Reducing the Cost of Probabilistic Knowledge Compilation

Giso H. Dal, Steffen Michels and Peter J.F. Lucas
Institute for Computing and Information Sciences
Radboud University
Nijmegen (The Netherlands)
{GDAL,S.MICHELS,PETERL}@CS.RU.NL

Abstract

Bayesian networks (BN) are a popular representation for reasoning under uncertainty. The computational complexity of inference, however, hinders its applicability to many real-world domains that in principle can be modeled by BNs. Inference methods based on *Weighted Model Counting* (WMC) reduce the cost of inference by exploiting patterns exhibited by the probabilities associated with BN nodes. However, these methods require a computationally intensive compilation step in search of these patterns, limiting the number of BNs that are eligible based on their size. In this paper, we aim to extend WMC methods in general by proposing a scalable, compilation framework that is language agnostic, which solves this problem by partitioning BNs and compiling them as a set of smaller sub-problems. This reduces the cost of compilation and allows state-of-the-art innovations in WMC to be applied to a much larger range of Bayesian networks.

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

1. Introduction

The field of probabilistic inference has made considerable progress in the past three decades by the development of novel probabilistic graphical models that support obtaining sparser probability distributions. The use of graphical models has resulted in the ability to solve much bigger probabilistic models than previously was possible. In particular, Bayesian networks (BNs) have become popular graphical models for reasoning under uncertainty. Despite progress in probabilistic inference, the NP hardness of probabilistic inference has remained a stumbling block for using exact inference in real-world domains. Usually, researchers resort to employing approximate probabilistic reasoning under such circumstances, which at least gives them results, but the computational complexity is not any better (Dagum and Luby (1993)).

Exact probabilistic inference has predominantly been performed by (variations of) the *Junction Tree* algorithm (Lauritzen and Spiegelhalter (1988)), which optimizes inference by reusing intermediate computations where possible. Most further improvements in probabilistic inference have been made by exploiting patterns present in probability tables, reducing size and reasoning requirements, e.g., by capturing these in a more concise symbolic representation as offered by *Weighted Model Counting* (WMC) (Bacchus et al. (2003)). However, the computational complexity of the compilation step required to obtain such an optimized symbolic representation also puts limits to the usefulness of these algorithms.

In the present paper we extend inference methods based on WMC by addressing these issues. A scalable framework is proposed that deals with the compilation bottleneck, allowing

state-of-the-art innovations in WMC to be used even in cases when the compilation costs become unacceptable. The solution discussed is to partition probabilistic models. Each partition in turn is compiled into a symbolic representation. We show how to maintain a consist model count with regard to the probability distribution when using this partitioned representation in order to perform inference.

The contributions of this paper are as follows. We introduce a language agnostic framework, facilitating the partitioning and compilation of BNs, and performing probabilistic inference using these partitioned representations. As a result, the cost of compilation is drastically reduced, while at the same time improving the capability to exploit aforementioned patterns. This allows partitions to be ordered independently, rather than maintaining one global ordering, and provides more fine grained control to exploit structure and network topology. Additionally, we propose an upperbound on the size of compiled symbolic representations, and use it in a new algorithm to determine compilation orderings and partitions.

2. Related Work

Advancements in exact probabilistic inference essentially find ways to perform as few operations as possible, either by identifying and eliminating redundant computations or finding more concise factorizations of a problem domain than possible with BNs.

Bayesian networks represent concise factorizations of a probability distribution by using conditional independence assumptions. Further improvements to a factorization have been made by exploiting causal (Heckerman and Breese (1996)), as well as contextual independence (Boutilier et al. (1996)), and determinism (Friedman and Goldszmidt (1998)), i.e., *local structure*.

However, explicit support for local structure is not provided by BNs. Initial attempts to capture local structure include probability trees (Cano et al. (2009)), which are then used in inference algorithms directly.

More recent work relies on the use of graphs, rather than only trees, and reduces inference to *Weighted Model Counting* (WMC). Numerous languages have been used to represent local structure more concisely. Examples include Sentential Decision Diagrams (SDD) (Choi et al. (2013)), Ordered Binary Decision Diagrams (OBDD) (Nielsen et al. (2000)) and Weighted Positive Binary Decision Diagrams (WPBDD) (Dal and Lucas (2017)), among many others. Unfortunately, the cost of compiling a BN into a symbolic representation based on these languages remains a bottleneck. It is exactly this issue that we tackle in this paper.

3. Preliminaries and Background

3.1 Bayesian Networks

A Bayesian network (BN) offers a concise factorization of a probability distribution based on independence assumptions. They model *conditional independence* (CI) as a directed acyclic graph (DAG) by representing independencies among variables $X = \{x^1, \ldots, x^n\}$ as the absence of (directed) edges, and their joint probability distribution as:

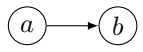
$$P(X) = \prod_{i=1}^{n} P(x^i \mid \textsc{Parents}(x^i)),$$

where $P(x^i \mid \textsc{Parents}(x^i))$ represents the conditional probabilities of variable x^i given its parents $\textsc{Parents}(x^i)$. The degree to which variables are dependent is captured by Conditional Probability Tables (CPTs) associated with every node. Example 1 demonstrates the graphical representation alongside its CPTs and factorization. Note that the example is designed to demonstrate the techniques presented throughout this paper, and should not be considered a typical Bayesian network.

Example 1 *Let a joint probability distribution be defined over variables $X = \{a, b\}$. Figure 1 shows its factorization as a BN together with its corresponding CPTs.*

a	b	$P(X)$
1	1	0.4
1	2	0.4
2	1	0.05
2	2	0.05
3	1	0
3	2	0.1

(a) Joint distribution



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

(b) Bayesian network

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

a	$P(b{=}1\|a)$	$P(b{=}2\|a)$
1	0.5	0.5
2	0.5	0.5
3	0	1

(c) Conditional probability tables

Figure 1 Bayesian network with local structure

3.2 Inference by Weighted Model Counting

The idea behind the WMC approach is to improve the efficiency of exact probabilistic inference, by improving a BN's factorization in the presence of *local structure*: (repetitive) patterns exhibited by conditional probabilities, given a certain context. This structure can be exploited through basic algebraic properties, removing one-valued terms in multiplications and zero-valued terms in additions, distributing exponents over products, or condensing equivalent summands and factors. Simplification relies on the extent to which these patterns are present, and the ability to find and exploit them.

WMC methods most commonly rely on the advances made in the field of Satisfiability solving, and typically consist of the following steps: A BN is encoded as a Boolean formula. This formula is compiled into a chosen logical language, taking into account the tradeoff between its ability to capture local structure vs. compilation time. Compilation can be achieved by using DPLL-style SAT solvers (Davis et al. (1962)), and recording evaluation paths in a manner determined by the chosen language. The arithmetic circuit induced by the obtained symbolic representation is then used to perform inference (Sang et al. (2005)).

Languages proposed in the context of compilation try to find the right tradeoff by imposing various restrictions in favor of tractability. Restrictions that have proven to be useful in *canonical* languages require each subfunction to be deterministic, unique and *read-once*: each variable occurs at most once along every path, typically in the same order. Finding the *optimal factoring* given such a language reduces to finding the optimal variable ordering. Unfortunately, the complexity of compilation given *one* ordering can already be exponential in the worst case (Bollig and Wegener (1996)).

3.3 Partitioning

Symbolic languages used in the field of WMC produce representations that are deterministic, as this property has computational advantages. Allowing non-determinism has shown to produce much more concise representations, but many operations would become intractable. We therefore employ a restricted form of *non-deterministic partitioning* (Bollig (2001)).

We partition the BN by finding a *cut* along the edges that decomposes the graph into two or more components. The *partition set* refers to the nodes in the component, and its *cutset* is comprised of the parents of nodes that have been orphaned by the cut. Multiple partitions will essentially depend on the variable in this cutset, which is the source of non-determinism. We aim to seek a *judicious partitioning* of BNs, i.e., a partitioning where multiple properties must be optimized at the same time: a partitioning that significantly reduces compilation cost, while minimizing the effect on inference cost by keeping cutsets as small as possible.

4. A Scalable Approach to Inference by Weighted Model Counting

The Weighted Model Counting (WMC) approach to inference has proven to be greatly beneficial with regard to a substantial number of real world problems. However, its application has been restricted due to the cost of compilation. We introduce a framework that deals with this problem using partitioning, affecting every stage traditionally present in the WMC approach. The framework consists of 4 distinct phases:

1. *Partition*: Partition a BN into k components.
2. *Compilation*: Encode and compile each component individually.
3. *Assembly*: Connect compiled representations.
4. *Inference*: Perform inference by WMC using assembled representation.

In this section, we provide insight into the advantages of partitioning and how to retain a consistent model count with regard to the probability distribution. We dedicated Section 5 to the *partition* phase where we elaborate on the mechanics involved in finding a good partitioning in the context of BNs.

4.1 Compiling Partitioned Bayesian Networks

Our framework reduces compilation cost, which essentially allows us, given an appropriate partitioning, to employ WMC methods to any BN regardless of compilation language and ordering. Without loss of generality we will use a particular combination of encoding and language for demonstration purposes, producing one variety of decision diagram, called *Weighted Positive Binary Decision Diagram* (WPBDD) (Dal and Lucas (2017)).

Let a Bayesian network be defined over variables X. It is encoded into a Boolean formula f, and we explore valuations of this formula through assignment of truth values by *conditioning* f on instantiated variables, defined as the projection:

$$f_{|x^i \leftarrow b}(x^1, \ldots, x^n) = f(x^1, \ldots, x^{i-1}, b, x^{i+1}, \ldots, x^n), \tag{1}$$

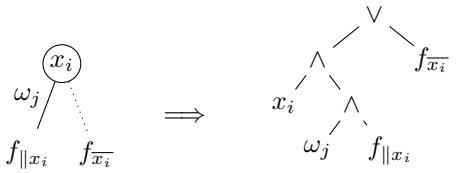
with $b \in \{0, 1\}$. We use shorthand notations f_x and $f_{\overline{x}}$ to denote $f_{|x \leftarrow 1}$ and $f_{|x \leftarrow 0}$, respectively. Evaluation paths are recorded as a decision diagram where each node denotes a portion of the logical circuit it induces. Formula f essentially depends on atoms $\mathcal{A}(X)$,

by introducing atom x_i for each unique variable-value pair: $\mathcal{A}(x) = \{x_1, \ldots, x_n\}$ for each $x \in X$, where n is determined by $\mathcal{D}(\{x\})$ using domain function $\mathcal{D}$:

$$\mathcal{D}(Y) = \prod_{x \in Y} |x|, \tag{2}$$

where $|x|$ denotes the dimension of $x \in X$. Finally, unique symbolic weights ω_j identify distinct probabilities local to x's CPT. We thus introduce into f a clause for each weight ω_j containing the variables on which the weight depends, and clauses to represent constraints among variables (Dal and Lucas (2017)).

Figure 2 show a decision node, where the solid and dotted edges indicate the assignment of true and false to x_i, respectively, where $f_{\|x_i}$ is called the positive cofactor and is obtained by conditioning f on $x_i \in \mathcal{A}(x)$ and $\overline{x_j}$ for all $x_j \in \mathcal{A}(x) \backslash x_i$. $f_{\overline{x_i}}$ is the negative cofactor.



(a) WPBDD node **(b)** Logical circuit

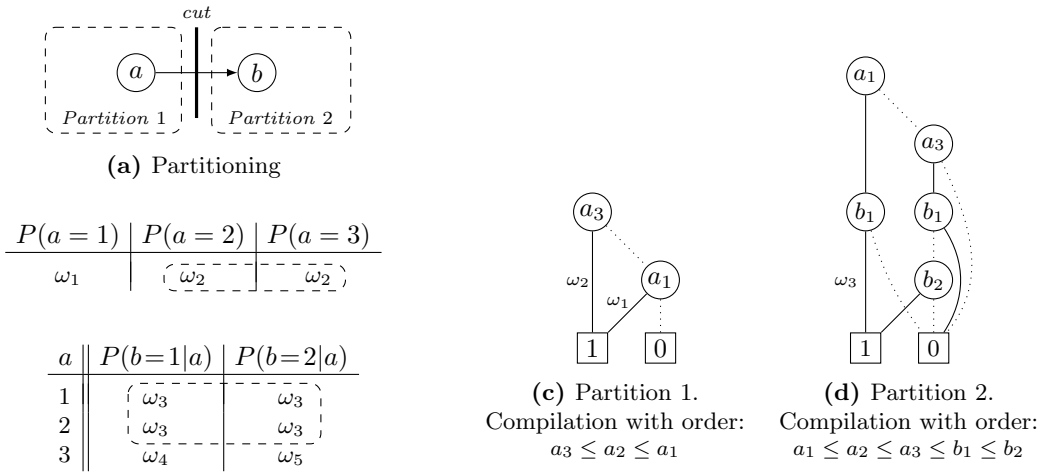
Figure 2 From node to logical circuit

When partitioning BNs we must consider their semantics: it's a factorization, where conditional probabilities are expressed in the underlying CPTs. Based on the definitions provided in Section 3.3, the cutset is essential to a partition, because the conditional probabilities of node x depend on $\textsc{Parents}(x)$ in addition to x itself. This means that the Boolean function representing a particular partition containing variable x, essentially depends on variables $\mathcal{A}(x) \cup \mathcal{A}(\textsc{Parents}(x))$.

Example 2 *Consider the BN of Example 1. Figure 3 shows a cut that decomposes it into two components. The partition- and cutset of partition* 1 *are* $\{a\}$ *and* $\{\}$*, and those of partition* 2 *are* $\{b\}$ *and* $\{a\}$*, respectively. We have identified equal probabilities per CPT, and have encircled the way structure is captured given the orderings used during compilation. Note that deterministic probabilities (*0 *and* 1*, represented as* ω_4 *and* ω_5*) are captured through appropriate edge connections and are therefore not explicitly present as edge weights. Particular to WPBDDs, any literal indicated in the ordering that is not present as the negative cofactor of its parent, infers that they have equivalent positive cofactors and that morgan's law has been applied.*

Several advantages to partitioning become evident from Example 2: (1) reduced compilation cost, and (2) improved capability of exploiting local structure through independent partition orderings. Let's elaborate on that.

The computational complexity of compilation can quickly become discouraging. Consider individual probabilities as distinct constraints that collectively represent a BN. Each probability ω_j local to x's CPT essentially depends on variables $\mathcal{A}(x) \cup \mathcal{A}(\textsc{Parents}(x))$. This means that there are as many constraints as there are entries in all CPTs. In Example 2, instead of compiling a monolithic BN consisting of 9 constraints, we compile two BNs



(a) Partitioning

(b) Capture symbolic structure in CPTs

(c) Partition 1. Compilation with order: $a_3 \leq a_2 \leq a_1$

(d) Partition 2. Compilation with order: $a_1 \leq a_2 \leq a_3 \leq b_1 \leq b_2$

Figure 3 Compilation of a partitioned BN, where x_i signifies x being equal to its i^{th} value.

consisting of 3 and 6 constraints. As compilation can be of exponential complexity (Bollig and Wegener (1996)) and is dominated by these constraints, we have found a scalable approach to compilation for the WMC approach.

The degree to which we can exploit problem structure is determined by the ordering used during compilation. An ordering that is good for one part of the network, might not be well suited for another. We improve upon this by allowing partitions to be ordered independently, rather than having one global ordering, thus allowing more fine grained control to exploit structure and capturing network topology.

4.2 Inference using Compiled Partitions

We need to take extra measures in order to maintain a consistent model count with regard to the probability distribution. In order to perform inference we connect compiled partitions as a *tiered architecture*, where edges to the `true` terminal in one tier are extended to the root node of the compiled partition in the next tier. By doing this we essentially lose the read-once property due to non-determinism: cutset variables are shared among multiple partitions and can therefore occur multiple times along each path. We can maintain consistency through *dynamic conditioning*: as we traverse depth-first through the connected representation, we make decisions *persistent*, i.e., we make the same decision we had made previously. This can also be achieved through *static conditioning*: we build a monolithic representation, not by conjoining compiled partitions, but by appropriately conditioning and connecting them.

The variables of interest with regard to maintaining consistency are precisely those upon which multiple partitions essentially depend. These *persistence variables* must retain their value in all subsequent tiers once a decision had been made. This creates a dependency *spanned* from the first to last tier where a particular persistence variable occurs in. Each

Algorithm 1 Dynamic conditioning

```
IsConditioned(node, E, tier)
1  v = Variable(node)
2  return E_v < tier

GetCacheId(S, E, tier)
1  id = 0, exp = 1
2  for variable in S_tier
3       id += GetValueIndex(variable) * |variable|^exp
4       exp = exp + 1
5  return id

TraverseTier(P, S, E, tier)
 1  if tier == |P|
 2       return 1
 3  else
 4       cacheid = GetCacheId(S, E, tier)
 5       if IsCached(cacheid, tier)
 6            return GetCachedValue(cacheid, tier)
 7       else
 8            Initialize(P_tier)
 9            p = Traverse(P_tier, P, S, E, tier)
10            CacheValue(cachid, tier, p)
11            return p

Traverse(node, P, S, E, tier)
 1  if IsTrueTerminal(node)
 2       return TraverseTier(P, S, E, tier+1)
 3  elseif IsFalseTerminal(node)
 4       return 0
 5  elseif IsTraversed(node)
 6       return GetProbability(node)
 7  else
 8       pt = 0, pe = 0
 9       if not IsConditioned(node, E, tier)
                 or IsTrue(node)
10            v = Variable(node)
11            E_v = tier
12            pt = Traverse(node.t, P, S, E, tier)
13       if not IsConditioned(node, E, tier)
                 or IsFalse(node)
14            pe = Traverse(node.e, P, S, E, tier)
15       return Probability(node, pt, pe)

ComputeProbability(P, S, E)
   input: Partition roots P, spanning sets S
        and evidence E
   output: probability given evidence E
1  return TraverseTier(P, S, E, 1)
```

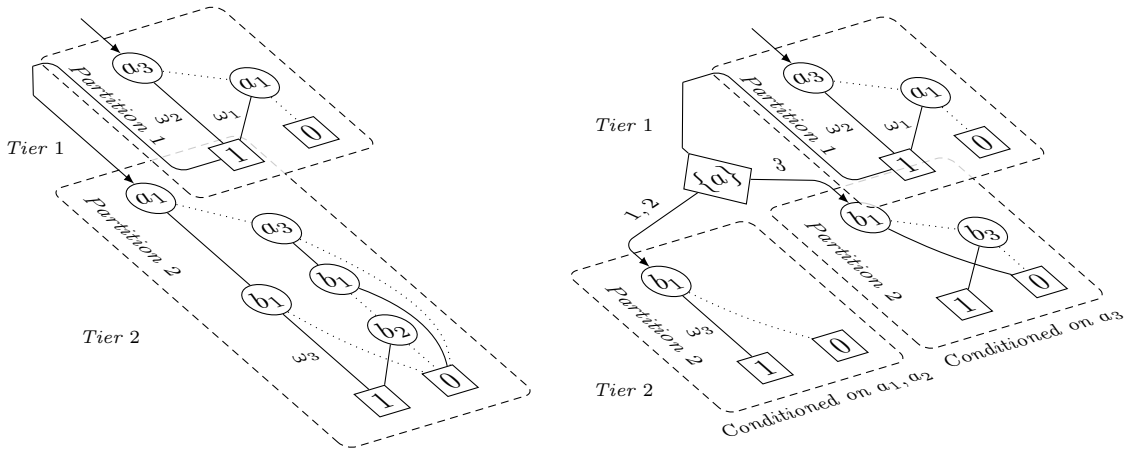
tier is associated with *spanning set* containing the persistence variables that take part in the aforementioned dependency. Together, these sets are used for caching purposes.

Algorithm 1 shows how to perform inference using dynamic conditioning. Partitions are traversed in the order imposed on P, starting at their root P_i. The values of evidence array E are initialized to ∞, and for each observed BN variable x to 0. Function IsConditioned thus determines if a variable has been assigned to in a preceding tier during the traversal, or was part of the evidence from the start. GetCacheId returns a unique key determined by a particular assignment of truth values to persistence variables present in the spanning set, corresponding to the given tier. Using this prevents unnecessary traversal of subsequent tiers. The algorithm can be extended for static conditioning, by using the same principles used in GetCacheId to choose a correct conditioned partition in subsequent tiers.

Example 3 *Consider the compiled partitions of Example 2. Figure 4 shows how we connect them. Both partitions represent functions that depend on* $\mathcal{A}(a) = \{a_1, a_2, a_3\}$*, i.e., the persistence variables. The spanning sets therefore are* $S_0 = \{\}$ *and* $S_1 = \{a\}$*.*

Consider Figure 4a. Assume that we have traversed partition 1 *depth-first and have encountered the* `true` *terminal based on decision* $a_3 =$ `true`*. When continuing to partition* 2 *we employ dynamic conditioning by restricting its traversal. This is done by not considering the positive cofactors of nodes that represent atoms* a_1 *and* a_2*, nor the negative cofactor of* a_3*.*

Figure 4b shows that the compiled representation of partition 2 *is conditioned on its persistence variables that have occurred in a preceding tier, i.e.,* $\mathcal{D}(S_1) = 3$ *times. Observe that this is an upperbound in the number of distinct conditioned partitions, and can be reduced when equivalent representations are revealed. This occurs as the result of conditioning in*



(a) Dynamic conditioning **(b)** Static conditioning

Figure 4 Tiered architecture

Tier 2. *The traversal is similar to the dynamic approach, except here we have added explicit logic between partitions responsible for choosing the correct conditioned partition.*

Both approaches have their own strengths. By using the dynamic conditioning approach we move some of the workload away from compilation, toward the inference phase. We are ready to perform inference immediately after compilation and are able to easily change the order in which partitions are connected, to better accommodate inference query optimizations. Building a monolithic representation through the static approach requires a fixed connection ordering, upon which it bases its conditioning and connection strategy. Advantageous to this approach is that it preserves the computational complexity of inference as linear in the size of the representation, however, at the cost of space.

5. Finding a Good Partitioning

The bottleneck of the WMC approach to probabilistic inference is compilation. Partitioning influences both compilation and inference cost, and obtaining one of high quality is therefore crucial. We search for a good partitioning by providing implications to its quality through the use of an upperbound on the size of the resulting symbolic representation.

5.1 Finding Partitionings and Variable Orders

The size of symbolic representations depends on the ordering that is imposed on the variables during compilation. In Section 5.2 we introduce an upperbound on the size of these representations. We search for good orderings by minimizing this upperbound. We have used *simulated annealing* to perform this task, as it seemed best suited.

The size implications provided by the upperbound are also used in the search for partitionings. However, the search space is too large for straightforward traversal. We therefore recursively split the partition with the largest bound, until a desirable overall bound is reached. We additionally reduce the state space by making sure that partitions represent

Algorithm 2 Computing upperbound

```
SpanningVariables(C, O)
    input: Constraints C and ordering O
    output: Spanning variables S
1   S = ∅
2   for l = 1 to |O| + 1
3       S_l = {}
4       for i = 1 to |C| + 1
5           if C_i ∩ {o_1, ..., o_{l-1}} ≠ C_i
6               S_l = S_l ∪ (C_i ∩ {o_1, ..., o_{l-1}})
7   return S

Opn(C, O, l)
    input: Constraints C, ordering O, level l
    output: Logical operators per node
1   weights = 0
2   for i = 1 to |C| + 1
3       if C_i \ {o_1, ..., o_{l-1}} = o_l
4           weights += 1
5   return 2 + weights

Upperbound(C, O)
    input: Constraints C and ordering O
    output: Number of logical operators
1   operators = 0
2   S = SpanningVariables(C, O)
3   for l = 1 to |S| + 1
4       nodes = D(S_l ∪ {o_l})
5       operators += nodes * Opn(C, O, l)
6   return operators
```

connected components in the moralized graph of the BN. Connected partitions generally produce smaller representations, while minimizing the number of variables in their cutset.

5.2 An Upperbound

Let a Bayesian network be defined over n variables X. We formulate it as a set of *constraints* $C_x \in C$, where C_x represents the dependencies of variable $x \in X$:

$$C_x = \{x\} \cup \textsc{Parents}(x). \tag{3}$$

During compilation, variables are evaluated one by one using a particular ordering $O = \{o_1, \ldots, o_n\}$ imposed on X by π, where $o_i = \pi(x)$, $o_j \leq o_k$ for $j < k$. A constraint set C_x is *satisfied* at a particular evaluation depth, or *level*, when all its variables are evaluated on that level, i.e., if $C_x \subseteq \{o_1, \ldots, o_l\}$ on level l. A constraint C_x is *spanned* over multiple levels from the moment it is partially- until fully satisfied.

The upperbound on the size of the compiled representation is a cumulative measure determined by the domain size of *spanning variables* $S_l \in S$ associated with each level l. Spanning variables S_l are precisely those variables that partially satisfy constraints C_x on level $l-1$:

$$S_l = \{o_1, \ldots, o_{l-1}\} \cap \bigcup_{\{C_x \in C \,:\, C_x \subseteq \{o_1, \ldots, o_{l-1}\}\}} C_x \tag{4}$$

A generalized upperbound depending solely on the structure of a BN is computed by:

$$\sum_{l=1}^{n} \mathcal{D}(S_l), \tag{5}$$

where the domain size $\mathcal{D}(S_l)$ of spanning variables S_l on level l represents the *maximum number of (multi-valued) decision nodes* required to represent the BN on that level. This occurs in the absence of local structure. The bound can be determined specifically for WPBDDs using Algorithm 2, which is based on Definition 1 and demonstrated in Example 4.

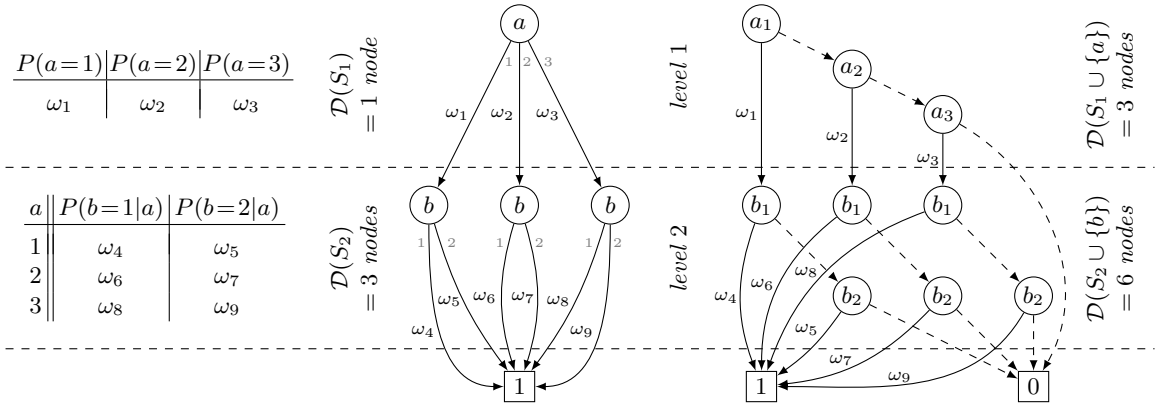


Figure 5 From generalized to tailored upperbound,
where x_i signifies x being equal to its i^{th} value.

Definition 1 *Let a BN be defined over variables X, with its dependency relations represented as constraints $C_x \in C$ for each $x \in X$ (Equation 3). Given ordering $O = \{o_1, \ldots, o_n\}$, an upperbound on the size of the corresponding WPBDD is determined by:*

$$\sum_{l=1}^{n} \mathcal{D}(S_l \cup \{o_l\}) * \textsc{Opn}(C, O, l),$$

where the size on level l is determined by $o_l \in O$, $S_l \in S$ (Equation 4), and Opn *(Algorithm 2) returns the number of logical operators per node.*

Example 4 *Consider the BN of Example 1. Its dependence relations are formulated as constraints $C = \{\{a\}, \{a, b\}\}$ (Equation 3). Assuming no local structure, Figure 5 shows multi-valued (left) and binary decision diagram (right) representations of the BN, given variable ordering O, $a \leq b$, and a corresponding literal ordering $a_1 \leq a_2 \leq a_3 \leq b_1 \leq b_2$, respectively. It shows how the general upperbound relates to the tailored one. Given constraints C and variable ordering O, the spanning variable sets S are $S_1 = \{\}$ and $S_2 = \{a\}$. The upperbound of the WPBDD is thus computed by:*

$$\mathcal{D}(S_1 \cup \{a\}) * \textsc{Opn}(C, O, 1) + \mathcal{D}(S_2 \cup \{b\}) * \textsc{Opn}(C, O, 2) = 3 * 3 + 6 * 3 = 27.$$

6. Empirical Results

We have used several publicly available Bayesian networks to empirically demonstrate the effects of our framework on compilation and inference cost. Table 1 shows compilation size and time for multiple representations given the same ordering. For SDDs this ordering was used to induce a balanced vtree, which was then used as a starting point for compilation. More important than whether or not a particular ordering is best suited for certain representations, is the fact that BNs can be compiled that previously could not even be compiled given this ordering by using partitioning. Additionally, compilation size and time have both significantly reduced, sometimes by many orders of magnitude, while using a very limited

Bayesian Network	X	$\mathcal{A}(X)$	P	WPBDD S	WPBDD T	WPBDD S	WPBDD T	SDD S	SDD T	OBDD S	OBDD T
insurance	27	89	2	33183	0.014	348956	0.077	1731415	2.046	1263540	0.289
weeduk	15	90	2	30735	0.406	30733	0.418	-	-	109734	0.176
alarm	37	105	2	2730	0.003	3788	0.003	35004	0.054	10008	0.003
water	32	116	2	49212	0.095	219797	0.506	-	-	-	-
powerplant	40	120	2	2451	0.002	4158	0.003	26662	0.038	11043	0.002
carpo	54	122	2	1937	0.003	2377	0.003	13405	0.028	7179	0.003
win95pts	76	152	2	49405	0.018	810957	0.361	1109210	2.057	4876152	4.997
hepar2	70	162	2	33234	0.025	56574	0.033	188453	2.423	142806	0.161
fungiuk	15	165	2	79682	1.515	234322	7.763	-	-	733551	0.812
hailfinder	56	223	2	225325	0.052	4025502	1.395	10508499	7.51	31493220	12.435
3nt	58	228	2	9844	0.015	858645	0.578	42774722	58.905	15592962	19.493
4sp	58	246	2	83156	0.035	918353	0.352	-	-	20558352	34.598
barley	48	421	2	13721258	11.197	-	-	-	-	-	-
mainuk	48	421	2	9045244	9.002	-	-	-	-	-	-
andes	220	440	2	426513	0.117	-	-	-	-	-	-
pathfinder	135	520	2	143032	0.493	577163	0.717	2287777	23.337	5732988	18.656
mildew	35	616	2	1634250	111.434	5666709	113.264	-	-	-	-
munin1	186	992	4	13196919	6.693	-	-	-	-	-	-
pigs	441	1323	9	3292450	1.534	-	-	-	-	-	-

Table 1 Compilation cost,
where X and $\mathcal{A}(X)$ are the number of variables in the BN and encoding, S is the number of logical operators that the symbolic representation induces, time T is in seconds, P is number of partitions, and - implies compilation failure due to memory requirements.

number of partitions. The results indicate that multiple representations can benefit from the techniques presented in this paper.

Figure 6 shows inference time, compilation time and representation size in relation to partitioning, to better understanding the effects of our framework. Inference becomes more difficult as we increase the number of partitions, but the point to focus on is that compilation time and representation size have decreased significantly before inference time becomes an issue. This is also confirmed by the tested networks in Table 1, where inference cost has increased by only 46% on average, while the reduction in compilation cost is many times greater. Since these results rely on the quality of the partitioning, its further improvement will have significant impact.

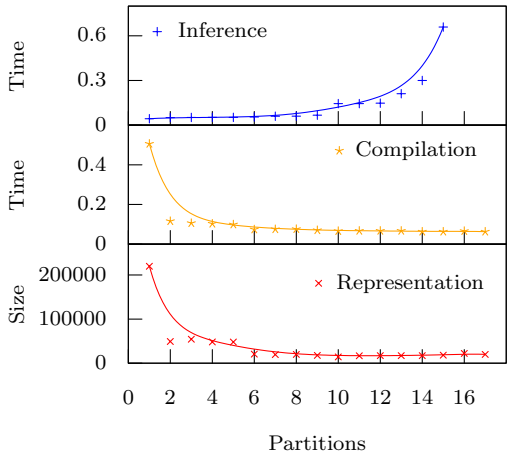


Figure 6 Effects of partitioning on the *water* network.

7. Conclusion

Weighted Model Counting has been proposed as a technique for exact probabilistic inference that supports exploiting local structure (Dal and Lucas (2017); Nielsen et al. (2000); Choi et al. (2013); Sang et al. (2005)). However, inference is performed on a representation that is obtained through compilation. It is exactly where the limitation of WMC is revealed: compilation can be very costly. In this paper we extend state-of-the-art compilers and model

counters, such as SDD, CUDD and WPBDD, and the ACE and CACHET model counters to tackle computation costs.

As the experimental evaluation demonstrates, our framework reveals several benefits: (1) the compilation cost can be drastically reduced while using only a limited number of partitions, (2) the representations obtained are much smaller, thus reducing resource requirements, and (3) the time saved in this way can be invested in searching for even more concise representations. As a consequence, with similar computation costs much bigger Bayesian networks can be handled. The need for algorithms that can deal with very big Bayesian networks is clearly visible in areas such as big-data analysis where lots of variables need to be handled.

References

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *Proceedings of the 44th Symposium on Foundations of Computer Science*, pages 340–351, 2003.

Beate Bollig. Restricted nondeterministic read-once branching programs and an exponential lower bound for integer multiplication. *Theoretical Informatics and Applications*, 35:149–162, 2001.

Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *Transactions on Computers*, 45:993–1002, 1996.

Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth international conference on Uncertainty in artificial intelligence*, pages 115–123, 1996.

Andrés Cano, Manuel Gómez-Olmedo, Serafín Moral, and Cora B Pérez-Ariza. Recursive probability trees for Bayesian networks. In *Conference of the Spanish Association for Artificial Intelligence*, pages 242–251, 2009.

Arthur Choi, Doga Kisa, and Adnan Darwiche. Compiling probabilistic graphical models using Sentential Decision Diagrams. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 121–132, 2013.

Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial intelligence*, 60:141–153, 1993.

Giso H. Dal and Peter J. F. Lucas. Weighted Positive Binary Decision Diagrams for exact probabilistic inference. *International Journal of Approximate Reasoning*, 2017.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

Nir Friedman and Moises Goldszmidt. Learning Bayesian networks with local structure. In *Learning in graphical models*, pages 421–459. 1998.

David Heckerman and John S Breese. Causal independence for probability assessment and inference using Bayesian networks. *Transactions on Systems, Man and Cybernetics*, 26:826–831, 1996.

Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, pages 157–224, 1988.

Thomas D Nielsen, Pierre-Henri Wuillemin, Finn V Jensen, and Uffe Kjærulff. Using ROBDDs for inference in Bayesian networks with troubleshooting as an example. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 426–435, 2000.

Tian Sang, Paul Beame, and Henry A Kautz. Performing Bayesian inference by weighted model counting. In *AAAI*, volume 5, pages 475–481, 2005.