regardless of the value of one or more variables that they have in common, with or without a certain context. *Determinism* is expressed when probabilities in a CPT are equal to 0 or 1, which can be used to simplify the Boolean formula representing it.

In order to exploit more of the problem structure than a BN, we *compile* it to a target language that is more capable of doing so. The compilation process is not just a way of reformulating into a different language, but is also about finding the minimal representation given that language. The goal is to reduce the cost of inference compared to the standard factorization, by using the arithmetic circuit induced by that representation.

m	a	b	$P(X)$
1	1	1	0.05
2	1	2	0.05
3	2	1	0.05
4	2	2	0.05
5	3	1	0
6	3	2	0.8

(a) Joint Distribution



$$P(X) = P(b|a)P(a)$$

(b) Bayesian network

$P(a=1)$	$P(a=2)$	$P(a=3)$
0.1	0.1	0.8

a	$P(b=1 \mid a)$	$P(b=2 \mid a)$
1	0.5	0.5
2	0.5	0.5
3	0	1

(c) Conditional probability tables

Fig. 2. Bayesian network with local structure.

4.2. Encoding Bayesian networks

In order to exploit local structure, we encode BNs into *conjunctive normal form* (CNF), which is the most common representation used in satisfiability solving. It consists of a conjunction of clauses, where each clause is a disjunction of literals. A literal is a propositional Boolean variable or its negation. A Bayesian Network defined over variables X can be seen as a multi-linear function $f : X \to \mathbb{R}$. There are several possibilities of mapping f onto the Boolean domain [6,18,46]. We encode f into a Boolean function $\mathcal{E}(f) = f^e$ by representing it as a weighted CNF.

$$\mathcal{E}(f) = f^c \wedge f^m, \tag{1}$$

where constraint clauses f^c support the mapping $\mathcal{M}(f) = f^m$ that encodes probabilities and introduces a Boolean variable for each unique variable-value pair. Details are discussed below.

4.2.1. Encoding constraints

The mapping function $\mathcal{M}$ introduces for each $x \in X$ atoms $\mathcal{A}(x) = \{x_1, \ldots, x_n\}$, where x_i signifies x being equal to its i^{th} value. To maintain consistency among variables we add to f^c an *at-least-once* (ALO) constraint clause for each x, to ensure x is assigned a value:

$$(x_1 \vee \cdots \vee x_n) \tag{2}$$

As values of a variable are mutually exclusive, we add to f^c the following *at-most-once* (AMO) constraint clauses:

$$\bigwedge_{i=1}^{n} \left(x_i \implies \bigwedge_{x_j \in \mathcal{A}(x) \setminus x_i} \overline{x_j} \right) = \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\overline{x_i} \vee \overline{x_j}), \tag{3}$$

where $\overline{x_i}$ indicates the negation of x_i.

4.2.2. Encoding CPTs

The BN's factored form is preserved by using a weighted adaptation of the *direct* encoding. The mapping function $\mathcal{M}$ adds a clause for every probability $P(x|pa(x))$, where x depends on parent variables $pa(x) = \{u^1, \ldots, u^r\}$:

$$(x \wedge u^1 \wedge \cdots \wedge u^r \Rightarrow \omega) = (\overline{x} \vee \overline{u^1} \vee \cdots \vee \overline{u^r} \vee \omega), \tag{4}$$

where atom ω represents probability $P(x|pa(x))$ symbolically. This symbolism has no influence on its propositional meaning, but will later be used during inference. We shall henceforth represent a *weighted clause* as $(\overline{x} \vee \overline{u^1} \vee \cdots \vee \overline{u^r}) : \omega$. We introduce a new symbolic weight into the encoding for every unique probability local to x's CPT, allowing multiple clauses to be associated with the same weight ω.

Example 2. Assume a BN as given in Example 1. The following clauses form f^c:

Variable	ALO (Eq. 2)	AMO (Eq. 3)
a	$(a_1 \vee a_2 \vee a_3)$	$(\overline{a_1} \vee \overline{a_2}) \wedge (\overline{a_1} \vee \overline{a_3}) \wedge (\overline{a_2} \vee \overline{a_3})$
b	$(b_1 \vee b_2)$	$(\overline{b_1} \vee \overline{b_2})$

The variables that make up the search space during compilation therefore are $\mathcal{A}(\{a, b\}) = \{a_1, a_2, b_1, b_2, b_3\}$. We encode equal probabilities $P(x|U)$ as unique symbolic weights per CPT:

$P(a=1)$	$P(a=2)$	$P(a=3)$
ω_1	ω_1	ω_2

a	$P(b=1|a)$	$P(b=2|a)$
1	ω_3	ω_3
2	ω_3	ω_3
3	ω_4	ω_5

In accordance with Equation (4), f^m consists of the following clauses, accompanied by their respective symbolic weights:

$$\begin{aligned}(\overline{a_1}):\omega_1 \;\wedge\; (\overline{a_2}):\omega_1 \;\wedge\; (\overline{a_3}):\omega_2 \;\wedge\\ (\overline{a_1}\vee\overline{b_1}):\omega_3 \;\wedge\; (\overline{a_1}\vee\overline{b_2}):\omega_3 \;\wedge\; (\overline{a_2}\vee\overline{b_1}):\omega_3 \;\wedge\\ (\overline{a_2}\vee\overline{b_2}):\omega_3 \;\wedge\; (\overline{a_3}\vee\overline{b_1}):\omega_4 \;\wedge\; (\overline{a_3}\vee\overline{b_2}):\omega_5.\end{aligned}$$

4.3. Ordered binary decision diagrams

A Boolean function f defined over a set of variables X is a function that maps each complete assignment of its variables to either *true* (1) or *false* (0). The *conditioning* of f on instantiated variable $\boldsymbol{x_i}$ is defined as the projection:

$$f_{|x_i \leftarrow b}(x_1,\ldots,x_n) = f(x_1,\ldots,x_{i-1},b,x_{i+1},\ldots,x_n), \tag{5}$$

with $b \in \{0,1\}$. We will use shorthand notations $f_{|x_i}$ and $f_{|\overline{x_i}}$ for $f_{|x_i\leftarrow 1}$ and $f_{|x_i\leftarrow 0}$, respectively. Shannon's theorem is used to find a more compact way to represent f by *factoring* it.

Theorem 1. [47] *Let a Boolean function $f:\{0,1\}^n \to \{0,1\}$ be defined over variables X. We can rewrite f using the following equivalence, with $x \in X$:*

$$f = x \wedge f_{|x} \;\vee\; \overline{x} \wedge f_{|\overline{x}}.$$

Definition 1. Theorem 1 demonstrates a decomposition step referred to as *Shannon's expansion*, where $f_{|x}$ is called the positive *cofactor* of f with respect to x, and $f_{|\overline{x}}$ the negative cofactor. The *decomposition* of f is defined as the recursive application of Shannon's expansion to cofactors until they evaluate to true or false.

Note that all proofs of theorems, lemmas, etc., can be found in the appendix. The *Shannon decomposition* is key to one of the most influential representations in Artificial Intelligence (AI), namely *Ordered Binary Decision Diagrams* (OBDD) [48].

Definition 2. [49] A *Binary Decision Diagram* (BDD) represents Boolean function f defined over variables X as a rooted, directed acyclic graph, where each node v represents a Shannon expansion on variable $var(v) \in X$. A BDD is *ordered* (OBDD) if variables appear in the same order on all paths from the root. It is a *canonical* representation if it is *reduced* by applying the following rules:

1. Merge rule: All isomorphic subgraphs are merged.
2. Delete rule: All nodes are removed whose children are isomorphic.

One can compile the described encoding of BNs to OBDDs to perform inference by WMC. Although other languages have been used in this context, we focus more on OBDDs as they are commonly used in comparisons with related work.

5. Weighted positive binary decision diagrams

Consider variable x and its corresponding mapped atoms $\mathcal{A}(x) = \{x_1, x_2\}$. Decision diagrams that are based on Shannon decompositions produce an unnecessarily large representation by redundantly representing constraints f^c provided as part of encoding $\mathcal{E}$. They also do not take advantage of the symmetric relation $x_1 = \overline{x_2}$ and $\overline{x_1} = x_2$ in the presence of ALO constraints. To ameliorate this, we propose a new canonical language called *Weighted Positive Binary Decision Diagrams* (WPBDD), which are based on *positive Shannon decompositions* and *implicit conditioning*. We will elaborate on these concepts through an intermediate unweighted variant of WPBDDs (PBDD).

5.1. Explicit and implicit conditioning

When encoded function f^e contains a *unit clause* x_i (a clause consisting of a single literal), it can be simplified using *unit propagation*:

1. Every clause containing x_i is removed, excluding the unit clause.
2. Literal $\overline{x_i}$ is removed from every clause containing it.

When conditioning f^e on literal x_i, constraints f^c guarantee us to obtain unit clauses containing negated literals $\overline{x_j}$, with $x_j \in \mathcal{A}(x)\backslash x_i$ (i.e., if x is equal to its i^{th} value, it cannot be equal to its j^{th} value).

Example 3. We will show by example what unit clauses are obtained by conditioning on positive literals. Consider the constraint clauses provided by Example 2, regarding only variable a:

$$f^c = (a_1 \vee a_2 \vee a_3) \wedge (\overline{a_1} \vee \overline{a_2}) \wedge (\overline{a_1} \vee \overline{a_3}) \wedge (\overline{a_2} \vee \overline{a_3})$$

According to Shannon's expansion the following holds:

$$f^c = a_1 \wedge f^c_{|a_1} \;\vee\; \overline{a_1} \wedge f^c_{|\overline{a_1}}$$

Now specifically look at the unit clauses that result from conditioning on positive literal a_1, i.e., instantiating a_1 and performing unit propagation.

$$\begin{aligned} f^c_{|a_1} &= (1 \vee a_2 \vee a_3) \wedge (0 \vee \overline{a_2}) \wedge (0 \vee \overline{a_3}) \wedge (\overline{a_2} \vee \overline{a_3}) \\ &= (1) \wedge (\overline{a_2}) \wedge (\overline{a_3}) \wedge (\overline{a_2} \vee \overline{a_3}) \\ &= (1) \wedge (\overline{a_2}) \wedge (\overline{a_3}) \\ &= (\overline{a_2}) \wedge (\overline{a_3}). \end{aligned}$$

Thus, conditioning on a_i will result in unit clauses containing negated literals $\overline{a_j}$, with $a_j \in \mathcal{A}(a) \setminus \{a_i\}$.

We distinguish between two types of conditioning based on the previous observation, and describe them in the following definition.

Definition 3. Let f^e be an encoded representation of function f, given encoding $\mathcal{E}$, where f is defined over variables X. We define $f^e_{\|x_i}$ as the conditioning of f^e on literals $\{x_i, \overline{x_1}, \dots, \overline{x_{i-1}}, \overline{x_{i+1}}, \dots, \overline{x_n}\}$ in any order, i.e., as the *explicit* conditioning of f^e on literal $x_i \in \mathcal{A}(x)$, and its *implicit* conditioning on literals $\overline{x_j} \in \mathcal{A}(x)\backslash x_i$, with $x \in X$:

$$f^e_{\|x_i} = f^e_{|x_i, \overline{x_1}, \dots, \overline{x_{i-1}}, \overline{x_{i+1}}, \dots, \overline{x_n}}.$$

It follows from Definition 3 and the constraints provided by encoding $\mathcal{E}$, that the relation between $f^e_{\|x_i}$ and $f^e_{|x_i}$ is given by the following equality:

$$f^e_{|x_i} = \left(\bigwedge_{\overline{x_j} \in \mathcal{A}(x)\backslash x_i} \overline{x_j} \right) \wedge f^e_{\|x_i}. \tag{6}$$

Implicit conditioning on unit clauses takes advantage of deterministic behavior expressed in f^c, while other representations would explicitly have to condition these unit clauses out. The advantage is two-fold. The size of the encoding can be reduced by removing constraint clauses f^c generated by Equation (2) and (3), and integrating them directly into the compilation process through theory $\mathcal{T}$. As will be shown later, this separation will allow the constraint clauses from f^e to be conditioned in *constant time*, as opposed to quadratic time. Secondly, the size of the compiled structure is reduced by not having to represent redundant constraint information using our variant on the Shannon expansion, introduced in the following section.

5.2. Positive Shannon decomposition

We propose *positive Shannon decompositions* that use background knowledge to improve upon Shannon decompositions by combining it with implicit conditioning.

Lemma 1. *Let an* encoded *Boolean function* $f^e : \{0,1\}^n \to \{0,1\}$ *be defined over* $\mathcal{A}(X)$, *obtained by* $\mathcal{E}(f)$, *with* f *defined over* X. *We can rewrite* f^e *using the following equivalence:*

$$f^e = \; f^c \;\wedge\; \left(x_i \wedge f^e_{\|x_i} \;\vee\; f^e_{|\overline{x_i}}\right),$$

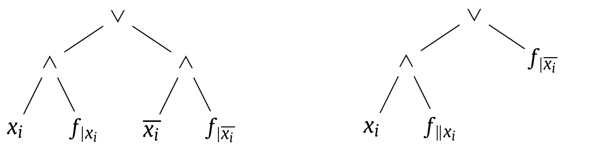
where $f^e = f^c \wedge f^m$, *and* $x_i \in \mathcal{A}(x)$, *with* $x \in X$.

As intuition might confirm, logical representations will grow by the reintroduction of constraints when we apply Lemma 1. We introduce a *reduced* form by removing constraint clauses f^c that introduces additional models, in turn *allowing us to find more concise representations*. These models can easily be removed by a post decomposition conjoin with f^c.

Theorem 2. *Let an* encoded *Boolean function* $f^e : \{0,1\}^n \to \{0,1\}$ *be defined over* $\mathcal{A}(X)$, *obtained by* $\mathcal{E}(f)$, *with* f *defined over* X. *Equivalent under constraints* f^c, *we can rewrite* f^e *using:*

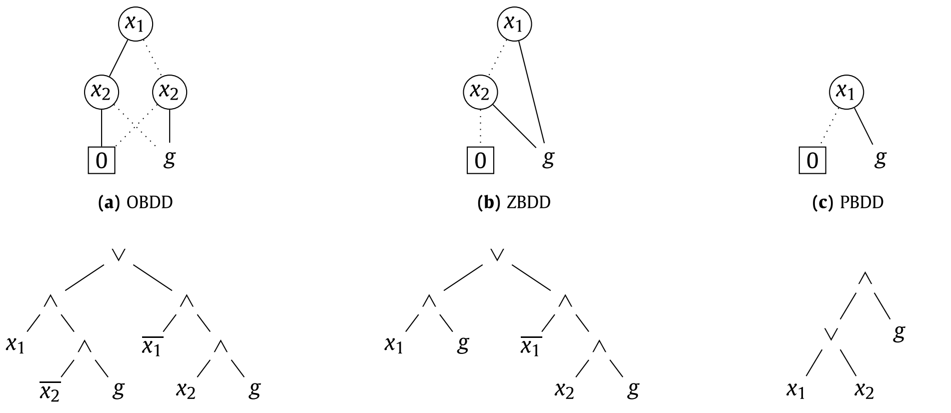
$$f^e \;\models\; x_i \wedge f^e_{\|x_i} \;\vee\; f^e_{|\overline{x_i}},$$

where $f^e = f^c \wedge f^m$, *and* $x_i \in \mathcal{A}(x)$, *with* $x \in X$.



(**a**) Shannon logical circuit (**b**) Positive Shannon logical circuit

Fig. 3. Logical circuits.



(**a**) OBDD (**b**) ZBDD (**c**) PBDD

(**d**) OBDD logical circuit (**e**) ZBDD logical circuit (**f**) PBDD logical circuit

Fig. 4. From BDDs to logical circuits.

Definition 4. Theorem 2 demonstrates a decomposition step referred to as the *(reduced) positive Shannon expansion*, where $f_{\|x}$ is called the positive *cofactor* of f with respect to x and is defined in Definition 3, and $f_{|\overline{x}}$ is called the negative cofactor. The *decomposition* of f is defined as the recursive application of the positive Shannon expansion to cofactors until they evaluate to true or false.

Let $\circ$ be a Boolean operator and let g and h be encoded Boolean functions, according to encoding $\mathcal{E}$. From Theorem 2 we derive how to create the representation of a function according to the operators in a Boolean expression:

$$g \circ h \;=\; x \wedge (g_{\|x} \circ h_{\|x}) \;\vee\; (g_{|\overline{x}} \circ h_{|\overline{x}}). \tag{7}$$

By using the reduced form of the expansion, we are able to represent constraints more concisely in corresponding logical circuits (Fig. 3), as well as in the soon to be introduced representation that utilizes it.

For OBDDs, it is not possible for the conditioning of constraint clauses f^c on any single variable $x_i \in \mathcal{A}(x)$ to result in equal cofactors (isomorphic children). This prohibits the application of the delete rule, effectively rendering OBDDs incapable of capturing local structure along the dimension of single variables. To ameliorate this, we introduce *positive* OBDDs (PBDD) as an unweighted intermediate representation, that are based on the positive Shannon decomposition and substitutes the delete rule with the *collapse* rule, which as opposed to deleting literals, applies the *distributive law* to involved literals in the induced logical circuit.

Definition 5. A *positive* OBDD (PBDD) represents Boolean function $\mathcal{E}(f) = f^e$, where f is defined over variables X, as an *ordered* BDD where each node v represents a positive Shannon decomposition on variable $var(v) \in \mathcal{A}(X)$. It is a *canonical* representation if *reduced* by applying the following rules:

1. Merge rule: All isomorphic subgraphs are merged.
2. Collapse rule: remove direct descendant u of node v iff $f_{\|x_i} = f_{\|x_j}$, where $var(v) = x_i$ and $var(u) = x_j$, with $x_i, x_j \in \mathcal{A}(x)$ and $x \in X$.

A function *essentially* depends on a variable if it appears in its prime implicate. The variable set $\mathcal{S}$, on which f^e essentially depends, is called the *support* of f^e. We will use this support set to identify to what variables the collapsed rule has been applied in order to produce its corresponding logical circuit, a trick similarly utilized with Zero-Suppressed BDDs (ZBDD) [45]. Note that a Boolean function $\mathcal{E}(f) = f^e$, where f is defined over variables X, essentially depends on $\mathcal{A}(X)$, because f^c is a prime implicate that mentions all $\mathcal{A}(X)$. The canonical property of PBDDs follows from the fact that a binary tree can be reconstructed from a PBDD and its support set, by applying its reduction rules reversely.

Fig. 4 shows the difference in representational size between an OBDD, a ZBDD and a PBDD representing the same function with constraints on $\mathcal{A}(x) = \{x_1, x_2\}$, where g is a Boolean function that does not essentially depend on literals

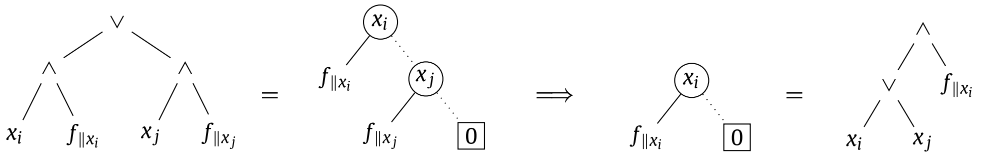


Fig. 5. Application of the collapse rule, where cofactors are equal.

$\{x_1, x_2\}$. The positive Shannon decomposition not only reduces the size of the corresponding logical circuit, it also reduces the size of the representation. More generally, OBDDs require an exponential number of nodes in the product of each constraint variable's dimension, where ZBDDs require a linear number, and where PBDDs require only 1 node.

Fig. 4 shows that there are functions where the corresponding minimal PBDD and OBDD differ exponentially in size. Not being represented in the figure, the collapse rule additionally removes the nodes that share the same positive cofactor with their parents (Fig. 5).

The semantics of nodes, whose children have been removed, changes. A missing node, inferable by the support set $\mathcal{S}$ and variable order, indicates the application of the collapse rule, i.e., the distributive law on x_i and x_j. There is no ambiguity regarding the delete rule as it can never be applied on $\mathcal{A}(X)$ due to constraint clauses f^c. Note however that the delete rule can be applied in case one optimizes Boolean variables of the BN by representing them with only one literal in f^e, as opposed to two. This will delete literals from the induced circuit and result in an inconsistent model count with regard to the probability distribution. It is precisely for this reason that the collapse rule uses the distributive law on involved literals to simplify the induced circuit, as opposed to deleting them from it. The combination of the merge and collapse rule facilitates a more finely grained control in exploiting CSI, because it allows independence given a subset of the values to be expressed more efficiently when dealing with multi-valued variables.

Proposition 1. *An OBDD representing Boolean function g and an PBDD representing $\mathcal{E}(g)$ induce isomorphic logical circuits under Boolean identity, given an appropriate ordering.*

Proposition 2. *Given an ordering on $\mathcal{A}(X)$, the size of PBDD φ is less than the size of OBDD ψ when they both represent $\mathcal{E}(f) = f^e$, where f is defined over variables X.*

5.3. Adding probabilities as weights

Encoding $\mathcal{E}$ represents a BN as a weighted propositional formula. We extend PBDDs to *weighted* PBDDs (WPBDD) using an intuitive scheme, taking advantage of the fact that probabilities are fully implied by the variables in the BN. Traditionally, an empty clause would result in a contradiction, i.e., the instantiation is unsatisfiable. We *implicitly* assign weight ω of empty clause c to the edge it is associated with, and remove c from the expression. When multiple empty clauses are associated with an edge, we simply assign the conjunction (multiplication) of their weights to the edge. To maintain canonicity, we only assign weights on the side of the positive cofactor.

Definition 6. A *weighted* PBDD (WPBDD) representing Boolean function $\mathcal{E}(f) = f^e$, where f is defined over variables X, is a PBDD where each node v is a tuple $\langle x_i, W, f^e_{\|x_i}, f^e_{|\overline{x_i}}\rangle$ that represents a weighted reduced positive Shannon expansion:

$$f^e \;\models\; x_i \wedge (W \wedge f^e_{\|x_i}) \;\vee\; f^e_{|\overline{x_i}},$$

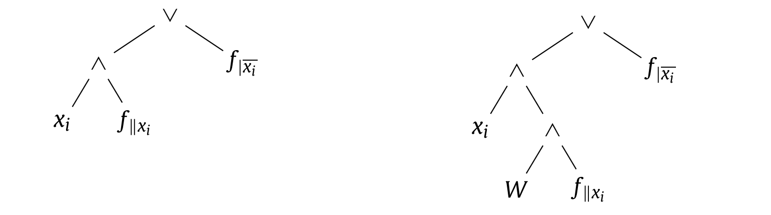
where $x_i \in \mathcal{A}(X)$, W is a conjunction of weights ω_i, and $(W \wedge f^e_{\|x_i}) = f^e_{\|x_i}$ under idempotence. It is a *canonical* representation if *reduced* by applying the following rules:

1. Merge rule: All isomorphic subgraphs are merged.
2. Collapse rule: remove direct descendant u of node v iff $W \wedge f_{\|x_i} = W \wedge f_{\|x_j}$, where $var(v) = x_i$ and $var(u) = x_j$, with $x_i, x_j \in \mathcal{A}(x)$ and $x \in X$.

Let $\circ$ be a Boolean operator and let g and h be encoded Boolean functions, according to encoding $\mathcal{E}$. From Definition 6 we derive how to create the representation according to the operators in a Boolean expression:

$$g \circ h \;=\; x \wedge ((W_g \wedge g_{\|x}) \circ (W_h \wedge h_{\|x})) \;\vee\; (g_{|\overline{x}} \circ h_{|\overline{x}}), \tag{8}$$

where W_g and W_h are the (conjunction of) weights that correspond to g and h, respectively. Fig. 6 shows the logical circuits induced by a weighted and unweighted node, and Fig. 7 shows how we extended the collapse rule in order to deal with weights.



(a) Positive Shannon logical circuit **(b)** Weighted Positive Shannon logical circuit

Fig. 6. Logical circuits.

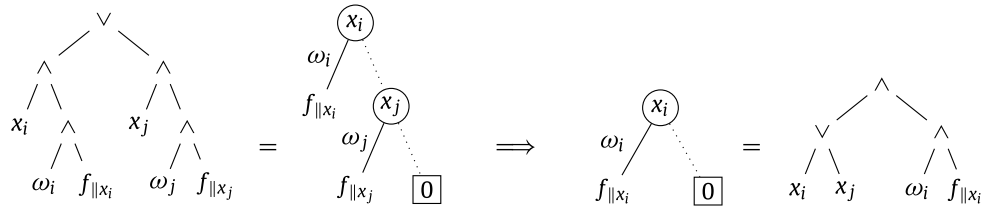
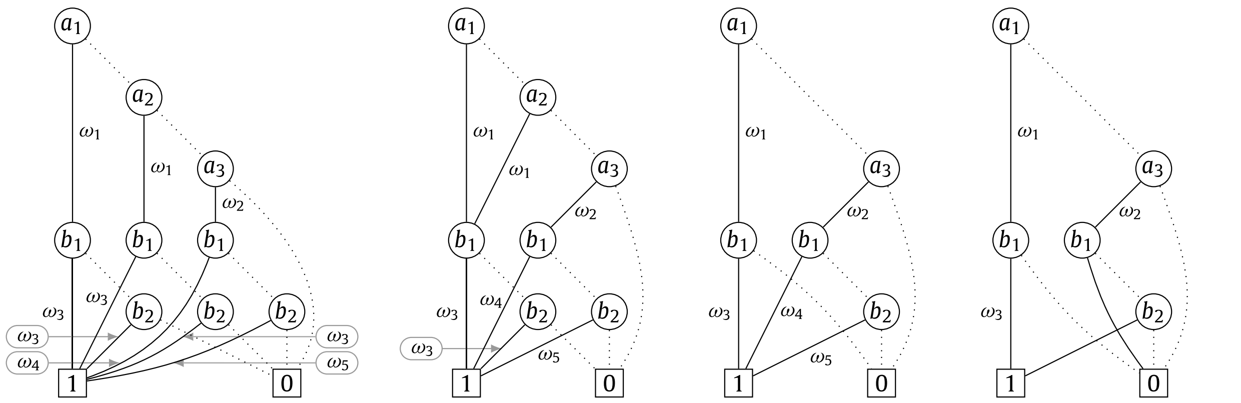


Fig. 7. Application of the collapse rule, where functions $f_{\|x_i} = f_{\|x_j}$ and weights $\omega_i = \omega_j$.



(a) Not reduced **(b)** Merged **(c)** Merged and collapsed **(d)** Determinism, merged and collapsed

Fig. 8. WPBDD reduction.

Example 4. Consider the CPTs from Example 1, where per CPT, equal probabilities are represented by unique symbolic weights ω_i.

$P(a=1)$	$P(a=2)$	$P(a=3)$
ω_1	ω_1	ω_2

a	$P(b=1\|a)$	$P(b=2\|a)$
1	ω_3	ω_3
2	ω_3	ω_3
3	ω_4	ω_5

Fig. 8 shows the minimization of a WPBDD using variable ordering $a_1 < a_2 < a_3 < b_1 < b_2$, that represents the BN with 3 probabilities instead of 9. The collapse rule contributes to the reduction of the model by applying it for instance to the node representing a_1. It would be prohibited when a_2 and a_3 are swapped in the ordering. This demonstrates that the ordering determines the degree to which a WPBDD can be reduced by using the reduction rules. Fig. 9 shows the comparison of this WPBDD with an OBDD representing the same function, given variable ordering $a_1 < a_2 < \omega_1 < a_3 < \omega_2 < b_1 < \omega_5 < b_2 < \omega_3 < \omega_4$, which results in a minimal OBDD that obeys the partial ordering used for the WPBDD. The WPBDD induces a logical circuit consisting of 17 logical operators, while the OBDD requires 54.

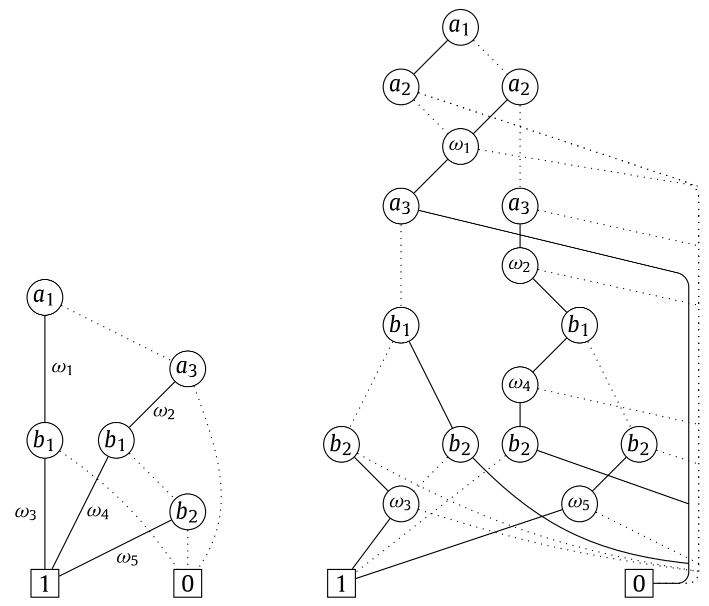


Fig. 9. WPBDD and OBDD comparison.

6. Symbolic inference

We perform Bayesian inference through a process of three phases: encoding, compiling and model counting:

	Composition	Input	Output
Encoder	Encoding $\mathcal{E}$	f	Theory $\mathcal{T} + f^e$
Compiler	$\mathcal{T}$-solver + SAT	$\mathcal{T} + f^e$	WPBDD φ
Counter	$\mathcal{T}$-solver + WMC	$\mathcal{T} + \varphi$	$P(x\|\boldsymbol{e})$

A BN represented by f is first encoded by the *encoder* as Boolean function f^e using encoding $\mathcal{E}$ as defined in Section 4.2. The encoder also provides background theory $\mathcal{T}$ representing f^c, i.e., the constraints among variables that support mapping $\mathcal{M}$.

The *compiler* uses a lazy SMT-solver to record evaluation paths as a WPBDD. The lazy SMT-solver combines a SAT-solver with a theory-solver, were a communication between the two is facilitated in order to support implicit conditioning.

The *counter* computes the probability of x given evidence $\boldsymbol{e}$, by translating the provided WPBDD into an arithmetic circuit and using the theory-solver to properly instantiate the variables.

6.1. Compilation

It is well known that the variable ordering used during compilation determines the representation size [50]. Compilation therefore reduces to finding an optimal variable ordering. Satisfiability (SAT) is key during compilation, where we consecutively condition CNF f in the sequence provided by the ordering to determine if a given valuation is satisfiable. Normally, when CNF f contains an empty clause as the result of conditioning, we derive a contradiction (unsatisfiable).

For optimization purposes we deal slightly differently with weighted clauses. Remember that a weighted clause is the same as the clause disjoined with the literal that symbolizes its weight. Conventional SAT solving would require the ordering to be defined over literals that symbolize a weight and literals that do not. By removing weight literals from the ordering we reduce the search space of possible variable orderings considerably, and at the same time we dynamically position weight literals in a proper position in the ordering. When a weighted clause is empty (all non weight literals have been conditioned out), we are essentially left with a unit clause containing only the weight literal. Upon discovery, we remove this unit clause (empty weighted clause) by consecutively conditioning on the containing weight literal, essentially positioning the weight literal dynamically in the ordering. This approach preserves the semantics of clauses, thereby still allowing conventional SAT solvers to be used.

Algorithm 1 shows the compilation procedure, where the encoded Bayesian network $f^e = f^c \vee f^m$ is compiled to the corresponding WPBDD by SMT-Solver using ordering O. We have used underscore (_) to indicate where we ignore a return value. A WPBDD is a directed graph that is build using nodes of type Node:

Algorithm 1 Compiler.

```
CONDITION($f, L$)
    input: CNF $f$ and literal set $L$.
    output: $f$ conditioned on literals $L$.
 1  if IS-EMPTY($L$)
 2      return $f$, {}
 3  else
 4      for $l$ in $L$
 5          if IS-NEGATED($l$)
 6              $f = f_{|l \leftarrow 0}$
 7          else
 8              $f = f_{|l \leftarrow 1}$
 9      $L_u$ = GET-UNIT-LITERALS($f$)
10      $f, \_ =$ CONDITION($f, L$)
11      return $f, L_u$

COMPILE($f^c, f^m, O, i$)
    input: CNF $f^c$ and $f^m$, ordering $O$,
        and evaluation depth $i$.
    output: A WPBDD representing $f^c \vee f^m$.
 1  if IS-SATISFIABLE($f^c \vee f^m$)
 2      return TRUE-TERMINAL()
 3  elseif IS-UNSATISFIABLE($f^c \vee f^m$)
 4      return FALSE-TERMINAL()
 5  else
 6      $l = O[i]$
 7      $node$ = CREATE-NODE()
 8      $node.l = l$
 9
10      $f^c_l\ , L_u =$ CONDITION($f^c\ , \{l\}$)
11      $f^m_l, W =$ CONDITION($f^m, \{l\} \cup L_u$)
12      $node.weights = W$
13      $node.pCofactor$ = COMPILE(
            $f^c_l, f^m_l, O, i{+}1$)
14
15      $f^c_{\bar{l}}\ , \_ =$ CONDITION($f^c\ , \{\bar{l}\}$)
16      $f^m_{\bar{l}}, \_ =$ CONDITION($f^m, \{\bar{l}\}$)
17      $node.nCofactor$ = COMPILE(
            $f^c_{\bar{l}}, f^m_{\bar{l}}, O, i{+}1$)
18
19      $node$ = COLLAPSE($node$)
20      $node$ = MERGE($node$)
21      return $node$

SMT-SOLVER($f^c, f^m, O$)
    input: Encoded Bayesian network
        $f^e = f^c \vee f^m$, and ordering $O$.
    output: A WPBDD representing $f^e$.
 1  return COMPILE($f^c, f^m, O, 0$)
```

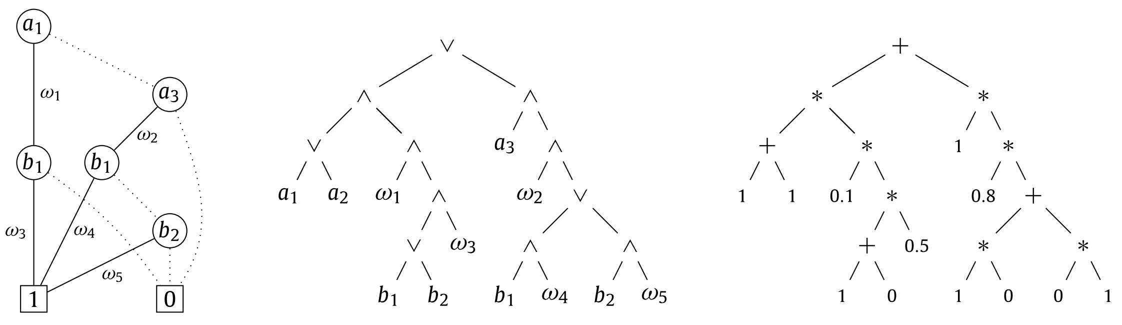
struct Node:

Literal : l	//*the literal used for conditioning cofactors*
Node : pCofactor	// *represents positive cofactor* $f_{\|l}$
Node : nCofactor	// *represents negative cofactor* $f_{\|\bar{l}}$
Literal set : weights	//*weight literals of positive cofactor*

Initialization: Let function f represent a Bayesian network. We obtain its CNF encoding $\mathcal{E}(f) = f^e$, and provide constraints f^c and probability encoding f^m as separate inputs to SMT-SOLVER. An ordering is determined by the literals on which f^c essentially depends, i.e., the literals that do not symbolically represent a weight. Using the encoding and ordering, SMT-SOLVER builds and returns the corresponding WPBDD.

Synopsis: The compiler consists of a theory-solver and SAT-solver. We will provide insight into the separation of the two, and where implementational particularities should be considered in the provided algorithm. A WPBDD is build in a DPLL-style depth-first manner, of which the order is determined by O. If f^e is found to be satisfiable or unsatisfiable (IS-SATISFIABLE, IS-UNSATISFIABLE) with a given valuation, while using the compile procedure (COMPILE), we return a true or false terminal of type Node (TRUE-TERMINAL, FALSE-TERMINAL), respectively. Otherwise, we create a new node of type Node (CREATE-NODE) and proceed to conditioning f^e on the ith literal in the ordering, depending on evaluation depth i. The positive cofactor (lines 10–13) and negative cofactor (lines 15–17) are computed using both the theory- and SAT-solver. Although they have a comparable task (CONDITION), they perform it on different parts of the encoding, and in practice they are build, based on different design considerations. The theory-solver (lines 10, 15) is responsible for maintaining consistency with regard to the constraints f^c by having the SAT-solver additionally condition on the literals L_u contained in its unit clauses (GET-UNIT-LITERALS), i.e., implicit conditioning. The SAT-solver (lines 11, 16) is responsible for introducing weights W into the WPBDD at the appropriate places based on f^m. We obtain a reduced WPBDD by applying the reduction rules as defined by Definition 6: the merge rule (MERGE), as described by [51], and collapse rule (COLLAPSE) to nodes where both cofactors have been determined in order to obtain a reduced WPBDD.

Termination: From the fact that the input ordering O consists of literals upon which f^c essentially depends, we conclude that a satisfiability state will be reached by COMPILE for f^c through conditioning. Additionally, O contains all the literals, not symbolizing a weight, that f^m essentially depends on. Encoding $\mathcal{E}$ produces no dependencies among weights, thus conditioning f^e on all literals in O will produce a function containing only unit clauses, where each containing literal symbolizes a weight. CONDITION implicitly removes these weights (line 11), thus termination is guaranteed when a satisfiability state will be reached for $f^c \vee f^m$. CONDITION terminates because conditioning a function on literals contained in its unit clauses cannot produce additional unit clauses. It essentially just removes



(**a**) WPBDD (**b**) Logical circuit (**c**) Instantiated arithmetic circuit

Fig. 10. Probabilistic inference.

them. A return value in Condition has been ignored due to this reason. Particular return values have been ignored in Compile because conditioning f^e on negated literals guarantees that no unit clauses will be produced, as this only says that a variable is not equal to some value, having no implication to what value it actually equals.

6.2. Inference by weighted model counting

In order to perform inference by WMC, a WPBDD must be converted into an arithmetic circuit. Recall that a missing variable along a path implies the application of the distributive law, identifiable by using the variable ordering and support set $\mathcal{S}$ (Definition 6). The logical circuit induced by a WPBDD can easily be translated into an arithmetic circuit using the conversions in Table 1. Note that $x \vee y$ reduces to $x + y$, when x and y originate from the same dimension, i.e., $x, y \in \mathcal{A}(z)$, with $z \in X$.

One of the reasons for using the positive Shannon decomposition is to prevent constraints among variables to be represented twice in the described process of symbolic inference: once as part of the compiled representation, and again when we substitute literals with their appropriate weight in order to perform model counting. During this later phase, theory $\mathcal{T}$ is used to prohibit inconsistent network instantiations, preventing a state where multiple values are assigned to one variable. To perform inference, all weights ω_i are set to the probability they represent, and all other literals are set to 1. By conditioning $\mathcal{T}$ on the evidence using the theory-solver, literals are found that conflict with the evidence in the form of unit clauses. These must be set to 0.

Example 5. Let $f^c = (b_1 \vee b_2) \ \wedge \ (\overline{b_1} \vee \overline{b_2})$ represent the constraint clauses for variable b of Example 1. When computing $P(b_1)$ we condition f^c on evidence b_1 yielding $f^c_{b_1} = \overline{b_2}$, thus evidence b_1 implies $b_1 = 1$ ($true$) and $b_2 = 0$ ($false$). Fig. 10 shows each stage, from decision diagram to the instantiated arithmetic circuit with which we compute $P(b_1)$.

7. Optimizations

7.1. Encoding

The constraint clauses f^c of encoded function $\mathcal{E}(f) = f^e$, where f is defined over variables X, introduce predictable symmetries into the encoding (demonstrated by Example 3). By incorporating these constraints directly into the compilation process through theory $\mathcal{T}$, the constraint clauses f^c generated by Equation (2) and (3) become obsolete and can thus be removed from f^e. This reduces the number of clauses in the encoding by

$$\sum_{x \in X} \underbrace{1}_{\substack{ALO\\clause}} + \underbrace{\binom{n}{2}}_{\substack{AMO\\clauses}},$$

where we sum over every $x \in X$, with n the domain size of x, i.e., $|\mathcal{A}(x)|$. Both the at-least-once (AMO) and at-most-once (AMO) clauses contribute to reducing the number of clauses in the encoding to the number of probabilities in the CPTs

Table 1
Converting logical to arithmetic operator.

Logical	Arithmetic
x	x
$\overline{x}$	$(1 - x)$
$x \wedge y$	$x * y$
$x \vee y$	$x + y - x * y$

of the BN. This gives an advantage over related work using the direct encoding, as it puts less strain on the SAT-solver by requiring it to only process $\mathcal{M}(f)$.

7.2. Compiler

The compiler uses a lazy SMT-solver, consisting of a theory- and SAT-solver. It allows for optimization in a natural way, by providing the ability to substitute the SAT-solver with any other state-of-the-art solver.

The theory-solver is responsible for determining satisfiability with regard to constraint clauses f^c. This responsibility could be fulfilled by using a conventional SAT-solver. However, we have an optimized theory-solver, that supports *constant time conditioning* as opposed to quadratic time by taking advantage of the structure expressed by the constraint clauses. This has a major performance impact as the encoding consists of up to one third of constraint clause, as shown by the experimental results later. We have implemented it as follows: we require the function $V : \mathcal{A}(X) \to X$ that maps literal x_i back to x, where $x_i \in \mathcal{A}(x)$. For each x we maintain a counter that is initialized to the domain size of x, i.e., $|\mathcal{A}(x)|$. Conditioning on negated literal $\overline{x_i}$ will decrease the counter corresponding to x by 1. If the counter reaches 0, we derive contradiction (i.e., unsatisfiable as x has no value). Conditioning on positive literal x_i will cause any following conditioning on $x_j \in \mathcal{A}(x)\backslash x_i$ as redundant (i.e., x can only have one value). The SAT-solver will be bypassed completely as a result and the compiler will continue with the next variable in the ordering, saving additional time.

The SAT-solver is responsible for determining satisfiability with regard to clauses f^m. Although any SAT-solver can be used here, we have created a simplified SAT-solver by taking advantage of the structure of $\mathcal{M}(f)$. We use an one-to-many map $Q : \mathcal{A}(X) \to O$ from literal $l \in \mathcal{A}(X)$ to the clauses O it occurs in, i.e., $Q(l) = \{c^1, \ldots, c^n\}$. For each clause c^i, we maintain if it is satisfied with a counter, initialized to the number of literals it consists of. When conditioning on positive literal l we decrease counters associated with $Q(l)$ by 1. The clauses of which the counters have reached 0 are marked as satisfied, and their corresponding weights are set aside to be introduced into the representation later. Conditioning on negated literal $\bar{l}$ will mark clauses $Q(\bar{l})$ as satisfied. This is possible because $\mathcal{M}(f)$ only contains negated literals, and we are able to assume that $Q(l) \cap Q(\bar{l}) = \emptyset$. When all clauses in f^m are satisfied, we derive f to be satisfiable given the evaluated instantiation. In combination with the SAT-solver being bypassed in the case of the previously mentioned redundant variables, this allows for conditioning in *linear time*, in the number of clauses that l occurs in.

8. Experimental results

We have developed a tool chain, that can encode a Bayesian network into CNF, compile it to various different representations, and perform inference using the arithmetic circuits they induce. Using over 30 publicly available Bayesian networks, we provide empirical results on encoding size, representation size and compilation time comparisons to other well-known representations and compilers. We also compare the time it takes to perform exact inference compared to the classic junction tree algorithm.

Statistics related to the encoding are shown in Table 2, which includes Example 1 as BN `example`. The number of clauses produced by encoding $\mathcal{E}$ is equal to $C + P$, when disregarding determinism. We can reduce the size of the encoding by up to a third, by moving constraint clauses to the theory solver, additionally allowing us to perform constant time conditioning on them. We can also see that the majority of the BNs will benefit greatly using the techniques in this paper by looking at the amount of equal and deterministic probabilities they contain.

We have developed a compiler that supports compilation of Bayesian networks to OBDDs and ZBDDs (using the CUDD 3.0.0 library), SDDs [44] (using the SDD 1.1.1 library), and WPBDDs.[1,2,3] Each decision diagram is created with the same ordering, within the same framework, i.e., doing the same amount of work in the same order. Quite literally, the only differences are the inserted appropriate function calls to different libraries, and the output representation. This will have comparative implications to whether a particular compilation will succeed given resource constraints as time and memory. At the same time, we did not tune the algorithms to ensure that our algorithm stood out favorably, ensuring fair comparison.

The framework divides the compilation process in two for efficiency. The logical representation of each CPT is first compiled separately, and then conjoined to represent the full distribution. The latter is essential for producing a logical circuit with a consistent model count in order to perform for inference. All results regarding the WPBDD compiler have been produced with a hybrid approach, where CPTs are compiled in a top-down fashion, and conjoined bottom-up. We found that a fully bottom-up approach is only favorable when BNs have large CPTs like `mildew`, where we got a 5x speedup compared to the top-down approach. In practice, this implies that Bayesian networks can be compiled containing larger CPTs in general.

Many strategies were explored in order to find a good variable ordering for each BN. We used simulated annealing in combination with an upper bound function for networks, which by far yielded best results. The variable orderings were used to induce orderings based on literals, by saying that literal x_1 must come before y_1 if variable x comes before y in

[1] CUDD is available at http://vlsi.colorado.edu/~fabio/.

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

[3] The WPBDD compiler is available at https://github.com/gisodal/wmc/.

Table 2
Various statistics on BNs and their encoding, where the number of variables X, literals $\mathcal{A}(X)$, constraint clauses C, probabilities P, unique probabilites P^u (unique within a CPT) and the number of deterministic probabilities P^d are shown.

Bayesian network	X	$\mathcal{A}(X)$	C	P	P^u	P^d
example	2	5	6	7	5	2
cancer	5	10	10	20	20	0
earthquake	5	10	10	20	20	0
asia	8	16	16	36	27	8
survey	6	14	16	37	37	0
student farm	12	25	26	70	44	17
sachs	8	24	32	228	175	51
poker	7	43	145	748	71	653
child	20	60	93	344	161	4
carpo	54	122	139	554	246	266
powerplant	40	120	160	432	360	0
alarm	37	105	143	752	182	7
win95pts	76	152	152	1148	274	448
insurance	27	89	142	1419	427	372
andes	220	440	440	2308	652	146
hepar2	70	162	190	2139	1922	0
hailfinder	56	223	470	3741	835	587
pigs	441	1323	1764	8427	1474	4736
link	714	1793	2304	20462	1282	19586
water	32	116	188	13484	3578	7288
munin1	186	992	3522	19226	4323	12700
pathfinder	135	520	3600	106432	2379	56297
weeduk	15	90	347	22611	4600	216
fungiuk	15	165	1144	43007	8990	207
munin2	1003	5376	19460	83920	23228	53102
munin3	1041	5601	20292	85615	24495	53942
munin	1041	5651	20432	98423	24222	63112
munin4	1038	5645	20426	97943	24621	63112
mildew	35	616	17550	547158	14772	510060
mainuk	48	421	3607	130180	18883	3784
diabetes	413	4682	31738	461069	17888	360333
barley	48	421	3607	130180	36924	0

the variable ordering, where $x, y \in X$, $x_1 \in \mathcal{A}(x)$ and $y_1 \in \mathcal{A}(y)$. The weights are introduced into the ordering as literals precisely when the WPBDD would introduce them as edge weights.

Tables 3 and 4 show a comparative study between representation size and compilation time of supported representations, where WPBDD^{nc} is a WPBDD without application of the collapse rule, in order to show the impact of this rule. SDDs and SDD^{r}s are compiled using a balanced and right-aligned vtree ordering, respectively. A left-to-right traversal of these vtrees produces the ordering also used for the other representations. Table 3 indicates compilation failure due to a 24 Gb RAM memory limitation or a one hour time limit. Progress before failure is can be seen in Table 4. All experiments were run using an Intel Xeon E5620 CPU.

Table 3 shows a size comparison of each representation by the only size metric they have in common: the number of operators in the logical circuits that individual decision diagrams induce. We can see that WPBDDs have 60% less logical operators than the corresponding OBDDs on average at both stages of compilation, reducing inference time and system requirements considerably. Also, a WPBDD is reduced by 15% on average by applying the collapse rule when compiling CPTs, and 6% reduction on average with fully compiled networks. This statistic is fully determined by the amount of local structure in the BN and the ordering used during compilation, and can greatly be improved by utilizing techniques like dynamic compilation in the future.

Observe that there is a close relation between the size of OBDDs and SDD^{r}s, as mentioned in [44]. We can see that the size of each SDD^r is marginally smaller than its corresponding OBDD in Table 3. We assume that this is because SDDs have multi-valued logical-OR operators, which allow for more concise representations. SDDs consist of binary logical-AND, and n-ary logical-OR operators. We have included OR operators in size computations as $n-1$ binary logical-OR operators.

In order to evaluate the WMC approach to exact inference and other methods, we have chosen to compare to Ace (version 3.0) and to the junction tree algorithm using the publicly available Dlib library (version 18.18).[45] We also provide an indicative comparison to the HUGIN library (version 8.4).[6] Table 5 shows how much time is spent by each method on an identical set of probabilistic queries of the form $P(x|\boldsymbol{e})$. We have excluded time spent on reading or processing the Bayesian network, as well as creating join trees, purely focusing on inference time.

[4] Ace is available at http://reasoning.cs.ucla.edu/ace/.
[5] Dlib is available at http://dlib.net/.
[6] HUGIN is available at http://www.hugin.com/.

Table 3
Number of arithmetic operators in intermediate and resulting decision diagrams (symbols – and ∗ indicate compilation failure due to memory or a one hour time limitation, respectively).

Bayesian network	Number of operators per CPT						Total number of operators					
	WPBDD	WPBDD^{nc}	OBDD	ZBDD	SDD^{r}	SDD	WPBDD	WPBDD^{nc}	OBDD	ZBDD	SDD^{r}	SDD
example	**11**	19	48	54	33	49	**9**	15	39	30	27	49
cancer	**80**	**80**	195	747	135	292	**66**	**66**	159	324	144	362
earthquake	**80**	**80**	195	747	135	292	**66**	**66**	159	324	144	362
survey	**147**	**147**	354	1671	270	510	**132**	**132**	312	825	291	819
asia	**131**	136	321	1473	231	425	**136**	**136**	321	564	306	765
student_farm	**231**	252	591	3522	441	747	**459**	465	1098	2007	1080	2135
sachs	**747**	759	1788	20928	1611	3176	**630**	**630**	1602	13164	1581	4881
poker	**731**	1128	2373	6774	2289	4219	**912**	1095	2370	5394	2355	6082
child	**1011**	1099	2739	26088	2385	4663	**2934**	2988	7872	21861	7857	16030
carpo	**1257**	1386	3264	77049	2574	4514	**2499**	3015	7179	16233	7164	13405
powerplant	**1414**	1558	3795	86184	3141	6382	**4158**	4362	11043	36276	11025	26662
alarm	**1553**	1701	3795	41688	3312	6183	**3832**	4294	10008	19227	9993	35004
hepar2	**8371**	8467	18528	1593654	16101	29584	**56574**	56871	142806	2658189	142791	188453
weeduk	**29196**	30543	70863	13762830	69135	170330	**32114**	34832	109734	23840949	109455	–
fungiuk	**68430**	81435	295458	88536714	287940	250500	**234715**	251739	733551	95350635	727209	–
win95pts	**1948**	2020	4722	122244	3822	6571	**312191**	320183	743631	1415904	743619	426550
insurance	**2776**	3171	7455	106959	6810	11836	**468514**	474682	1263420	5465610	1263393	1731502
pathfinder	**23366**	40838	240915	6158832	237549	295272	**1899921**	2135180	5732988	29435283	5732976	2287777
hailfinder	**8909**	9520	25371	457860	23865	35707	11141550	11270157	31493220	137548032	31493157	**10508499**
water	**23498**	25458	56268	4776201	53979	81614	**18460995**	18561108	44005977	∗	–	–
mildew	**139386**	833215	1482432	56435169	–	832829	**21698281**	22322743	71621388	∗	–	–
andes	**4592**	5546	11667	1125528	9192	13809	–	–	–	–	–	–
mainuk	**232466**	240710	577866	216698319	567135	∗	–	–	∗	∗	–	–
barley	**244447**	252935	649902	581976891	637101	∗	–	–	∗	–	–	–
munin	**145258**	179744	435102	363002097	415134	723740	–	–	–	–	–	–
munin4	**145373**	180658	432672	345413400	413277	733401	–	–	∗	–	–	–
munin3	**140318**	172843	420537	327439035	400932	702203	∗	∗	–	–	–	–
diabetes	**186630**	288075	707679	111194403	696837	984197	–	–	–	–	–	–
pigs	**19953**	22482	49872	6976569	44127	68288	–	–	∗	–	–	–
munin1	**25415**	32188	74613	7756470	71538	131514	–	–	–	–	–	∗
link	**22869**	27249	60279	12387426	54291	85356	–	–	–	–	–	–
munin2	**133493**	165853	398550	323654088	379137	670702	–	–	–	–	–	–

Table 4
Compilation time in seconds. (In case of compilation failure, the percentage of successfully conjoined/processed variables is shown, of which the reason is documented in Table 3.)

Bayesian network	CPT compile time						Conjoin compile time					
	WPBDD	WPBDD^{nc}	OBDD	ZBDD	SDD^{r}	SDD	WPBDD	WPBDD^{nc}	OBDD	ZBDD	SDD^{r}	SDD
example	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
cancer	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
earthquake	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
survey	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
asia	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
student_farm	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	0.001	**0.000**	**0.000**	**0.000**	0.001
sachs	**0.000**	**0.000**	**0.000**	**0.000**	0.002	0.007	0.001	0.001	**0.000**	0.001	**0.000**	0.012
poker	**0.001**	**0.001**	0.002	0.002	0.015	0.030	0.002	0.004	**0.000**	**0.000**	0.001	0.016
child	**0.000**	**0.000**	0.001	**0.000**	0.004	0.009	0.004	0.004	**0.002**	0.007	0.010	0.041
carpo	**0.001**	**0.001**	0.003	0.002	0.006	0.008	0.005	0.006	**0.004**	0.012	0.018	0.033
powerplant	**0.001**	**0.001**	0.002	0.002	0.005	0.006	0.007	0.008	**0.003**	0.025	0.018	0.045
alarm	**0.001**	**0.001**	0.002	0.002	0.008	0.018	0.006	0.007	**0.003**	0.008	0.015	0.059
hepar2	0.006	**0.005**	0.016	0.063	0.055	0.097	**0.072**	**0.072**	0.201	7.045	1.192	3.047
weeduk	0.815	0.817	**0.249**	1.178	1.615	2512.160	0.084	0.112	**0.013**	3.988	0.077	(92.86%)
fungiuk	1.799	1.803	**1.027**	11.011	7.276	179.470	126.701	356.738	**0.264**	1285.111	1.455	(14.29%)
win95pts	**0.003**	**0.003**	**0.003**	0.005	0.014	0.016	**0.129**	0.132	1.257	4.203	6.293	0.882
insurance	**0.002**	**0.002**	0.004	0.005	0.020	0.022	**0.201**	0.211	0.366	3.070	1.946	2.485
pathfinder	0.685	**0.678**	7.840	8.066	30.405	2.226	**1.333**	1.501	13.974	101.344	67.038	22.888
hailfinder	**0.011**	**0.011**	0.018	0.035	0.105	0.177	5.556	**5.546**	17.409	567.997	71.913	9.464
water	0.107	0.107	**0.090**	0.253	0.764	4.922	**44.400**	45.883	55.055	(32.26%)	(96.77%)	(32.26%)
mildew	275.356	275.445	**129.703**	134.639	(11.43%)	1146.805	304.752	2062.466	**160.524**	(5.88%)	(0.00%)	(2.94%)
andes	**0.006**	**0.006**	0.007	0.032	0.027	0.041	(94.52%)	(94.52%)	(21.92%)	(21.00%)	(20.55%)	(21.92%)
mainuk	7.503	7.528	**2.272**	27.709	20.179	(4.17%)	(95.744%)	(95.74%)	(44.68%)	(8.51%)	(42.55%)	(0.00%)
barley	7.682	7.683	**2.247**	77.280	27.284	(4.17%)	(95.74%)	(93.62%)	(44.68%)	(2.13%)	(42.55%)	(0.00%)
munin	0.193	**0.190**	0.458	30.999	21.165	2.568	(75.67%)	(75.67%)	(1.83%)	(1.35%)	(1.73%)	(2.40%)
munin4	**0.186**	0.189	0.438	28.238	21.256	2.560	(78.98%)	(78.98%)	(1.54%)	(1.06%)	(1.35%)	(1.45%)
munin3	**0.165**	0.173	0.395	26.503	23.430	2.225	(74.23%)	(74.23%)	(2.50%)	(1.25%)	(1.63%)	(2.40%)
diabetes	3.142	**3.137**	6.479	13.272	106.197	11.352	(54.85%)	(54.85%)	(1.21%)	(0.73%)	(0.97%)	(1.21%)
pigs	**0.016**	**0.016**	0.036	0.228	0.248	0.114	(94.09%)	(94.09%)	(7.05%)	(6.59%)	(6.59%)	(8.18%)
munin1	0.037	**0.036**	0.101	0.322	0.998	0.416	(89.73%)	(89.73%)	(14.05%)	(11.89%)	(11.89%)	(12.97%)
link	**0.040**	**0.040**	0.070	0.621	0.512	0.403	(89.200%)	(89.20%)	(3.51%)	(2.95%)	(3.09%)	(3.23%)
munin2	**0.162**	**0.162**	0.400	28.014	19.650	2.255	(78.94%)	(78.94%)	(1.30%)	(0.60%)	(1.20%)	(1.098%)

Table 5
Inference time in seconds.

BN	Queries	WPBDD	OBDD	Ace	Dlib
example	17	**0.000**	0.000	0.015	0.001
cancer	714	**0.001**	0.002	0.735	0.138
earthquake	714	**0.001**	0.002	0.735	0.139
survey	4448	**0.014**	0.023	5.214	2.244
asia	18360	**0.053**	0.108	21.041	9.798
sachs	258324	**2.142**	5.323	503.670	1471.963
student_farm	1109885	**7.346**	17.942	1405.520	2627.214
poker	145999	**2.204**	5.229	238.969	3574.706
child	121647	**4.058**	14.603	331.598	3545.921
carpo	213257	**7.697**	21.340	500.063	3515.298
powerplant	457407	**21.575**	64.450	1398.680	3393.056
alarm	94034	**5.062**	13.324	297.079	3547.504
hepar2	26530	**16.419**	50.950	241.252	3463.509
weeduk	1051	**0.420**	1.608	20.541	3586.316
fungiuk	718	**1.911**	6.556	19.679	3567.136
win95pts	13595	**50.586**	135.639	75.531	3220.060
insurance	280	**1.391**	4.477	6.598	3571.825
pathfinder	765	**23.501**	78.277	30.369	3391.518
hailfinder	875	133.381	432.411	**20.879**	2448.653
water	2221	474.047	1319.978	**93.262**	$*$
mildew	1312	**290.731**	1401.271	387.664	$*$
Avg speedup			2.69	149.65	991.82

Remember that the computational complexity of inference by Weighted Model Counting (WMC) is linear in the size of the representation [13]. It implies that whenever we obtain a representation of smaller size, we will achieve faster runtimes. This relation between size and inference time is reflected in the empirical evaluations between WPBDDs and OBDDs, and can be inferred with regard to other representations like ZBDDs and SDDs.

Interesting is the comparison to Ace, which uses DNNF representations. It is known that OBDDs support more polytime queries than d-DNNFs [39]. This is partly reflected in the results of Table 5. There are two factors that are currently influencing the speedup and that compared to Ace could be improved. The compiler used to create the OBDDs and WPBDDs does not yet support exploitation of determinism, while Ace does. Depending on the amount of determinism (stated in Table 2) this can have a major impact. The other factor concerns the increasing complexity of finding good orderings as problem sizes grow. The ordering used during compilation directly influences the size of the representation and inference time, and improving methods for finding good orderings will become essential in future work.

We have also achieved a staggering speedup compared to the junction tree algorithm by Dlib, confirmed by an exceptional amount of cache misses reported by cachegrind (Valgrind tool), and other profile information by GNU Gprof and GNU Perf on resource usage. A comparison to HUGIN LITE over a subset of networks provides implications to performance for Bayesian networks of limited size, where we were able to achieve a speedup averaging at over 5x. Although this result provides intuitions on the performance using larger BNs, it should be considered purely indicative. Collectively, compilation and inference results show that WPBDDs make a valuable addition in the field of exact probabilistic inference.

9. Conclusion

To reduce the cost of Bayesian inference through Weighted Model Counting (WMC), we proposed a new canonical language called *Weighted Positive Binary Decision Diagrams* that represents probability distributions more concisely. We have provided theoretical results in addition to practical results on compilation size with regard to 30+ Bayesian networks, where we have seen WPBDD induced logical circuits reduced by 60% on average in comparison to OBDD induced circuits. The introduced reduction rule is responsible for a 15% reduction on average among compiled CPTs. These results can be improved even further in the future by finding a better variable ordering. This is made easier by WPBDDs, as they do not consider probabilities as auxiliary literals and reduce the search space considerably. We have evaluated the costs of inference compared to OBDD induced circuits, Ace, Dlib and Hugin. We were able to achieve a speedup of several orders of magnitude. The proposed language thus gives computational benefits during model counting as well as compilation.

Appendix A. Proofs

Lemma 1. *Let an* encoded *Boolean function $f^e : \{0,1\}^n \to \{0,1\}$ be defined over $\mathcal{A}(X)$, obtained by $\mathcal{E}(f)$, with f defined over X. We can rewrite f^e using the following equivalence:*

$$f^e = \; f^c \wedge \left(x_i \wedge f^e_{\|x_i} \;\vee\; f^e_{|\overline{x_i}} \right),$$

where $f^e = f^c \wedge f^m$, and $x_i \in \mathcal{A}(x)$, with $x \in X$.

Proof. We show that the lemma holds by reducing the Shannon expansion to the positive Shannon expansion through equivalence. Function f is defined over variables $X = \{x\}$, and encoded as Boolean function $f^e = f^c \wedge f^m$ given $\mathcal{E}$, where

$$\mathcal{A}(x) = \{x_1, \ldots, x_n\}, \qquad f^c = \left(\bigvee_{x_j \in \mathcal{A}(x)} \overline{x_j}\right) \wedge \left(\bigwedge_{x_k \in \mathcal{A}(x)} \bigwedge_{x_l \in \mathcal{A}(x) \setminus x_k} (\overline{x_k} \vee \overline{x_l})\right), \qquad f^m = 1.$$

Equation (2) and (3) show that f^c has a cardinality of at least two when encoding a non-trivial function, and that f^m essentially depends on a (non-strict) subset of variables on which f^c essentially depends. We therefore chose f^m to be simplistic to make the following reductions more intuitive. Note that the *at-most-once* clauses generated for x_i are subsumed by f^c at the final step.

$$\begin{aligned}
f^e &= f^c \wedge f^m \\
&= \underbrace{\left(\bigvee_{x_j \in \mathcal{A}(x)} x_j\right) \wedge \left(\bigwedge_{x_k \in \mathcal{A}(x)} \bigwedge_{x_l \in \mathcal{A}(x) \setminus x_k} (\overline{x_k} \vee \overline{x_l})\right)}_{f^c} \wedge \underbrace{1}_{f^m} \\
&= x_i \wedge \underbrace{\left(\bigwedge_{x_j \in \mathcal{A}(x) \setminus x_i} \overline{x_j}\right)}_{f^e_{|x_i}} \vee \overline{x_i} \wedge \underbrace{\left(\bigvee_{x_j \in \mathcal{A}(x) \setminus x_i} \overline{x_j}\right) \wedge \left(\bigwedge_{x_k \in \mathcal{A}(x) \setminus x_i} \bigwedge_{x_l \in \mathcal{A}(x) \setminus x_{\{ik\}}} (\overline{x_k} \vee \overline{x_l})\right)}_{f^e_{|\overline{x_i}}} \\
&\quad \text{Shannon expansion of } f^e \text{ on } x_i. \\
&= x_i \wedge \left(\bigwedge_{x_j \in \mathcal{A}(x) \setminus x_i} \overline{x_j}\right) \wedge \underbrace{1}_{f^e_{\|x_i}} \vee \overline{x_i} \wedge \underbrace{\left(\bigvee_{x_j \in \mathcal{A}(x) \setminus x_i} \overline{x_j}\right) \wedge \left(\bigwedge_{x_k \in \mathcal{A}(x) \setminus x_i} \bigwedge_{x_l \in \mathcal{A}(x) \setminus x_{\{ik\}}} (\overline{x_k} \vee \overline{x_l})\right)}_{f^e_{|\overline{x_i}}} \\
&\quad \text{Positive Shannon expansion of } f^e \text{ on } x_i\text{. Equivalent by Definition 3, and Equation (6).} \\
&= x_i \wedge \left(\bigwedge_{x_j \in \mathcal{A}(x) \setminus x_i} \overline{x_j}\right) \wedge f^e_{\|x_i} \vee \overline{x_i} \wedge f^e_{|\overline{x_i}} \\
&= \left(\bigwedge_{x_j \in \mathcal{A}(x) \setminus x_i} (\overline{x_i} \vee \overline{x_j})\right) \wedge \left(x_i \wedge f^e_{\|x_i} \vee f^e_{|\overline{x_i}}\right) \\
&= f^c \wedge \left(x_i \wedge f^e_{\|x_i} \vee f^e_{|\overline{x_i}}\right) \quad \square
\end{aligned}$$

Theorem 2. *Let an* encoded *Boolean function $f^e : \{0,1\}^n \to \{0,1\}$ be defined over $\mathcal{A}(X)$, obtained by $\mathcal{E}(f)$, with f defined over X. Equivalent under constraints f^c, we can rewrite f^e using:*

$$f^e \models x_i \wedge f^e_{\|x_i} \vee f^e_{|\overline{x_i}},$$

where $f^e = f^c \wedge f^m$, and $x_i \in \mathcal{A}(x)$, with $x \in X$.

Proof. We first show what models are introduced by one reduced positive Shannon expansion, providing clear implications what models are introduced by n expansions (i.e. a decomposition). We then show that f^c can be factored out of the positive Shannon decomposed function, and used to remove the introduced models, as they are subsumed by f^c.

Let function f and its encoding be defined as provided in the proof of Lemma 1. We will first show that applying one reduced positive Shannon expansion removes at-most-once (AMO) clauses related to the variable we expand.

$$\begin{aligned}
f^e &= f^c \wedge f^m \\
&= \underbrace{\left(\bigvee_{x_j \in \mathcal{A}(x)} x_j\right) \wedge \left(\bigwedge_{x_k \in \mathcal{A}(x)} \bigwedge_{x_l \in \mathcal{A}(x) \setminus x_k} (\overline{x_k} \vee \overline{x_l})\right)}_{f^c} \wedge \underbrace{1}_{f^m}
\end{aligned}$$

$$= x_i \wedge \underbrace{1}_{f^e_{\|x_i}} \vee \underbrace{\left(\bigvee_{x_j \in \mathcal{A}(x)\setminus x_i} \overline{x_j}\right) \wedge \left(\bigwedge_{x_k \in \mathcal{A}(x)\setminus x_i} \bigwedge_{x_l \in \mathcal{A}(x)\setminus x_{\{ik\}}} (\overline{x_k} \vee \overline{x_l})\right)}_{f^e_{|\overline{x_i}}}$$

$$= f^w$$

We have $M_j \models f^e$, with $j = \{1, \ldots, n\}$, where $M_j = \{\overline{x_1}, \ldots, \overline{x_{j-1}}, x_j, \overline{x_{j+1}}, \ldots, \overline{x_n}\}$ and $M_k \models f^w$, with $k = \{1, \ldots, (n-1) + 2^{(n-1)}\}$, where $M_k = \{\overline{x_1}, \ldots, \overline{x_{k-1}}, x_k, \overline{x_{k+1}}, \ldots, \overline{x_n}\}$ for $1 \le k \le n$ and $k \ne i$, and $M_k = \{\ldots, x_i, \ldots\}$ for the remainder. We conclude that $f^e \not\equiv f^w$, because f^e has n models, while f^w has $(n-1) + 2^{(n-1)}$. Expanding x_i has caused any model containing x_i to be true, i.e., model $\{\overline{x_1}, \ldots, \overline{x_{i-1}}, x_i, \overline{x_{i+1}}, \ldots, \overline{x_n}\}$ has changed to the models $\{\ldots, x_i, \ldots\}$. It is precisely those additional models in M_k that are not in M_j, which can be removed by AMO clauses created for x_i, i.e.:

$$f^e \equiv \left(\bigwedge_{x_j \in \mathcal{A}(x)\setminus x_i} (\overline{x_i} \vee \overline{x_j})\right) \wedge f^w.$$

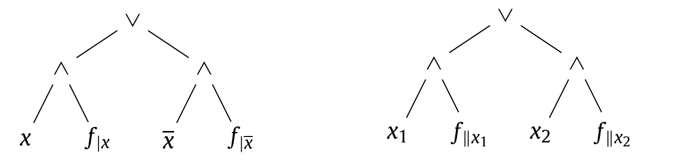
Where one expansion on x_i removes AMO clauses related to x_i, a decomposition clearly removes AMO clauses related to all variables $\mathcal{A}(x)$. This holds regardless of ordering, as a positive Shannon decomposition of f^e is guaranteed to produce an isomorphic representation due to the symmetric nature of the constraints. We can reduce the positive Shannon expansion to its reduced by form factoring out f^c:

$$\begin{aligned}
f^w &= f^c \wedge f^m \\
&= \underbrace{\left(\bigvee_{x_j \in \mathcal{A}(x)} \overline{x_j}\right) \wedge \left(\bigwedge_{x_k \in \mathcal{A}(x)} \bigwedge_{x_l \in \mathcal{A}(x)\setminus x_k} (\overline{x_k} \vee \overline{x_l})\right)}_{f^c} \wedge \underbrace{1}_{f^m} \\
&= f^c \wedge \left(x_1 \wedge \underbrace{1}_{f^e_{\|x_1}} \vee \underbrace{\left(\bigvee_{x_j \in \mathcal{A}(x)\setminus x_1} \overline{x_j}\right) \wedge \left(\bigwedge_{x_k \in \mathcal{A}(x)\setminus x_1} \bigwedge_{x_l \in \mathcal{A}(x)\setminus x_{\{1k\}}} (\overline{x_k} \vee \overline{x_l})\right)}_{f^e_{|\overline{x_1}}} \right) \\
&= f^c \wedge (x_1 \vee f^c \wedge (x_2 \wedge \underbrace{1}_{f^e_{\|x_2}} \vee \underbrace{\ldots}_{f^e_{|\overline{x_2}}})) \\
&= f^c \wedge (x_1 \vee f^c \wedge (x_2 \vee f^c \wedge (x_3 \wedge \underbrace{1}_{f^e_{\|x_3}} \vee \underbrace{\ldots}_{f^e_{|\overline{x_3}}}))) \\
&= \ldots \\
&= f^c \wedge (x_1 \vee \ (f^c \wedge (x_2 \vee (f^c \wedge \ldots x_{n-1} \vee (f^c \wedge x_n))))) \\
&= f^c \wedge (x_1 \vee \ldots \vee x_n) \\
&= f^c
\end{aligned}$$

This confirms that, when using reduced positive Shannon decompositions, equivalence is only achieved under domain closure constraints, as AMO clauses are subsumed by f^c. □

Proposition 1. *An OBDD representing Boolean function g and an PBDD representing $\mathcal{E}(g)$ induce isomorphic logical circuits under Boolean identity, given an appropriate ordering.*

Proof. Let $\mathcal{E}(g) = g^e$ be a Boolean function, with g a Boolean function defined over variables $X = \{x^1, \ldots, x^n\}$. We show that an OBDD representing g is equivalent to a PBDD representing $\mathcal{E}(g)$ by comparing their induced logical circuits, under the premise that the collapse rule does not apply distribution, but deletes literals. Note that this comparison requires g to be a Boolean function, and thus each $x \in X$ is mapped to two atoms $\mathcal{A}(x) = \{x_1, x_2\}$. For every Shannon expansion on g, there is an equivalent positive Shannon expansion on g^e (Fig. 11).



(a) Standard expansion on x **(b)** Positive expansion on $\mathcal{A}(x) = \{x_1, x_2\}$

Fig. 11. Expansions.

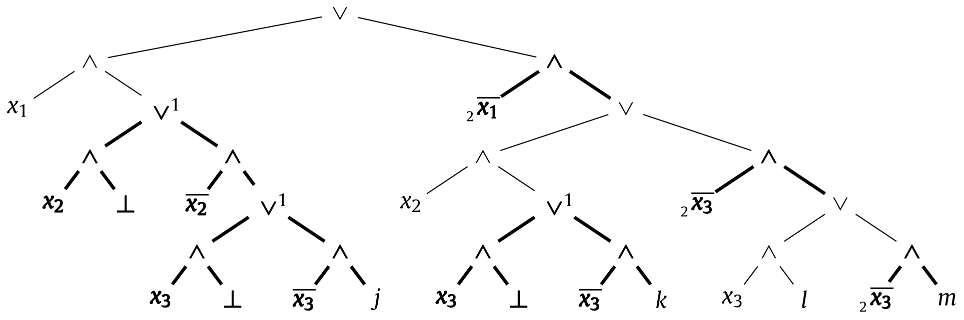


Fig. 12. Constraints in OBDD ψ.

We have equality, because there exists a trivial mapping between the two decomposition types, namely $x = x_1$ and $\overline{x} = x_2$. Observe the implication that the collapse rule can be applied to g^e whenever the delete rule can be applied to g. Under our pretense we can infer that the OBDD and PBDD induce equivalent circuits, because they are isomorphic due to a trivial mapping for precisely those orderings where, for each $x \in X$, atoms $\mathcal{A}(x) = \{x_1, x_2\}$ are placed adjacent in the order, e.g., $\mathcal{A}(x^1) < \ldots < \mathcal{A}(x^n)$ and for each $x \in X$ we have partial orders $x_1 < x_2$. The delete rule alters the circuit if cofactors are equal by applying the distributive law and identity, e.g., $x \wedge g \;\vee\; \overline{x} \wedge g = (x \vee \overline{x}) \wedge g = g$. The collapse rule forgoes that last step. We therefore conclude that the OBDD and PBDD induce equivalent circuits under Boolean identity. $\square$

Proposition 2. *Given an ordering on $\mathcal{A}(X)$, the size of PBDD φ is less than the size of OBDD ψ when they both represent $\mathcal{E}(f) = f^e$, where f is defined over variables X.*

Proof. In the case where f is not Boolean, we will show that PBDD φ is always smaller than OBDD ψ, when they both represent $\mathcal{E}(f) = f^e$. Consider some $x \in X$, for which we find atoms $\mathcal{A}(x) = \{x_1, x_2, x_3, \ldots, x_n\}$ adjacent in the ordering, where n is the domain size of x. OBDD ψ will contain the subfunction shown in Fig. 12.

Here, j, k, l and m are subfunctions that essentially depend on $\{x_4, \ldots, x_n\}$. We can transition from the OBDD induced logical circuit above to the corresponding logical circuit induced by a PBDD, by removing the bold edges, literals and operators. Shannon expansions that have indicator 1 at their root can be removed, which coincides with implicit conditioning. More generally, this allows us to remove $\sum_{i=1}^{n}(i - 1)$ nodes from the OBDD. Additionally, the implicates for negative cofactors are removed that are marked by indicator 2.

Removing the implicate for the negative cofactor together with implicit conditioning, contribute to reducing the size of induced logical circuits by removing operators while maintaining equivalence. The extent to which is lower bounded by:

$$\underbrace{\sum_{x \in X} n}_{\substack{negative\\ cofactor}} + \underbrace{\sum_{i=1}^{n}(i-1) * 3}_{\substack{implicit\\ conditioning}},$$

where we sum over every $x \in X$, with n the domain size of x, i.e., $|\mathcal{A}(x)|$. This increases when atoms $\mathcal{A}(x)$ are not adjacent in the ordering, and is multiplied by the number of distinct subfunctions in OBDD ψ that depend on $\mathcal{A}(x)$. Furthermore, we are guaranteed to encounter each $x \in \mathcal{A}(X)$ in an OBDD along every path from root to the *true* terminal, which serves as an upper bound regarding PBDDs. The collapse rule can effectively reduce the size of the representation if the structure of f^e allows it, by reducing the number of nodes from $|\mathcal{A}(pa(x))|$ to $|pa(x)|$ for every subfunction, provided that literals $\mathcal{A}(y)$, with $y \in pa(x)$, for each parent are adjacent in the ordering. $\square$

References

[1] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, 1988.
[2] D. Heckerman, J. Breese, Causal independence for probabilistic assessment and inference using Bayesian networks, IEEE Trans. Syst. Man Cybern. Syst. 26 (1996) 826–831.
[3] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context-specific independence in Bayesian networks, in: Proceedings of the International Conference on Uncertainty in Artificial Intelligence, 1996, pp. 115–123.
[4] F. Jensen, S. Anderson, Approximations in Bayesian belief universe for knowledge based systems, arXiv:1304.1101.
[5] F. Bacchus, S. Dalmao, T. Pitassi, DPLL with caching: a new algorithm for #SAT and Bayesian inference, in: Proceedings of Electronic Colloquium on Computational Complexity, vol. 10, 2003.
[6] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (2008) 772–799.
[7] M. Wachter, R. Haenni, Logical compilation of Bayesian networks with discrete variables, in: Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2007, pp. 536–547.
[8] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.-J. Hwang, Symbolic model checking: 10^{20} states and beyond, in: Proceedings of the Fifth Symposium on Logic in Computer Science, 1990, pp. 428–439.
[9] R.D. Shachter, B. D'Ambrosio, B. Del Favero, Symbolic probabilistic inference in belief networks, in: Proceedings of Advancement of Artificial Intelligence, vol. 90, 1990, pp. 126–131.
[10] C.W. Barrett, R. Sebastiani, S.A. Seshia, C. Tinelli, Satisfiability modulo theories, in: Handbook of Satisfiability, vol. 185, 2009, pp. 825–885.
[11] F. Somenzi, CUDD: CU Decision Diagram Package Release 3.0.0, Tech. rep., Department of Electrical Computer, and Energy Engineering, University of Colorado at Boulder, 2015.
[12] A. Choi, D. Kisa, A. Darwiche, Compiling probabilistic graphical models using sentential decision diagrams, in: Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2013, pp. 121–132.
[13] M. Bozga, O. Maler, On the representation of probabilities over structured domains, in: Proceedings of Computer Aided Verification, 1999, pp. 261–273.
[14] N.J. Nilsson, Probabilistic logic, Artif. Intell. 28 (1986) 71–87.
[15] J.Y. Halpern, An analysis of first-order logics of probability, Artif. Intell. 46 (1990) 311–350.
[16] D. Poole, First-order probabilistic inference, in: Proceedings of the International Joint Conference on Artificial Intelligence, vol. 3, 2003, pp. 985–991.
[17] F. Bacchus, S. Dalmao, T. Pitassi, Algorithms and complexity results for #SAT and Bayesian inference, in: Proceedings of the 44th Symposium on Foundations of Computer Science, 2003, pp. 340–351.
[18] T. Walsh, SAT vs CSP, in: Principles and Practice of Constraint Programming, 2000, pp. 441–456.
[19] O. Bailleux, Y. Boufkhad, Efficient CNF encoding of Boolean cardinality constraints, in: Proceedings of the Principles and Practice of Constraint Programming, 2003, pp. 108–122.
[20] T. Tanjo, N. Tamura, M. Banbara, A compact and efficient SAT-encoding of finite domain CSP, in: Theory and Applications of Satisfiability Testing-SAT, 2011, pp. 375–376.
[21] M. Gavanelli, The log-support encoding of CSP into SAT, in: Principles and Practice of Constraint Programming, 2007, pp. 815–822.
[22] S.-i. Minato, K. Satoh, T. Sato, Compiling Bayesian networks by symbolic probability calculation based on Zero-suppressed BDDs, in: Proceedings of the International Joint Conference on Artificial Intelligence, 2007, pp. 2550–2555.
[23] H. Poon, P. Domingos, Sum-product networks: a new deep architecture, in: Proceedings of the International Conference on Computer Vision Workshops, 2011, pp. 689–690.
[24] T. Sang, P. Beame, H.A. Kautz, Performing Bayesian inference by weighted model counting, in: Proceeding of Advancement of Artificial Intelligence, vol. 5, 2005, pp. 475–481.
[25] N.L. Zhang, D. Poole, On the role of context-specific independence in probabilistic inference, in: Proceedings of the International Joint Conference on Artificial Intelligence, vol. 85, 1999.
[26] A. Cano, S. Moral, A. Salmeron, Penniless propagation in join trees, Int. J. Intell. Syst. 15 (2000) 1027–1059.
[27] A.V. Kozlov, D. Koller, Nonuniform dynamic discretization in hybrid networks, in: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, 1997, pp. 314–325.
[28] A. Cano, M. Gómez-Olmedo, S. Moral, C.B. Pérez-Ariza, Recursive probability trees for Bayesian networks, in: Current Topics in Artificial Intelligence, 2009, pp. 242–251.
[29] M. Jaeger, Probabilistic decision graphs, combining verification and AI techniques for probabilistic inference, Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 12 (2004) 19–42.
[30] C.P. Gomes, A. Sabharwal, B. Selman, Model Counting, 2008.
[31] P. Beame, R. Impagliazzo, T. Pitassi, N. Segerlind, Memoization and DPLL: formula caching proof systems, in: Proceedings of the 18th Conference on Computational Complexity, 2003, pp. 248–259.
[32] S.M. Majercik, M.L. Littman, Using caching to solve larger probabilistic planning problems, in: Proceedings of Advancement of Artificial Intelligence, 1998, pp. 954–959.
[33] E. Fischer, J.A. Makowsky, E.V. Ravve, Counting truth assignments of formulas of bounded tree-width or clique-width, Discrete Appl. Math. 156 (2008) 511–529.
[34] P. Sen, A. Deshpande, L. Getoor, PrDB: managing and exploiting rich correlations in probabilistic databases, VLDB J. 18 (2009) 1065–1090.
[35] K. Kersting, B. Ahmadi, S. Natarajan, Counting belief propagation, in: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, 2009, pp. 277–284.
[36] W. Li, P. Poupart, P. Van Beek, Exploiting causal independence using weighted model counting, in: Proceeding of Advancement of Artificial Intelligence, 2008, pp. 337–343.
[37] P. Sen, A. Deshpande, L. Getoor, Read-once functions and query evaluation in probabilistic databases, Proc. VLDB Endow. 3 (2010) 1068–1079.
[38] R. Mateescu, R. Dechter, R. Marinescu, AND/OR multi-valued decision diagrams (AOMDDs) for graphical models, J. Artif. Intell. Res. 33 (2008) 465–519.
[39] A. Darwiche, Decomposable negation normal form, J. Assoc. Comput. Mach. 48 (2001) 608–647.
[40] T.D. Nielsen, P.-H. Wuillemin, F.V. Jensen, U. Kjærulff, Using ROBDDs for inference in Bayesian networks with troubleshooting as an example, in: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, 2000, pp. 426–435.
[41] E.M. Clarke, K.L. McMillan, X. Zhao, M. Fujita, J. Yang, Spectral transforms for large Boolean functions with applications to technology mapping, in: Proceedings of the 30th Conference on Design Automation, 1993, pp. 54–60.
[42] Y.-T. Lai, S. Sastry, Edge-Valued Binary Decision Diagrams for multi-level hierarchical verification, in: Proceedings of the 29th Design Automation Conference, 1992, pp. 608–613.
[43] S. Sanner, D. McAllester, Affine Algebraic Decision Diagrams (AADDs) and their application to structured probabilistic inference, in: Proceedings of the International Joint Conference on Artificial Intelligence 2005, 2005, pp. 1384–1390.
[44] A. Darwiche, SDD: a new canonical representation of propositional knowledge bases, in: Proceedings of the International Joint Conference on Artificial Intelligence, vol. 22, 2011, p. 819.

[45] S.-i. Minato, Zero-suppressed BDDs for set manipulation in combinatorial problems, in: Proceedings of the 30th Conference on Design Automation, 1993, pp. 272–277.

[46] H.H. Hoos, SAT-encodings, search space structure, and local search performance, in: Proceedings of the International Joint Conference on Artificial Intelligence, vol. 99, 1999, pp. 296–303.

[47] C.E. Shannon, A symbolic analysis of relay and switching circuits, Electr. Eng. 57 (12) (1938) 713–723.

[48] R.E. Bryant, Symbolic Boolean manipulation with ordered binary decision diagrams, ACM Comput. Surv. 24 (1992) 293–318.

[49] R.E. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Trans. Comput. 100 (1986) 677–691.

[50] B. Bollig, I. Wegener, Improving the variable ordering of OBDDs is NP-complete, IEEE Trans. Comput. 45 (1996) 993–1002.

[51] K.S. Brace, R.L. Rudell, R.E. Bryant, Efficient implementation of a BDD package, in: Proceedings of the Design Automation Conference, 1991, pp. 40–45.