Abstract

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, in order to represent local structure expressed among conditional probabilities more efficiently. We show that further improvements are possible, by exploiting the knowledge that is lost during the encoding phase and 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 \textit{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.

Keywords: knowledge compilation, probabilistic inference, weighted model counting, Bayesian networks, binary decision diagrams.

1. Introduction

Bayesian networks, BNs for short, have been a subject of great interest partly due to their contribution in solving real-life problems that involve uncertainty. Bayesian networks are probabilistic graphical models that represent joint probability distributions concisely by factoring them into conditional probabilities based on independence assumptions, in order to perform inference more efficiently \cite{1}. Further representational and computational advances have been made by exploiting causal independence \cite{2}, as well as contextual independence \cite{3} and determinism \cite{4} expressed in conditional probability tables (CPTs). In order to capture these independencies local to CPTs, Bayesian networks have been represented as weighted Boolean formulas \cite{5, 6}, reducing inference to \textit{Weighted Model Counting} (WMC), or \textit{weighted \#SAT} \cite{5}. By representing a Bayesian network as Boolean formula $f$ in \textit{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 \cite{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 \cite{8}. Symbolic inference unifies the work of probabilistic inference and the extensive research done in the field of model checking, verification and satisfiability \cite{9}. \textit{Ordered Binary Decision Diagrams} (OBDDs) are based on Shannon decompositions and have been a very
influential symbolic representation that reduces compilation to the problem of finding the variable ordering resulting in the optimal factoring.

We focus on the disadvantage of the approaches in recent work that encode a BN as an independent CNF $f$, motivated by the ability to use off-the-shelf SAT-solvers. While maintaining this ability, we exploit the knowledge that is lost during the encoding without requiring this independence.

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 decompositions, 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), namely a lazy SMT-solver $[10]$. It provides the means to view constraints among variables in the encoding as background theory $T$ which supports the SAT-solver, allowing constant time conditioning. We compile the conditional probability tables of a BN explicitly, but leave implicit the domain closure implied by the encoding. This approach allows us to remove by 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 a corresponding OBDD circuits at 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 an average speedup between 5 and 1000 times compared to different implementations of the Junction Tree algorithm, making WPBDDs a valuable addition to the field of exact probabilistic inference.

After preliminaries and background (Section 3), we introduce WPBDDs (Section 4). The process of using a BN to perform exact inference by WMC is explained (Section 5) in addition to its optimization (Section 6). We conclude with experimental results (Section 7), and review achievements (Section 8). First we summarize related work.

2. 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, e.g., BNs represent a joint probability distribution as a multiplicative factorization, exploiting independencies among variables. Further improvements to this factorization have been made by using propositional and first order logic $[13, 14, 15]$. 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 $[16]$.

A BN can be viewed as a constraint satisfaction problem (CSP) and translated to a satisfiability (SAT) instance in conjunctive normal form (CNF), a form commonly used in satisfiability solving. Various encodings of CSPs have been proposed, namely log, direct $[17]$, order $[18]$, compact order $[19]$, log-support encoding $[20]$, etc. In the context of BNs, a probability distribution can be considered a pseudo Boolean function $f : \{0,1\}^n \rightarrow \mathbb{R}$, with arity $n$, which can be uniquely written as an exponentially sized multi-linear polynomial
Others have used the direct encoding \[6\], or a combination between the direct and order encoding \[23\], where a BN is viewed as a set of discrete real valued functions, where each function represents a distinct CPT.

Inference by WMC is motivated by linear time complexity in the size of the representation \[24\], where the common goal is to exploit local structure \[25\]. Choosing a representation or language to compile to is therefore a critical task, where one must deal with the balance between the functions a language can represent concisely versus its algorithmic properties. 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 \[23\]. 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) \[12\], Zero-suppressed Binary decision diagrams (ZBDD) \[21\] and Ordered Binary Decision Diagrams (OBDD) \[39\], which view probabilities as auxiliary literals, resulting in an intractably large search space. Multi-Terminal BDDs \[40\] represent multi-valued functions, but would require too many terminal nodes considering the size of a probability distribution. Variants of Edge-Value BDDs \[41, 42\] focus on real valued functions. When multiple CPTs have probabilities in common, we lose the ability to distinguish from which CPT the probabilities originated. 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 edges weights symbolically. This differentiates our approach from Multi-Terminal BDDs \[40\] and Edge-Value BDDs \[41\]. And unlike SDDs \[43\], 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 \[44\]. 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 upcoming discussions.

3. Preliminaries and Background

We provide here a description of what Bayesian networks are and introduce a running example (Section 3.1), show how to encode a BN onto the Boolean domain (Section 3.2), and describe an influential representation that will serve for comparison with ours (Section 3.3).

3.1. Bayesian Networks

Probabilistic inference is an important computational problem in Artificial Intelligence. A full joint probability distribution defined over \(n\) Boolean variables is of size \(O(2^n)\). Finding
the minimal representation of a function describing a probability distribution reduces memory and inference complexity, which is the motivation for this paper.

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 an 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 \(O(n^2 k)\), where \(k\) is the maximum number of parents of any node.

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

![Figure 1 Bayesian network with local structure](image)

Figure 1 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 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 that represents 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 about reformulating into a different language, it is also about finding the minimal representation given that language. The goal is to derive an arithmetic circuit corresponding to that representation with improved complexity compared to the standard factorization.

### 3.2. 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 \(f : X \rightarrow \mathbb{R}\). Using a weighted adaptation of the sparse or direct
encoding \[6, 17, 15\], we encode \(f\) into a Boolean function \(E(f) = f^e\) by representing it as a weighted CNF:

\[
E(f) = f^c \land f^m, \tag{1}
\]

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

### 3.2.1. Encoding Constraints

The mapping function \(M\) introduces for each \(x \in X\) atoms \(A(x) = \{x_1, \ldots, x_n\}\), 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:

\[
(x_1 \lor \cdots \lor x_n) \tag{2}
\]

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

\[
\bigwedge_{i=1}^{n} \left( x_i \implies \bigwedge_{j \in A(x) \setminus \{x_i\}} \overline{x}_j \right) = \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\overline{x}_i \lor \overline{x}_j), \tag{3}
\]

where \(\overline{x}_i\) indicates the negation of \(x_i\).

### 3.2.2. Encoding CPTs

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

\[
(x \land u^1 \land \cdots \land u^r \Rightarrow \omega_i) = (x \lor \overline{u}^1 \lor \cdots \lor \overline{u}^r \lor \omega_i), \tag{4}
\]

which we henceforth shall view as a weighted clause \((x \lor \overline{u}^1 \lor \cdots \lor \overline{u}^r) : \omega_i\), where its symbolic weight \(\omega_i\) represents probability \(P(x|U)\). We introduce a symbolic weight for every unique probability per CPT, and thus allow multiple models to be associated with it:

\[
I^1 \land \cdots \land I^s, \tag{5}
\]

where each \(I^j\) has weight \(\omega_i\) and forms the implicate of model \(m\) associated with \(P(x|U)\), i.e., \(P = \omega_i : (x \land u^1 \land \cdots \land u^r)\), where the variables \(x \cup U\) have been appropriately mapped by \(M\), i.e., \(x \in A(x)\) and each \(u^k \in A(u^k)\).

**Example 2.** Assume a BN as given in Example [7]. The following clauses form \(f^c\):

<table>
<thead>
<tr>
<th>Variable</th>
<th>ALO (Eq. 2)</th>
<th>AMO (Eq. 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>((a_1 \lor a_2))</td>
<td>((\overline{a}_1 \lor \overline{a}_2))</td>
</tr>
<tr>
<td>(b)</td>
<td>((b_1 \lor b_2 \lor b_3))</td>
<td>((\overline{b}_1 \lor \overline{b}_2) \land (\overline{b}_1 \lor \overline{b}_3) \land (\overline{b}_2 \lor \overline{b}_3))</td>
</tr>
</tbody>
</table>

The variables that make up the search space during compilation therefore are \(A(\{a, b\}) = \{a_1, a_2, b_1, b_2, b_3\}\). We encode equal probabilities \(P(x|U)\) as unique symbolic weights per CPT:
In accordance with Equations 4 and 5, \( f^m \) consists of the following clauses, accompanied by their respective symbolic weights:

\[
\begin{align*}
&\left( a_1 \right) : \omega_1 \land \left( a_2 \right) : \omega_2 \\
&\left( a_1 \lor b_1 \right) : \omega_2 \land \left( a_1 \lor b_2 \right) : \omega_2 \land \left( a_1 \lor b_1 \right) : \omega_2 \\
&\left( a_1 \lor b_2 \right) : \omega_2 \land \left( a_1 \lor b_3 \right) : \omega_3 \land \left( a_2 \lor b_3 \right) : \omega_3.
\end{align*}
\]

3.3. 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 \( x_4 \) is defined as the projection:

\[
f_{|x_4 = b}(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n),
\]

with \( b \in \{0, 1\} \). We will use shorthand notations \( f_{|x_4} \) and \( f_{|\overline{x}_4} \) for \( f_{|x_4 = 1} \) and \( f_{|x_4 = 0} \), respectively. Shannon’s theorem is used to find a more compact way to represent \( f \) by factoring it.

**Theorem 1.** [46] Shannon’s expansion allows a Boolean function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) defined over variables \( X \), to be written in terms of its inputs:

\[
f = x \land f_{|x} \lor \overline{x} \land f_{|\overline{x}},
\]

with \( x \in X \) and where \( f_{|x} \) is called the positive cofactor of \( f \) with respect to \( x \), and \( f_{|\overline{x}} \) the negative cofactor. Applying Shannon’s theorem is known as an (additive) decomposition step. The decomposition of \( f \) is defined as the recursive application of Shannon’s expansion theorem to cofactors, removing one variable at a time, until no variables are left.

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) [46].

**Definition 1.** [48] 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 decomposition on variable \( \text{var}(v) \in X \). 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 it is commonly used in comparisons with related work.
4. Weighted Positive Binary Decision Diagrams

Consider variable $x$ and its corresponding mapped atoms $A(x) = \{x_1, x_2\}$. Decision diagrams that are based on Shannon decompositions produce an unnecessarily large arithmetic circuit by redundantly representing constraints $f^c$ provided as part of encoding $E$. 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).

4.1. 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 $x_i$ we are guaranteed, due to constraints $f^c$, to obtain unit clauses containing negated literals $\overline{x_j}$, with $x_j \in A(x) \setminus x_i$ (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 $b$:

$$f^c = (b_1 \lor b_2 \lor b_3) \land (\overline{b_1} \lor \overline{b_2}) \land (\overline{b_1} \lor b_3) \land (b_2 \lor \overline{b_3})$$

According to Shannon’s expansion the following holds:

$$f^c = b_1 \land f^c_{|b_1} \lor \overline{b_1} \land f^c_{|\overline{b_1}}$$

Now specifically look at the unit clauses that result from conditioning on positive literal $b_1$, i.e., instantiating $b_1$ and performing unit propagation.

$$f^c_{|b_1} = (1 \lor b_2 \lor b_3) \land (0 \lor \overline{b_2}) \land (0 \lor \overline{b_3}) \land (b_2 \lor \overline{b_3})$$

$$= (1) \land (\overline{b_2}) \land (\overline{b_3}) \land (b_2 \lor \overline{b_3})$$

$$= (1) \land (\overline{b_2}) \land (\overline{b_3}) \land (\overline{b_3})$$

Thus, conditioning on $b_i$ will result in unit clauses containing negated literals $\overline{b_j}$, with $b_j \in A(b) \setminus \{b_i\}$.

We distinguish between two types of conditioning based on the previous observation, and describe them in the following definition.
Definition 2. Let $f^e$ be an encoded representation of function $f$, given encoding $E$, where $f$ is defined over variables $X$. We define $f^e_{\parallel x_i}$ as the conditioning of $f^e$ on literals $\{x_i, \overline{x_1}, \ldots, \overline{x_{i-1}}, x_{i+1}, \ldots, x_n\}$ in any order, i.e., as the explicit conditioning of $f^e$ on literal $x_i \in A(x)$, and its implicit conditioning on literals $x_j \in A(x) \setminus x_i$, with $x \in X$:

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

It follows from Definition 2 and the constraints provided by encoding $E$, that the relation between $f^e_{\parallel x_i}$ and $f^e_{|x_i}$ is given by the following equality:

$$f^e_{|x_i} = \left( \bigwedge_{x_j \in A(x) \setminus x_i} \overline{x_j} \right) \land f^e_{\parallel x_i}. \quad (7)$$

Implicit conditioning on unit clauses takes advantage of deterministic behavior expressed in $f^c$, while other representations would have to explicitly 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 $T$. As will be shown later, by separating constraint clauses from $f^e$ as theory $T$ allows them 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 with our variant on the Shannon expansion, introduced in the following section.

4.2. 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. A positive Shannon expansion allows an encoded Boolean function $E(f) = f^e$, where $f$ is defined over $X$, to be written in terms of its inputs:

$$f^e = f^c \land \left( x_i \land f^e_{\parallel x_i} \lor f^e_{\parallel \overline{x_i}} \right),$$

where $f^e = f^c \land f^m$, and $x_i \in A(x)$, with $x \in X$. The positive cofactor $f^e_{\parallel x_i}$ incorporates implicit conditioning (Definition 2). The negative cofactor $f^e_{\parallel \overline{x_i}}$ and the decomposition of $f^e$ are as defined by Theorem 1.

As intuition might confirm, logical representations will grow by the reintroduction of constraints at every expansion. We introduce a reduced expansion 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. A reduced positive Shannon expansion allows an encoded Boolean function $E(f) = f^e$, where $f$ is defined over $X$, to be written in terms of its inputs under constraints $f^c$:

$$f^e \models x_i \land f^e_{\parallel x_i} \lor f^e_{\parallel \overline{x_i}},$$

where $f^e = f^c \land f^m$, and $x_i \in A(x)$, with $x \in X$. Cofactors and decomposition are as defined by Lemma 1.
By using the reduced form of the expansion, we are able represent constraints more concisely in corresponding logical circuits (Figure 2), as well as in the soon to be introduced representation that utilizes it.

\[
\begin{array}{cc}
\text{(a) Shannon Logical Circuit} & \text{(b) Positive Shannon Logical Circuit} \\
\end{array}
\]

Figure 2 Logical Circuits

Using encoding $E$ we can infer that the delete rule used to reduce OBDDs will never be applied to constraint clauses $f^e$. OBDDs are therefore not capable of capturing local structure along one dimension. 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 3. A positive OBDD (PBDD) represents Boolean function $E(f) = f^e$, 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 $S$, 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 arithmetic circuit, a trick similarly utilized with Zero-Suppressed BDDs (ZBDD) [44]. Note that a Boolean function $E(f) = f^e$, where $f$ is defined over variables $X$, essentially depends on $\mathcal{A}(X)$, because $f^e$ 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 apply its reduction rules reversely.

Figure 3 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 $\{x_1, x_2\}$. The positive Shannon decomposition does not only reduce the size 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.
Figure 3 From BDDs to logical circuits

Figure 3 shows that there are functions where the corresponding minimal PBDD and OBDD differ exponentially in size. Not represented in the figure, the collapse rule additionally removes the nodes that share the same positive cofactor with their parents (Figure 4).

The semantics of nodes whose child has been removed changes. A missing node, inferable by the support set $S$ 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 that the delete rule can however 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 why 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 allow for more fine 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 $f$ and an PBDD representing $\mathcal{E}(f)$ induce isomorphic logical circuits under Boolean identity, given an appropriate ordering.

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

10
4.3. Adding Probabilities as Weights

Encoding $E$ 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 $\omega$ 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.

The collapse rule previously introduced can easily be extended for the non-binary and weighted case as shown in Figure 5.

![Figure 5](image)

**Figure 5** Application of the collapse rule, where functions $f_{x_i}$ and weights $\omega_i = \omega_j$.

**Definition 4.** A weighted PBDD (WPBDD) representing Boolean function $E(f) = f^e$, where $f$ is defined over variables $X$, is a PBDD where each node $v$ is a tuple $\langle x_i, W, f_{x_i} \rangle$ that represents a weighted reduced positive Shannon expansion:

$$f^e = x_i \land (W \land f_{x_i}) \lor f_{\overline{x_i}},$$

where $x_i \in A(X)$, and $W$ is a conjunction of weights $\omega_i$ that correspond to $f_{x_i}$. Positive and negative cofactors are as described by Lemma 4. 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 $W \land f_{x_i} = W \land f_{x_j}$, where $\text{var}(v) = x_i$ and $\text{var}(u) = x_j$, with $x_i, x_j \in A(x)$ and $x \in X$.

**Example 4.** Consider the CPTs from Example 4, where per CPT, equal probabilities are represented by unique symbolic weights $\omega_i$.

| $P(a_1)$ | $P(a_2)$ | $P(b_1|a)$ | $P(b_2|a)$ | $P(b_3|a)$ |
|----------|----------|-------------|-------------|-------------|
| $1$      | $2$      | $\omega_1$ | $\omega_2$ | $\omega_3$ |

Figure 6 shows the minimization of a WPBDD using variable ordering $a_1 < a_2 < b_1 < b_2 < b_3$, that represents the BN with 3 probabilities instead of 8. Figure 7 shows the comparison of this WPBDD with an OBDD representing the same function, given variable ordering $a_1 < a_2 < a_3 < b_1 < b_2 < b_3 < \omega_2 < \omega_3$, which results in a minimal OBDD that obeys the partial ordering used for the WPBDD.
5. Symbolic Inference

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

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
<td>&quot;Encoding $E$&quot;</td>
</tr>
<tr>
<td>Counter</td>
<td>&quot;Theory $T + f^e$&quot;</td>
</tr>
<tr>
<td>Input</td>
<td>$f$</td>
</tr>
<tr>
<td>Output</td>
<td>$\wp$BD $\varphi$</td>
</tr>
<tr>
<td>$P(x</td>
<td>e)$</td>
</tr>
</tbody>
</table>

A BN represented by $f$ is first encoded by the encoder as Boolean function $f^e$ using encoding $E$ as defined in Section 3.2. The encoder also provides background theory $T$ representing $f^e$, i.e., the constraints among variables that support mapping $M$.

The compiler uses a lazy SMT-solver to record evaluation paths and as a WPBDD, given $f^e$ and theory $T$. A lazy SMT-solver combines a SAT-solver with a theory-solver (or $T$-solver) for some theory $T$. Traditionally, the role of a theory-solver is to purely report back on the satisfiability of $T$. We have extended the theory-solver to provide more information in order to support implicit conditioning in the SAT-solver.
The counter computes the probability of $x$ given evidence $e$, by translating the provided WPBDD into an arithmetic circuit and using the extended capabilities of the $T$-solver to properly instantiate the variables.

5.1. Compilation

The order of decomposition greatly influences representation size. Compilation therefore reduces to finding the optimal variable ordering. Satisfiability (SAT) is key during compilation. Normally, when CNF $f$ contains an empty clause we derive a contradiction. Note that if all contradicting clauses are weighted, we supersede the contradiction and introduce their weights into our representation at the corresponding edge. In order to obtain a minimal BDD representation, the search space of all variable instantiations is traversed with a DPLL-style algorithm, in order to find partial instantiations that describe equal (sub)functions, as depicted in Figure 8.

Algorithm 1 shows a Depth-first search/dynamic programming (DFS+DP) approach that uses a lazy SMT-solver (solver) to compile $f^e$ into a WPBDD, given some ordering on the variables (ordering), where $\top$ and $\bot$ denote the terminal nodes representing true and false, respectively.

The lazy SMT-solver is unique in the sense that there exists a link beyond satisfiability feedback between the $T$-solver and the SAT-solver (solver.theory and solver.sat, respectively.)
The traditional role of the theory-solver is to solely report back on satisfiability. To implement implicit conditioning, we have extended it to also provide its unit clauses, that are used by the SAT-solver to further condition $f^e$. This connection is possible because $f^e$ and theory $T$ both essentially depend on the same variables. The SMT-solver reports $f^e$ to be satisfiable only when both the theory and SAT-solver agree on this. Storing intermediate states as an undo mechanism for the solver is infeasible, thus it has the ability to dynamically undo any conditioning (solver.undo).

The compiler uses the satisfiability state of the SMT-solver (solver.state) to build the WPBDD and achieves a canonical form by applying to each subfunction the merge rule (as describe by [49]) and collapse rule (apply_merge_rule(n) and apply_collapse_rule(n), respectively).

5.2. Inference by Weighted Model Counting

In order to perform inference by WMC, a WPBDD must be converted into a logical (refactored) form using Definition 4. Recall that a missing variable along a path implies the use of the distributive law, identifiable by using the variable ordering and support set $S$. The logical form can easily be translated into an arithmetic circuit according to Table 1. Note that $x \lor y$ reduces to $x + y$, when $x$ and $y$ originate from the same dimension, i.e., $x, y \in A(z)$, with $z \in X$. 

---

**Algorithm 1 Compiler**

```
struct node {
    literal l;
    node * t, * e;
    set W;
};

eenum satisfy_t {
    satisfiable = 0,
    unsatisfied = 1,
    unsatisfiable = 3
};

satisfy_t solver::condition(literal l){
    solver.theory.condition(l);
    solver.sat.condition(l);
    solver.sat.condition(solver.theory.unit_clauses());
    return solver.theory.state() | solver.sat.state();
}

node * condition(literal l)
    solver.condition(l);
    if (not negated(l))
        n->W += solver.sat.weights();

switch(solver.state()){
    case unsatisfied:
        n = compile(new node,i+1);
        apply_collapse_rule(n);
        apply_merge_rule(n);
        return n;
    case satisfiable:
        n = T;
        break;
    case unsatisfiable:
        n = ⊥;
        break;
}

node* compile(n,i=0){
    n->l = ordering[i];
    n->t = condition(n->l);
    n->e = condition(not n->l);
    apply_collapse_rule(n);
    apply_merge_rule(n);
    return n;
}

wpbdd* compiler(T,f)e{
    solver.theory.init(T);
    solver.sat.init(fm);
    return compile(new node);
}
```
Logical | Arithmetic
---|---
x | x
\(\overline{x}\) | \((1 - x)\)
x ∧ y | \(x * y\)
x ∨ y | \(x + y - x * y\)

Table 1 Converting logical to arithmetic operator

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 \(T\) is used to prohibit inconsistent network instantiations, preventing a state where multiple values are assigned to one variable. To perform inference, all weights \(\omega_i\) are set to the probability they represent, and all other literals are set to 1. By conditioning \(T\) 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 \(f^c = (a_1 \lor a_2) \land (\overline{a_1} \lor \overline{a_2})\) represent the constraint clauses for variable \(a\) of Example 4. When computing \(P(a_1)\) we condition \(f^c\) on evidence \(a_1\) yielding \(f^c_{a_1} = \overline{a_2}\), thus evidence \(a_1\) implies \(a_1 = 1\) (true) and \(a_2 = 0\) (false). This process is shown in Figure 9.

![Figure 9 Probabilistic inference](image)

6. Optimizations

6.1. Encoding

The constraint clauses \(f^c\) of encoded function \(E(f) = f^c\), 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 \(T\), the constraint clauses \(f^c\) generated by Equation 2 and 3 become obsolete and can thus be removed from \(f^c\). This reduces the number of clauses in the encoding by:
\[
\sum_{x \in X} 1 + \binom{n}{2} ALO \text{ clause } + AMO \text{ clauses },
\]
where we sum over every \( x \in X \), with \( n \) the domain size of \( x \), i.e., \( |A(x)| \). 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 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 \( M(f) \).

6.2. Compiler

The compiler uses a lazy SMT-solver, consisting of a theory- and SAT-solver. In the way we have build the compiler, it naturally allows for optimization by providing the ability to substitute the SAT-solver with any other state-of-the-art solver. We have optimized the SMT-solver by using the structure expressed by the encoding, and the fact that theory \( T \) and \( f^e \) are defined over the same variables.

We have optimized the theory-solver such that it now supports \textit{constant time conditioning} of constraint clauses, which can take up one third of the encoding as shown by experimental results later. All one needs is the function \( V: A(X) \to X \) that maps literal \( x_i \) back to \( x \), where \( x_i \in A(x) \). For each \( x \) we maintain a counter that is initialized to the domain size of \( x \), i.e., \( |A(x)| \). 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 \( x_j \in A(x) \setminus x_i \) 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.

We have simplified the SAT-solver considerably by taking advantage of the structure of \( M(f) \). We use an one-to-many map \( Q: A(X) \to O \) from literal \( l \in A(X) \) 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 \( \overline{l} \) will mark clauses \( Q(\overline{l}) \) as satisfied. This is possible because \( M(f) \) only contains negated literals, and we are able to assume that \( Q(l) \cap Q(\overline{l}) = \emptyset \). 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 \textit{linear time}, in the number of clauses that \( l \) occurs in.

7. 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
Bayesian Network | $X$ | $A(X)$ | $C$ | $P$ | $P^u$ | $P^d$
--- | --- | --- | --- | --- | --- | ---
example | 2 | 5 | 6 | 8 | 3 | 5
Cancer | 5 | 10 | 20 | 20 | 0
Earthquake | 5 | 10 | 20 | 20 | 0
Asia | 8 | 16 | 36 | 27 | 9
Survey | 6 | 14 | 37 | 37 | 9
Student Farm | 12 | 25 | 70 | 44 | 26
Sachs | 8 | 24 | 228 | 175 | 53
Poker | 7 | 43 | 748 | 71 | 677
Child | 20 | 60 | 344 | 161 | 183
Carpo | 54 | 122 | 554 | 246 | 308
Powerplant | 40 | 120 | 432 | 360 | 72
Alarm | 37 | 105 | 752 | 182 | 570
Win95pts | 76 | 152 | 1148 | 274 | 874
Insurance | 27 | 89 | 1419 | 427 | 992
Andes | 220 | 440 | 2308 | 652 | 1656
Hepar2 | 70 | 162 | 2139 | 1922 | 217
Hailfinder | 56 | 223 | 3741 | 835 | 2906
Pigs | 441 | 1323 | 8427 | 1474 | 6953
Link | 714 | 1793 | 20462 | 1282 | 19180
Water | 32 | 116 | 13484 | 3578 | 9906
Mumin1 | 186 | 992 | 19226 | 4323 | 14903
Pathfinder | 135 | 520 | 106432 | 2379 | 104053
Weeduk | 15 | 90 | 22611 | 4600 | 18011
Funguuk | 15 | 165 | 43007 | 8990 | 34017
Mumin2 | 1003 | 5376 | 83920 | 23228 | 60662
Mumin3 | 1041 | 5601 | 85615 | 24495 | 61120
Mumin | 1041 | 5651 | 98423 | 24222 | 74201
Mumin4 | 1038 | 5645 | 97943 | 24621 | 73322
 Mildew | 35 | 616 | 547158 | 14772 | 532386
Mainuk | 48 | 421 | 130180 | 18883 | 111297
Diabetes | 413 | 4682 | 461069 | 17888 | 443181
Barley | 48 | 421 | 3607 | 36924 | 93256

Table 2 Various statistics on BNs and their encoding, where the number of variables $X$, literals $A(X)$, constraint clauses $C$, probabilities $P$, cumulative amount of unique probabilities per CPT $P^u$ and the number of deterministic probabilities $P^d$ are shown.

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 include Example 1 as BN example. The number of clauses produced by encoding $E$ is equal to $|f^c| + |P|$, for constraint clauses $f^c$ and mapping $M$, 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 by 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\footnote{Available at http://vlsi.colorado.edu/~fabio/} 3.0.0 library), SDDs \footnote{Available at http://reasoning.cs.ucla.edu/sdd/} (using the SDD\footnote{Available at http://reasoning.cs.ucla.edu/sdd/} 1.1.1 library), and WPBDDs. 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 later 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 topdown fashion, and conjoined bottom-up. We found that a fully bottomup approach is only favorable when BNs have large CPTs like mildew, where we got a 5x speedup compared to the topdown approach. In practice, large CPTs are usually avoid as they increase the complexity of inference.

Many strategies were explored in order to find a good variable ordering for each BN. Using simulated annealing in combination with an upper bound function yielded best results by far. 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 the variable ordering, where \( x, y \in X, x_1 \in A(x) \) and \( y_1 \in A(y) \). 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\textsuperscript{nc} is a WPBDD where the collapse rule has not been applied, in order to show the impact that this rule has. SDDs and SDD\textsuperscript{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\textsuperscript{3} indicates compilation failure due to a 24Gb RAM memory limitation or an one hour time limit by symbols - and *, respectively. The progress each failed compilation made before is indicated in Table\textsuperscript{4}. All experiments were run using an Intel Xeon E5620 CPU.

Table 3 shows a size comparison of each representation by the only common size metric in the logical circuit that each decision diagram induces. 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 upon utilizing techniques as dynamic compilation in the future.

Observe that there is a close relation between then size of OBDDs and SDD\textsuperscript{r}’s, as mentioned in \textsuperscript{43}. We can see that the size of each SDD\textsuperscript{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.
<table>
<thead>
<tr>
<th>Bayesian Network</th>
<th>Number of operators per CPT</th>
<th>Total number of operators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WPBDD</td>
<td>WPBDD&quot;gc&quot;</td>
</tr>
<tr>
<td>example</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>cancer</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>earthquake</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>survey</td>
<td>147</td>
<td>147</td>
</tr>
<tr>
<td>asia</td>
<td>131</td>
<td>136</td>
</tr>
<tr>
<td>student_farm</td>
<td>231</td>
<td>252</td>
</tr>
<tr>
<td>sachs</td>
<td>747</td>
<td>759</td>
</tr>
<tr>
<td>poker</td>
<td>731</td>
<td>1128</td>
</tr>
<tr>
<td>child</td>
<td>1011</td>
<td>1099</td>
</tr>
<tr>
<td>carpo</td>
<td>1257</td>
<td>1386</td>
</tr>
<tr>
<td>powerplant</td>
<td>1414</td>
<td>1558</td>
</tr>
<tr>
<td>alarm</td>
<td>1553</td>
<td>1701</td>
</tr>
<tr>
<td>hepar2</td>
<td>8371</td>
<td>8467</td>
</tr>
<tr>
<td>weeduk</td>
<td>29196</td>
<td>30543</td>
</tr>
<tr>
<td>fangiuk</td>
<td>68430</td>
<td>81435</td>
</tr>
<tr>
<td>win95pts</td>
<td>1948</td>
<td>2020</td>
</tr>
<tr>
<td>insurance</td>
<td>2776</td>
<td>3171</td>
</tr>
<tr>
<td>pathfinder</td>
<td>23366</td>
<td>40838</td>
</tr>
<tr>
<td>hailfinder</td>
<td>8909</td>
<td>9520</td>
</tr>
<tr>
<td>water</td>
<td>23498</td>
<td>25458</td>
</tr>
<tr>
<td>mildew</td>
<td>139386</td>
<td>833215</td>
</tr>
<tr>
<td>andes</td>
<td>4592</td>
<td>5546</td>
</tr>
<tr>
<td>mainuk</td>
<td>232466</td>
<td>240710</td>
</tr>
<tr>
<td>barley</td>
<td>244447</td>
<td>252935</td>
</tr>
<tr>
<td>minlin</td>
<td>145258</td>
<td>179744</td>
</tr>
<tr>
<td>minlin4</td>
<td>145373</td>
<td>180658</td>
</tr>
<tr>
<td>minlin3</td>
<td>140318</td>
<td>172843</td>
</tr>
<tr>
<td>diabetes</td>
<td>186630</td>
<td>288075</td>
</tr>
<tr>
<td>pigs</td>
<td>19953</td>
<td>22482</td>
</tr>
<tr>
<td>minlin1</td>
<td>25415</td>
<td>32188</td>
</tr>
<tr>
<td>link</td>
<td>22869</td>
<td>27249</td>
</tr>
<tr>
<td>minlin2</td>
<td>133493</td>
<td>165853</td>
</tr>
</tbody>
</table>

Table 3 Number of arithmetic operators in intermediate and resulting decision diagrams
(symbols - and * indicate compilation failure due to memory or an one hour time limitation, respectively)
<table>
<thead>
<tr>
<th>Bayesian Network</th>
<th>CPT compile time</th>
<th>Conjoin compile time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WPBDD</td>
<td>WPBDD\textsuperscript{ac}</td>
</tr>
<tr>
<td>example</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>cancer</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>earthquake</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>survey</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>asia</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>student_farm</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>sachs</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>poker</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>child</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>carpo</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>powerplant</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>alarm</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>hepar2</td>
<td>0.006</td>
<td>0.005</td>
</tr>
<tr>
<td>weeduk</td>
<td>0.815</td>
<td>0.817</td>
</tr>
<tr>
<td>fungui</td>
<td>1.799</td>
<td>1.803</td>
</tr>
<tr>
<td>win95pts</td>
<td>0.003</td>
<td>0.003</td>
</tr>
<tr>
<td>insurance</td>
<td>0.002</td>
<td>0.002</td>
</tr>
<tr>
<td>pathfinder</td>
<td>0.685</td>
<td>0.678</td>
</tr>
<tr>
<td>hailfinder</td>
<td>0.011</td>
<td>0.011</td>
</tr>
<tr>
<td>water</td>
<td>0.107</td>
<td>0.107</td>
</tr>
<tr>
<td>mildew</td>
<td>275.356</td>
<td>275.445</td>
</tr>
<tr>
<td>andes</td>
<td>0.006</td>
<td>0.006</td>
</tr>
<tr>
<td>mainuk</td>
<td>7.503</td>
<td>7.528</td>
</tr>
<tr>
<td>barley</td>
<td>7.682</td>
<td>7.683</td>
</tr>
<tr>
<td>munin</td>
<td>0.193</td>
<td>0.190</td>
</tr>
<tr>
<td>munin4</td>
<td>0.186</td>
<td>0.189</td>
</tr>
<tr>
<td>munin3</td>
<td>0.165</td>
<td>0.173</td>
</tr>
<tr>
<td>diabetes</td>
<td>3.142</td>
<td>3.137</td>
</tr>
<tr>
<td>pigs</td>
<td>0.016</td>
<td>0.016</td>
</tr>
<tr>
<td>munin1</td>
<td>0.037</td>
<td>0.036</td>
</tr>
<tr>
<td>link</td>
<td>0.040</td>
<td>0.040</td>
</tr>
<tr>
<td>munin2</td>
<td>0.162</td>
<td>0.162</td>
</tr>
</tbody>
</table>

**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.)
In order to evaluate the WMC approach to exact inference with other methods, we have chosen to compare to the Junction tree algorithm using the publicly available Dlib\textsuperscript{3} C++
library (version 18.18), and the HUGIN\textsuperscript{4} library (through the C++ API version 8.4). We
exhaustively go through all possible probabilistic queries. Table 5 and 6 show how much time
spent by each method on an identical set of queries. We have excluded time spent on
reading or processing the Bayesian network, as well as creating the join tree, purely focusing
on inference time. We went through all possible queries up to poker, and limited others
by a reasonable amount of time. Time \( T_{\text{total}} \) indicates the total time spent on compilation
and inference, and shows that it occasionally depends on how many queries you intend to
answer which language must be chosen to get results faster. In theory, the speedup \( S \) of
WPBDDs vs other logical representations coincides with the sizes difference of the arithmetic
circuits they induce (see Table 3), as inference has linear complexity in the size of induced
circuits. This is confirmed with an average speedup of over 2.6x compared to OBDDs. We
achieved an average speedup of over 5x compared to HUGIN (Note that we were not able to
process all BNs as we used the LITE (free) version, which comes with limitations). We 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. Collectively, compile

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
Bayesian & Queries & WPBDD & OBDD & Dlib \\
Network & & \( T \) & \( T_{\text{total}} \) & \( T \) & \( T_{\text{total}} \) & \( S \) & \( T \) & \( S \) \\
\hline
example & 17 & 0.000 & 0.000 & 0.000 & 0.000 & 1.20 & 0.001 & 34.70 \\
cancer & 714 & 0.001 & 0.001 & 0.002 & 0.002 & 1.57 & 0.138 & 96.33 \\
earthquake & 714 & 0.001 & 0.001 & 0.002 & 0.002 & 1.50 & 0.139 & 96.53 \\
survey & 4448 & 0.014 & 0.014 & 0.023 & 0.023 & 1.70 & 2.244 & 165.40 \\
asia & 18360 & 0.053 & 0.053 & 0.108 & 0.108 & 2.03 & 9.798 & 185.70 \\
sachs & 258324 & 2.142 & 2.143 & 5.323 & 5.323 & 2.50 & 1471.963 & 687.23 \\
student\_farm & 1109885 & 7.346 & 7.347 & 17.942 & 17.943 & 2.47 & 2627.214 & 357.73 \\
poker & 145999 & 2.204 & 2.209 & 5.229 & 5.231 & 2.37 & 3574.706 & 1621.99 \\
powerplant & 457407 & 21.575 & 21.584 & 64.450 & 64.477 & 2.99 & 3393.056 & 157.27 \\
alarm & 94034 & 5.062 & 5.07 & 13.324 & 13.334 & 2.63 & 3545.921 & 873.81 \\
hepar2 & 26530 & 16.419 & 16.496 & 50.950 & 50.981 & 3.10 & 3463.509 & 210.95 \\
weeduk & 1051 & 0.420 & 1.349 & 1.608 & 5.845 & 3.83 & 3586.316 & 8538.88 \\
fungiu & 718 & 1.911 & 360.452 & 6.556 & 1292.7 & 3.43 & 3567.136 & 1866.32 \\
win95pts & 13595 & 50.586 & 50.721 & 135.639 & 139.845 & 2.68 & 3220.060 & 63.65 \\
insurance & 280 & 1.391 & 1.604 & 4.477 & 7.551 & 3.22 & 3571.825 & 2567.81 \\
pathfinder & 765 & 23.501 & 25.68 & 78.277 & 87.461 & 3.33 & 3391.518 & 144.32 \\
hailfinder & 875 & 133.381 & 138.938 & 432.411 & 1000.43 & 3.24 & 2448.653 & 18.36 \\
water & 2221 & 474.047 & 520.037 & 1319.978 & 1352.33 & 2.78 & * & * \\
mildew & 1312 & 290.731 & 2628.64 & 1401.271 & 1536.85 & 3.59 & * & * \\
\hline
Avg Speedup & & & 2.69 & & & 991.82 & & \\
\hline
\end{tabular}
\caption{Inference time in seconds.}
\end{table}

\textsuperscript{3}Available at http://dlib.net/
\textsuperscript{4}http://www.hugin.com
Table 6 Inference time in seconds.

and inference results show that WPBDDs make a valuable addition in the field of exact probabilistic inference.

8. 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 represent 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 responsbly for a 15% reduction on average among compiled CPTs. These results can be improved upon even further in the future by finding a better variable ordering, which is made easier by WPBDDs, as they do not consider probabilities as auxiliary literals, reducing the search space considerably. We have evaluated the cost of inference compared to OBDD induced circuits, yielding a 2.5x speedup on average, and to the Junction tree algorithm, approaching a speedup of 1000x. The language thus gives computational benefits during model counting as well as compilation.

References

References


Appendices

**Theorem 1.** Shannon’s expansion allows a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ defined over variables $X$, to be written in terms of its inputs:

$$f = x \land f|_x \lor \overline{x} \land f|_{\overline{x}},$$

with $x \in X$ and where $f|_x$ is called the positive cofactor of $f$ with respect to $x$, and $f|_{\overline{x}}$ the negative cofactor. Applying Shannon’s theorem is known as an (additive) decomposition step. The decomposition of $f$ is defined as the recursive application of Shannon’s expansion theorem to cofactors, removing one variable at a time, until no variables are left.

**Proof.** See [46].

**Lemma 1.** A positive Shannon expansion allows an encoded Boolean function $E(f) = f^e$, where $f$ is defined over $X$, to be written in terms of its inputs:

$$f^e = f^c \land \left(x_i \land f^e|_{x_i} \lor f^e|_{\overline{x_i}}\right),$$

where $f^e = f^c \land f^m$, and $x_i \in A(x)$, with $x \in X$. The positive cofactor $f^e|_{x_i}$ incorporates implicit conditioning (Definition 2). The negative cofactor $f^e|_{\overline{x_i}}$ and the decomposition of $f^e$ are as defined by Theorem 1.

**Proof.** We show that the theorem 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 $f^e = f^c \land f^m$ given $E$, where:

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

It follows from Equation 2 and 3 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 that $f^c$ essentially depends on. 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.

$$f^e = f^c \land f^m$$

$$= \left(\bigvee_{x_j \in A(x)} x_j\right) \land \left(\bigwedge_{x_k \in A(x)} \bigwedge_{x_l \in A(x) \setminus x_k} (\overline{x_k} \lor x_l)\right) \land 1 \quad \underbrace{f^c}_{f^m}$$
\[ f^e = x_i \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \lor \bar{x}_i \land \left( \bigvee_{x_j \in A(x) \setminus x_i} x_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]

Shannon expansion of \( f^e \) on \( x_i \).

\[ f^e = x_i \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \bar{x} \land f_{\bar{x}}^e \]

Positive Shannon expansion of \( f^e \) on \( x_i \). Equivalent by Definition 3 and Equation 7.

\[ f^e = x_i \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land f_{\bar{x}}^e \land f_{\bar{x}}^e \]

\[ f^e = x_i \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} x_k \lor x_i \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]

Theorem 2. A reduced positive Shannon expansion allows an encoded Boolean function \( E(f) = f^e \), where \( f \) is defined over \( X \), to be written in terms of its inputs under constraints \( f^e \):

\[ f^e \models x_i \land f_{x_i}^e \lor f_{\bar{x}}^e, \]

where \( f^e = f^c \land f^m \), and \( x_i \in A(x) \), with \( x \in X \). Cofactors and decomposition are as defined by Lemma 4.

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 decom- position). 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 \( f^e \).

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.

\[ f^e = f^c \land f^m \]

\[ f^e = x_i \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]

\[ f^e = f_{x_i}^e \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]

\[ f^e = f_{x_i}^e \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]

\[ f^e = f_{x_i}^e \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]

\[ f^e = f_{x_i}^e \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]

\[ f^e = f_{x_i}^e \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]

\[ f^e = f_{x_i}^e \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]

\[ f^e = f_{x_i}^e \land \left( \bigwedge_{x_j \in A(x) \setminus x_i} \bar{x}_j \right) \land \left( \bigwedge_{x_k \in A(x) \setminus x_i} \left( x_k \lor x_i \right) \right) \land 1 \]
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 $1 \leq k \leq n$ and $k \neq i$, 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.:

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

Where one expansion on $x_i$ removes AMO clauses related to $x_i$, a decomposition clearly removes AMO clauses related to all variables $A(x)$. 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 $f^c$:

$$f^w = f^c \wedge f^m$$

$$= \left( \bigvee_{x_j \in A(x)} \overline{x_j} \right) \wedge \left( \bigwedge_{x_k \in A(x) \setminus x_k} \bigwedge_{x_l \in A(x) \setminus x_l} (\overline{x_k} \lor x_l) \right) \wedge 1$$

$$= f^c \wedge \left( x_1 \wedge 1 \vee \left( \bigvee_{x_j \in A(x) \setminus x_1} \overline{x_j} \right) \wedge \left( \bigwedge_{x_k \in A(x) \setminus x_1} \bigwedge_{x_l \in A(x) \setminus x_1 \setminus x_{1k}} (\overline{x_k} \lor x_l) \right) \right)$$

$$= f^c \wedge (x_1 \vee f^c \wedge (x_2 \wedge 1 \vee \ldots))$$

$$= f^c \wedge (x_1 \vee f^c \wedge (x_2 \vee f^c \wedge (x_3 \wedge 1 \vee \ldots)))$$

$$= \ldots$$

$$= f^c \wedge (x_1 \vee (f^c \wedge (x_2 \vee (f^c \wedge \ldots x_{n-1} \vee (f^c \wedge x_n)))))$$

$$= f^c \wedge (x_1 \vee \ldots \vee x_n)$$

$$= f^c$$

This confirms when using reduced positive Shannon decompositions, equivalence is only achieved under domain closure constraints, as AMO clauses are subsumed by $f^c$. \(\square\)

**Proposition 1.** An OBDD representing Boolean function $f$ and an PBDD representing $E(f)$ induce isomorphic logical circuits under Boolean identity, given an appropriate ordering.
Proof. Let $E(f) = f^e$ be a Boolean function, with $f$ a boolean function defined over variables $X = \{x^1, \ldots, x^n\}$. We show that an OBDD representing $f$ is equivalent to a PBDD representing $E(f)$ by comparing their induced logical circuits, under the premiss that the collapse rule does not apply distribution, but deletes literals. Note that this comparison requires $f$ to be a Boolean function, and thus each $x \in X$ is mapped to two atoms $A(x) = \{x_1, x_2\}$. For every Shannon expansion on $f$, there is an equivalent positive Shannon expansion on $f^e$ (Figure 10).

$$\begin{align*}
\text{Standard expansion on } x & \quad \text{Positive expansion on } A(x) = \{x_1, x_2\}
\end{align*}$$

Figure 10 Expansions

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 $f^e$ whenever the delete rule can be applied to $f$. 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 $A(x) = \{x_1, x_2\}$ are placed adjacent in the order, e.g., $A(x^1) < \ldots < A(x^n)$ 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., $x \land f \lor \overline{x} \land f = (x \lor \overline{x}) \land f = f$. The collapse rule forgoes that last step. We therefore conclude that the OBDD and PBDD induce equivalent circuits under Boolean identity.

Proposition 2. Given an ordering on $A(X)$, the size of PBDD $\varphi$ is less than the size of OBDD $\psi$ when both representing $E(f) = f^e$, where $f$ is defined over variables $X$.

Proof. In the case where $f$ is not Boolean, we will show that PBDD $\varphi$ is always smaller than OBDD $\psi$, when both representing $E(f) = f^e$. Consider some $x \in X$, for which we find atoms $A(x) = \{x_1, x_2, x_3, \ldots, x_n\}$ adjacent in the ordering, where $n$ is the domain size of $x$. OBDD $\psi$ will contain the subfunction shown in Figure 11.

$$\begin{align*}
\text{Figure 11 Constraints in OBDD } \psi
\end{align*}$$
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, and 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:

\[
\sum_{x \in X} n + \sum_{i=1}^{n}(i - 1) \ast 3,
\]

where we sum over every \(x \in X\), with \(n\) the domain size of \(x\), i.e., \(|A(x)|\). This increases when atoms \(A(x)\) are not adjacent in the ordering, and is multiplied by the number of distinct subfunctions in OBDD \(\psi\) that depend on \(A(x)\). 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 \(|A(pa(x))|\) to \(|pa(x)|\) for every subfunction, provided that literals \(A(y)\), with \(y \in pa(x)\), for each parent are adjacent in the ordering.