Proceedings of Machine Learning Research vol 72, 97-108, 2018 PGM 2018

Parallel Probabilistic Inference by Weighted Model Counting

Giso H. Dal GDAL@CS.RU.NL
Institute for Computing and Information Sciences, Radboud University, The Netherlands

Alfons W. Laarman A.W.LAARMAN@LIACS.LEIDENUNIV.NL
Institute of Advanced Computer Science, Leiden University, The Netherlands

Peter J.F. Lucas PETERL@CS.RU.NL
Institute for Computing and Information Sciences, Radboud University, The Netherlands

Abstract

Knowledge compilation as part of the Weighted Model Counting approach has proven to be an efficient tool for exact inference in probabilistic graphical models, by exploiting structures that more traditional methods can not. The availability of affordable high performance commodity hardware has been an inspiration for other inference approaches to exploit parallelism, to great success. In this paper, we explore the possibilities for Weighted Model Counting. We have empirically confirmed that exploited parallelism yields substantial speedups using a set of real-world Bayesian networks.

1. Introduction

Probabilistic inference is of central importance to Artificial Intelligence (AI) for reasoning under uncertainty. It has applications in areas such as medical diagnosis, speech recognition, weather forecasting, machine learning, and so on Hommersom et al. (2013); Russell and Norvig (1995). Unfortunately, probabilistic inference is NP-hard Cooper (1990). As a consequence, for more than three decades researchers have investigated ways to exploit independence and structure in probability distributions in attempts to make probabilistic methods tractable and practically more useful. Weighted Model Counting (WMC) is considered a state-of-the-art approach to inference and exploits independence and structure in an excellent way. It typically entails two phases: (1) The search for, and construction of, a concise representation of a probability distribution, referred to as *knowledge compilation*, and (2), the subsequent evaluation on that representation to perform inference Chavira and Darwiche (2008). In this article, we explore the possibilities to parallelize both phases. At first sight, in particular the parallel nature of the logic behind model counting creates the idea that parallelization may yield significant performance gains.

Bayesian networks (BN) provide intuitive graph representations of problems dealing with joint probability distributions. Shachter and D'Ambrosio introduced a symbolic-algebraic approach to represent BNs in order to perform exact probabilistic inference more efficiently Shachter et al.. Darwiche identified key properties of these symbolic representations that are essential to tractable inference Darwiche (2002), inspiring influential representations such as *Deterministic Decomposable Negation Normal Form* (d-DNNF) and *Sentential Decision Diagrams* (SDD) Choi et al. (2013). These advancements confirmed the viability of the field that is now better known as inference by Weighted Model Counting Chavira and Darwiche (2008).

The compilation step in WMC is computationally intensive, and thus, that is where one finds most tractability issues. Inference is linear in the size of the compiled representation Darwiche (2002). In practice this implies that if compilation succeeds given time and memory limitations, inference is not only guaranteed to succeed, it is much faster than compilation. The exponential trend of growing pro-

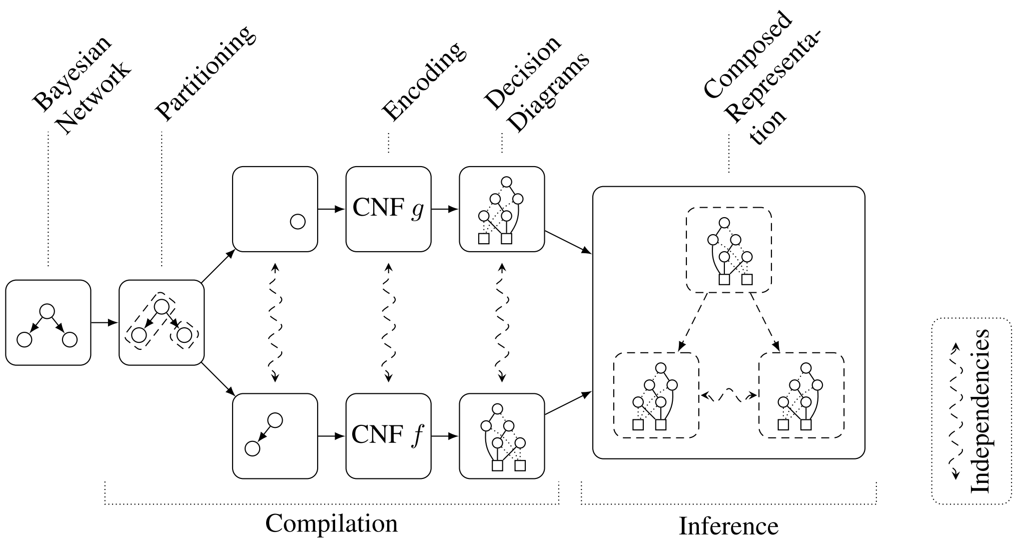


Figure 1 From partitioned knowledge compilation to inference.

cessor speeds, known as Moore's Law, recently shifted towards exponential growth in parallelism. So, to benefit from the next generation of processors, the WMC approach should exploit parallel hardware. The challenge here is that this approach is applied to problems with an abundance of dependencies, e.g. BNs, limiting the amount of independence that can be exploited through parallelism. In order to increase the amount of independence, we also apply a partitioning technique as depicted in Figure 1 (see Section 4 for a detailed explanation). Our contributions are the following:

- a novel parallel approach to compiling BNs to different representations (Section 4.1);
- a novel parallel inference algorithm (Section 4.2);
- the implementation of compilation and inference algorithms is discussed and it is shown that 15-fold speedups are attainable (Section 4 and 5).

2. Related Work

Now a decade ago, inference by WMC was discovered as a new state-of-the-art method to perform probabilistic inference Chavira and Darwiche (2008). At the core of WMC is a model-theoretic analysis, *model counting*, of a representation that symbolically describes the probability distribution of a BN. In the first step of this approach, the BN is *compiled* into a symbolic representation based on a logical formalism. In the second step, inference is performed through model counting.

While the memory required to store all models increases exponentially Darwiche (2002), compiled representations typically behaves much better. Examples of used representations in this context are (binary) decision diagrams (BDD) Bryant (1986), d-DNNF, SDDs, OBDDs, and many more Choi et al. (2013); Dal and Lucas (2017); Nielsen et al. (2000). In essence, inference by WMC employs techniques from reachability analysis that are used to find more concise representations of probability distributions in order to perform inference more efficiently.

Parallelism has be used since the 1990s to improve inference algorithms, by Kozlov and D'Ambrosio for instance D'Ambrosio et al. (1992); Kozlov and Singh (1994). Several parallel implementations have been presented since then. Some, specialized to particular models or settings like the works of Funiak, Newman and Paskin et. al. Funiak et al. (2007); Newman et al. (2008). Parallelism in approximate inference was addressed by Gonzalez and Mendiburu Gonzalez et al. (2009); Mendiburu et al. (2007), and exact inference by Prasanna et. al. Namasivayam and Prasanna (2006). Parallel #SAT solving has been attempted before, but not for the weighted case, nor compiled representations. Maximum a posteriori (MAP) queries already are trivial to parallelize, due to the independence between configurations one has to go through to answer such probabilistic queries. Further related work has been done in the field of reachability analysis, where they focused on performing multiple independent BDD operations in parallel, as well as parallelizing the BDD operations themselves Dijk et al. (2013). Performance gains where also achieved by using GPU to perform graph traversals, applicable during the counting of models Dal et al. (2014).

Attempts to improve performance by exploiting parallelism have been limited to either particular models or settings, or popular inference methods such as variational inference, belief propagation and the junction tree algorithm, while inference by WMC has not been investigated.

3. Preliminaries

3.1 Bayesian Networks

Definition 1 *A Bayesian network $B = (G, P)$ is a Directed Acyclic Graph (DAG) $G = (V, E)$, with nodes V and (directed) edges $E \subseteq V \times V$, that models a factorization of joint probability distribution $P(X)$ defined over variables X as:*

$$P(X) = \prod_{x \in X} P(x \mid \mathrm{pa}_G(x)), \tag{1}$$

such that node $v \in V$ is associated with a distinct $x \in X$, and corresponds to probabilities $P(x|\mathrm{pa}_G(x))$ depending on parents $\mathrm{pa}_G(x) \triangleq \{z \mid (z, x) \in E\}$, which make up node v's Conditional Probability Table (CPT). Furthermore, $\mathrm{pa}_G(U) \triangleq \bigcup_{u \in U} \mathrm{pa}_G(u) \setminus U$, with $U \subseteq X$.

Definition 2 *Let X be a set of variables. We denote the dimension of $x \in X$ with $\mathcal{D}_X(x)$. Furthermore, $\mathcal{D}_X$ determines the domain size of $U \subseteq X$:*

$$\mathcal{D}_X(U) \triangleq \prod_{x \in U} \mathcal{D}_X(x). \tag{2}$$

3.2 Inference by Weighted Model Counting

A BN models a concise factorization of discrete probability distribution $P(X)$, defined over X. The WMC approach attempts to improve on the computational advantages of the factorization by employing additional algebraic properties. This process is typically done in a framework where a BN $B = (G, P)$ is encoded as a Boolean formula Chavira and Darwiche (2008). This formula is then represented as, or *compiled* to, a symbolic representation that respects the factorization and allows for more efficient inference. The computational complexity of inference is linear in the size of this compiled symbolic representation if it adheres to a set of key properties identified by Darwiche Darwiche (2002). We now introduce the preliminaries required to employ this method and its 3 main steps.

3.2.1 REPRESENTATION: BOOLEAN LOGIC AND SETS

A *literal* l is a Boolean variable x, or its negation $\bar{x}$. A *propositional formula* (proposition for short) ψ is a literal, Boolean constant, or composite proposition with connectives such as negation $\overline{\varphi}$, conjunction $(\varphi \wedge \varphi')$, disjunction $(\varphi \vee \varphi')$, and implication $(\varphi \implies \varphi')$, where φ and φ' are propositions, with precedence of the connectives in that (descending) order. We use $\top$ to denote the always true and $\perp$ to denote the always false proposition. A proposition can be represented as a Boolean function $f : \mathbb{B}^m \to \mathbb{B}$ defined over m Boolean variables. The *conditioning* of f on x_i is defined as the projection $f_{|x_i \leftarrow b}(x_1, \ldots, x_m) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_m)$, with $b \in \{\top, \bot\}$. We use f_{x_i} and $f_{\overline{x_i}}$ to denote $f_{|x_i \leftarrow 1}$ and $f_{|x_i \leftarrow 0}$, respectively. A Boolean function f *depends* on x_i if $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_m) \neq f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_m)$. A proposition φ is in *conjunctive normal form* (CNF) if it is a conjunction of clauses, where a clause is a disjunction of literals $(l_1 \vee \cdots \vee l_n)$. In the following we will often not make a distinction between a proposition φ and its associated Boolean function f.

A *set* is an unordered collection of unique objects, denoted using curly brackets, e.g., $\{1, 2, 3\}$. Symbols used to operate on sets include $\cap$ (intersection), $\cup$ (union), $\setminus$ (set difference) and $\times$ (cartesian product). The symbol $\emptyset$ denotes the empty set.

3.2.2 STEP 1: BAYESIAN NETWORK ENCODING

BNs are defined over multi-valued domains. An encoding is required to transition to the Boolean domain. Multiple encodings have been proposed Chavira and Darwiche (2008). We employ the so-called direct encoding that introduces a Boolean variable x_i for each unique variable-value pair Dal and Lucas (2017), adding to x_i the semantic that if $x_i = \top$ then x *is equal to its* i^{th} *value.*

Definition 3 *Let BN $B = (G, P)$ be defined over variables X. Variable $x \in X$ is encoded as Boolean atoms $\mathcal{A}_X(x) \triangleq \{x_1, \ldots, x_n\}$ with $n = \mathcal{D}_X(x)$. Variables $U \subseteq X$ are encoded as Boolean atoms $\mathcal{A}_X(U)$ such that $\mathcal{A}_X(y) \cap \mathcal{A}_X(z) = \emptyset$ for all $y, z \in X$ with $y \neq z$:*

$$\mathcal{A}_X(U) \triangleq \bigcup_{x \in U} \mathcal{A}_X(x), \tag{3}$$

Definition 4 *A <u>propositional probabilistic knowledge base</u> (PPKB) is a set of weighted formulas $\{\langle \varphi_1, \omega_1 \rangle, \ldots, \langle \varphi_n, \omega_n \rangle\}$, where each propositional formula φ_i is associated with <u>weight</u> ω_i, symbolically representing probability* $\textsc{Prob}(\omega_i) \in [0, 1]$. *A PPKB is written as proposition by conjoining each pair $\langle \varphi_i, \omega_i \rangle$ that is syntactic sugar for $(\varphi_i \rightarrow \omega_i)$, or $(\overline{\varphi_i} \vee \omega_i)$ in CNF.*

Example 1 *Let BN $B = (G, P)$ be defined over variables $X = \{a, b, c\}$ as shown in Figure 2a, with $\mathcal{A}_X(a) = \{a_1, a_2, a_3\}$ and* $\mathrm{pa}_G(a) = \emptyset$. *We construct a PPKB KB by adding for every $x \in X$ mutual exclusion constraints for the values of x based on $\mathcal{A}_X(x)$, and add a weighted formula for every probability in x's CPT Dal and Lucas (2017): $\{\langle \overline{a_1} \wedge \overline{a_2} \wedge \overline{a_3}, \omega_1 \rangle, \langle a_1 \wedge a_2, \omega_2 \rangle, \langle a_1 \wedge a_3, \omega_3 \rangle, \langle a_1, \omega_4 \rangle, \langle a_2, \omega_5 \rangle, \langle a_3, \omega_6 \rangle, \ldots\}$, where* $\textsc{Prob}(\omega_i) \in [0, 1]$. *Its CNF is $(a_1 \vee a_2 \vee a_3 \vee \omega_1) \wedge (\overline{a_1} \vee \overline{a_2} \vee \omega_2) \wedge (\overline{a_1} \vee \overline{a_3} \vee \omega_3) \wedge (\overline{a_1} \vee \omega_4) \wedge (\overline{a_2} \vee \omega_5) \wedge (\overline{a_3} \vee \omega_6) \wedge \ldots$. The clause $(a_1 \vee a_2 \vee a_3 \vee \omega_1)$ acts as one of the constraints to ensure that at least one of the values a_1, a_2, and a_3 is true and this is ensured by taking $\omega_1 \equiv \bot$. Note that $(a_1 \vee a_2 \vee a_3 \vee \bot) \equiv (a_1 \vee a_2 \vee a_3)$, so we can also leave out $\bot$ here. Mutual exclusiveness is expressed by clauses such as $(\overline{a_1} \vee \overline{a_2} \vee \omega_2)$. For similar reasons we have that $\omega_2 \equiv \omega_3 \equiv \bot$*

3.2.3 STEP 2: COMPILING TO SYMBOLIC REPRESENTATIONS

To refactor joint probability distribution $P(X)$ we use a representation that symbolically represents it, based on the encoding E (Definition 3 and 4). Using representations such as Binary Decision Diagrams (BDD) to represent the PPKB provides access to a rich set of tools specifically developed for refactoring. A BDD is a rooted DAG, with two leaves labeled 1 ($\top$) and 0 ($\bot$). Each node v is labeled with a Boolean variable $\textsc{Var}(v)$ and has two outgoing edges to children $\textsc{High}(v)$ and $\textsc{Low}(v)$, denoting respectively the *positive cofactor* $f_{\textsc{Var}(v)}$, and the *negative cofactor* $f_{\overline{\textsc{Var}(v)}}$. Here, edges to the HIGH and LOW children of each node are depicted with solid and dashed lines, respectively. Node v represents (a portion of) encoding E as $\text{REP}(v) = (\text{VAR}(v) \wedge f_{\text{VAR}(v)}) \vee (\overline{\text{VAR}(v)} \wedge f_{\overline{\text{VAR}(v)}})$, where $\text{REP}(r) \equiv E$ at root node r. A BDD is *ordered* if every variable is encountered at most once and in the same order along each distinct path from the root to a leaf. A *total ordering* is imposed on variables X by permutation σ, such that $x_{\sigma(i)}$ precedes $x_{\sigma(j)}$, denoted by $x_{\sigma(i)} \prec x_{\sigma(j)}$, iff $i < j$ with $x_{\sigma(i)}, x_{\sigma(j)} \in X$.

A *Weighted Positive Binary Decision Diagram* (WPBDD) Dal and Lucas (2017) is an ordered BDD, with a set of weight variables $\text{WEIGHTS}(v)$ at the high edge of each node v. *Reduced* WPBDDs are a concise and canonical representation for probability distributions. A WPBDD is reduced if it does not contain isomorphic subgraphs nor redundant nodes. Child $\text{LOW}(v)$ of node v is redundant if $\text{HIGH}(v) = \text{HIGH}(\text{LOW}(v))$ and $\text{WEIGHTS}(v) = \text{WEIGHTS}(\text{LOW}(v))$. Redundant children of node v can be reconstructed using the ordering σ with $\text{REDUN}(\sigma, v) \triangleq \{x_{\sigma(i)} \in \mathcal{A}_X(X) \mid \text{VAR}(v) \prec x_{\sigma(i)} \prec \text{VAR}(\text{LOW}(v))\}$. Node v represents $\text{REP}(v) = (\text{VAR}(v) \vee \bigvee_{x \in \text{REDUN}(\sigma,v)} x) \wedge (\bigwedge_{\omega \in \text{WEIGHTS}(v)} \omega) \wedge f_{\text{VAR}(v)} \vee f_{\overline{\text{VAR}(v)}}$, where $\text{REP}(r) \models E$, i.e., any model of the symbolic representation of the WPBDD at root node r is also a model of the encoding E.

3.2.4 STEP 3: INFERENCE

Using WPBDDs to represent joint probability distribution $P(X)$ defined over variables X, inference is linear in the size of the representation, and is performed by traversing its nodes at most once while evaluating the function that each node and its descendants represent. Node v represents:

$$f(v) = \left(\text{VAL}(\text{VAR}(v)) + \sum_{x \in \text{REDUN}(\sigma,v)} \text{VAL}(x)\right) \cdot \left(\prod_{\omega \in \text{WEIGHTS}(v)} \text{PROB}(\omega)\right) \cdot f(\text{HIGH}(v)) + f(\text{LOW}(v)), \quad (4)$$

where $+$ and $\cdot$ denote algebraic addition and multiplication, respectively, and $\text{VAL}(x_i)$ returns 1 or 0 if x_i is $\top$ or $\perp$, respectively. $\text{VAL}(x_i)$ returns 1 by default and 0 if inconsistent with given evidence. To include "x is equal to its i^{th} value" (Subsection 3.2.2), $\text{VAL}(x_j)$ will return 0 iff $x_j \in \mathcal{A}_X(x) \backslash \{x_i\}$, with $x \in X$ and $x_i \in \mathcal{A}_X(x)$.

4. Parallelism and Inference by WMC

Computational problems that contain a lot of independence are inherently easy to parallelize. The WMC approach to inference does not contain obvious independencies due to the many relations among variables in typical Bayesian networks. This makes it a challenging problem to parallelize. However, several sources of independence in both the compilation and inference phases of WMC can be identified. As WMC is typically applied using monolithic representations, there are bottlenecks obscuring Independencies. We therefore introduce additional independence by employing a partitioning technique that is compatible with various different target representations Dal et al. (2017). This allows a BN to be compiled as a set of subproblems, where each subproblem is compiled in parallel, reducing compilation time to that of the largest subproblem. Recent work has also shown that compilation operations can be parallelized themselves as well Dijk et al. (2013). Finally, we capitalize on the partitioning by traversing independent compiled subproblems in parallel during inference. The following subsections describe the parallelization of the aforementioned high level Independencies (depicted in Figure 1), and many additional ones.

4.1 Parallel Compilation

Knowledge compilation of BNs entails its encoding and the creation of a symbolic target representation using this encoding. Example 3 shows how we can obtain a symbolic propositional formula in CNF that encodes a BN, in which each clause is conjoined. Conjunction ($\wedge$) is both commutative and associative, i.e., the order in the evaluation of a proposition with conjunctions is irrelevant. These properties allow us to perform any conjoin operation with distinct operands in parallel. This is a particularly useful when compiling CNF formulas, as this additionally allows us to create a target representation for each clause, also referred to as a *cube*, in parallel. Conjoining cubes and any pair of distinct intermediate results is done in parallel until the final representation is reached. In the following, parallel compilation is described in more detail.

4.1.1 Partitioning a Bayesian Network

As we approach the final representation, fewer distinct operands become available, reducing opportunities for parallelism. Partitioning a BN into subproblems significantly alleviates this bottleneck, as all subproblems can be compiled in parallel.

Definition 5 *A DAG $G = (V, E)$ is a connected component if there is a path along the edges (ignoring direction) from any node $v \in V$ to any other node $u \in V$, $v \neq u$.*

A DAG $G = (V, E)$ can be decomposed into its connected components, such that for each connected component $C_i = (V_i, E_i)$, there is no path to any of the other connected components $C_j = (V_j, E_j)$, $i \neq j$. The way connected components are defined creates an equivalence relationship on them and any node in a component can act as class representative Lauritzen (1996). If there is only one connected component, then we can still decompose a graph into multiple components by means of a cutset, the (arbitrary) set of edges that breaks up the graph into connected components when removed.

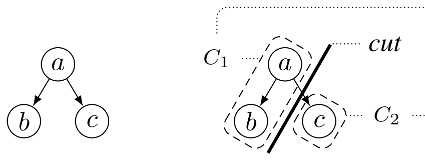
Definition 6 *A cutset S of a DAG $G = (V, E)$, with nodes V and edges E, $S \subseteq E$, decomposes the DAG into connected components by removing the edges in S from E.*

Definition 7 *A partitioning of a DAG $G = (V, E)$, with nodes V and edges E, decomposes G into k subgraphs $C = \{C_1, \ldots, C_k\}$, where each subgraph $C_i = (V_i, E_i)$ is an induced subgraph with V_i nonempty and mutually disjoint, $\bigcup_{j\in\{1,\ldots,k\}} V_j = V$, and $E_i = E \cap (V_i \times V_i)$.*

Proposition 8 *The partitioning of a DAG $G = (V, E)$ into its induced subgraphs $C = \{C_1, \ldots, C_k\}$, $C_i = (V_i, E_i)$, corresponds to the connected components $C' = \{C'_1, \ldots, C'_k\}$ of the graph $H = (V, E \setminus S)$, with $S = \{(u, v) \in E \mid u \in V_i, v \in V_j, i \neq j\}$.*

Proof Since the node sets of G and H are the same, it suffices to consider the edges of the graphs. Let the cutset S be defined as in Definition 6, then we need to prove that $S = E \setminus \bigcup_{i\in\{1,\ldots,k\}} E_i = \{(u, v) \in E \mid u \in V_i, v \in V_j, i \neq j\}$. Take any subgraph $C_i = (V_i, E_i)$, which according to Definition 7 is disjoint with any other subgraph $C_j = (V_j, E_j)$; together the subgraphs cover all the nodes in G and hence also in H. As the subgraphs are induced, they also cover all edges with the exception of those that connect the subgraphs, i.e., $E \setminus \bigcup_{i\in\{1,\ldots,k\}} E_i$. As this corresponds to the definition of a cutset, the subgraphs are the connected components of H. ■

However, a BN $B = (G, P)$ is not just a graph. A subproblem is therefore a component C_i, as the result of partitioning, in addition to probabilities $P(x \mid \mathrm{pa}_G(x))$ for those variables $x \in X$ with the corresponding nodes included in V_i.



(a) BN **(b)** Partitioning

$P(a{=}1)$	$P(a{=}2)$	$P(a{=}3)$
0.6	0.2	0.2

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

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

(c) Conditional Probability Tables

Figure 2 Example Bayesian Network.

Definition 9 *Let BN $B = (G, P)$ be defined over variables X. A function that represents the CPTs associated with variables $U \subseteq X$ depends on variables:*

$$\mathcal{R}_G(U) \;\triangleq\; \bigcup_{x\,\in\,U} \{x\} \cup \mathrm{pa}_G(x). \tag{5}$$

Example 2 *Let BN $B = (G, P)$ be defined over variables $X = \{a, b, c\}$ as illustrated in Figure 2a. The corresponding CPTs are provided in Figure 2c. Figure 2b shows a partitioning, yielding two subproblems given components C_1 and C_2, where $V_1 = \{a, b\}$ and $V_2 = \{c\}$. Probabilities $P(a)$ and $P(b|a)$ for all values of $a, b \in X$ are associated with C_1, and probabilities $P(c|a)$ for all values of $a, c \in X$ are associated with C_2. Component C_1 thus depends on $\mathcal{R}_G(V_1) = \{a, b\}$ and component C_2 on $\mathcal{R}_G(V_2) = \{a, c\}$.*

4.1.2 ENCODING

Each individual probability of x's CPT is captured as a weighted formula $\langle\varphi, \omega\rangle$, where φ would depend on a subset of $\mathcal{A}_X(\mathcal{R}_G(\{x\}))$. A more detailed description of the encoding can be found in Chavira and Darwiche (2008); Dal and Lucas (2017). Here, we illustrate it by an example.

Example 3 *Consider BN $B = (G, P)$ and its partitioning from Example 2. Boolean formula f and g are the propositional (CNF) forms of the PPKBs that encode components C_1 and C_2, respectively:*

$$\begin{aligned} f = {} & (a_1 \vee a_2 \vee a_3) \;\wedge\; (\overline{a_1} \vee \overline{a_2}) \;\wedge\; (\overline{a_1} \vee \overline{a_3}) \wedge \\ & (\overline{a_2} \vee \overline{a_3}) \;\wedge (b_1 \vee b_2) \;\wedge\; (\overline{b_1} \vee \overline{b_2}) \wedge \\ & (\overline{a_1} \vee \omega_1) \;\wedge\; (\overline{a_2} \vee \omega_2) \;\wedge\; (\overline{a_3} \vee \omega_2) \;\wedge (\overline{a_1} \vee \overline{b_1}) \wedge \\ & (\overline{a_2} \vee \overline{b_1} \vee \omega_3) \wedge (\overline{a_2} \vee \overline{b_2} \vee \omega_3) \wedge (\overline{a_3} \vee \overline{b_1} \vee \omega_3) \wedge \\ & (\overline{a_3} \vee \overline{b_2} \vee \omega_3) \end{aligned}$$

$$\begin{aligned} g = {} & (a_1 \vee a_2 \vee a_3) \;\wedge\; (\overline{a_1} \vee \overline{a_2}) \;\wedge\; (\overline{a_1} \vee \overline{a_3}) \wedge \\ & (\overline{a_2} \vee \overline{a_3}) \;\wedge\; (c_1 \vee c_2) \;\wedge\; (\overline{c_1} \vee \overline{c_2}) \wedge \\ & (\overline{a_1} \vee \overline{c_1}) \;\wedge\; (\overline{a_2} \vee \overline{c_1} \vee \omega_4) \wedge \\ & (\overline{a_2} \vee \overline{c_2} \vee \omega_4) \;\wedge\; (\overline{a_3} \vee \overline{c_1}) \end{aligned}$$

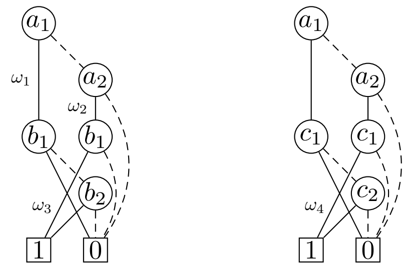
Formula $f \wedge g$ thus represents B and depends on $\mathcal{A}_G(X) = \{a_1, a_2, a_3, b_1, b_2, c_1, c_2\}$ and weights $\{\omega_1, \ldots, \omega_4\}$, where $\textsc{Prob}(\omega_1) = P(a = 1)$, $\textsc{Prob}(\omega_2) = P(a = 2) = P(a = 3)$, $\textsc{Prob}(\omega_3) = P(b = 1, a = 2) = P(b = 2, a = 2) = P(b = 1, a = 3) = P(b = 2, a = 3)$ *and* $\textsc{Prob}(\omega_4) = P(c = 1, a = 2) = P(c = 2, a = 2)$. *Using the same ω_i for identical probabilities per CPT allows us to exploit local structure during compilation and thus find a better factorization. Note that the mutual exclusion constraints (as mentioned in Example 1) of BN variable a must be included in both formulas as a result of partitioning.*

4.1.3 COMPILATION

Using the encoding, we can compile each subproblem to a target representation of choice. Without loss of generality, we will use WPBDDs as target representation for demonstration purposes Dal and Lucas (2017).

Example 4 *Consider the BN, its partitioning and encoding from Example 2 and 3. The compiled representation each subproblem is illustrated in Figure 3a and 3b. The function they represent at their root is recursively defined by Equation 4.*

To summarize, during the compilation process (1) each cube can be created in parallel, (2) the conjoin operations can be performed on each pair of distinct operands (cubes, intermediate results, compiled CPTs) in parallel, and (3) each subproblem can be compiled in parallel.



(a) WPBDD of C_1, given ordering $a_1 \prec a_2 \prec a_3 \prec b_1 \prec b_2$

(b) WPBDD of C_2, given ordering $a_1 \prec a_3 \prec a_2 \prec c_1 \prec c_2$

Figure 3 Compiled WPBDDs of a partitioned BN (Example 4).

4.2 Parallel Inference

In order to perform inference, we employ a compositional approach that connects the compiled representation of each subproblem. Independencies arise based on how they are connected and what evidence is provided.

4.2.1 Composition

Definition 10 *Assume components $C_1, \ldots, C_k$ and component ordering τ. The context of component $C_{\tau(i)}$ is determined by:*

$$\mathcal{Z}_\tau(i) \;=\; \left(\bigcup_{h=1}^{i-1} \mathcal{R}_G(V_{\tau(h)})\right) \cap \left(\bigcup_{j=i}^{k} \mathcal{R}_G(V_{\tau(j)})\right). \tag{6}$$

Compiled subproblems are connected as a *tiered architecture* Dal et al. (2017): a graph with $|\mathcal{D}_X(\mathcal{Z}_\tau(i))|$ nodes on Tier i, where a node represents a distinct traversal of the compiled subproblem associated with component $C_{\tau(i)}$ based on the valuation of context variables $\mathcal{Z}_\tau(i)$. By employing *dynamic programming*, $C_{\tau(i)}$ is thus traversed at most $|\mathcal{D}_X(\mathcal{Z}_\tau(i))|$ times depending on given evidence.

Example 5 *Consider the compiled subproblems in Example 4, the BN $B = (G, P)$ it represents and component ordering τ. The context of Tier 2 is determined by variable $\mathcal{Z}_\tau(2) = V_1 \cap V_2 = \{a\}$. Therefore, we (partially) traverse C_2 at most $|\mathcal{D}_X(\mathcal{Z}_\tau(2))| = 3$ times. We can perform traversals of a particular component given different context valuations in parallel.*

4.2.2 Inference

Probabilistic inference can be performed by evaluating the function represented at the root of the compiled representation, which is defined by Equation 4 for WPBDDs. Using the compositional approach, we perform this evaluation by doing a depth-first traversal starting at the WPBDD root in Tier 1, continuing at the root of the next tier each time the 1 terminal is encountered.

Example 6 *Consider the tiered architecture from Example 5. Figure 5 shows how the 1 terminal in tier 1 is connected to the root node in tier 2, with component ordering $C_1 \prec C_2$. Assume we want to compute a joint probability where variable a is part of the evidence, e.g., $P(a = 1)$. We do this by traversing the tiered architecture as described whilst computing the function it represents. We only traverse the* HIGH *edge of nodes v if* $\text{VAR}(v) = a_1$*, and only traverse the* LOW *edge if* $\text{VAR}(v) \in U \setminus \{a_1\}$*, where $U = \mathcal{A}_X(Z) = \{a_1, a_2, a_3\}$ with $Z = \mathcal{Z}_\tau(2) = \{a\}$. This reduces the traversal of C_2 to $|\mathcal{D}_X(Z \setminus \{a\})| = 1$ time. Note that this could actually be reduced to zero times, because the subproblems are conditionally independent given the evidence.*

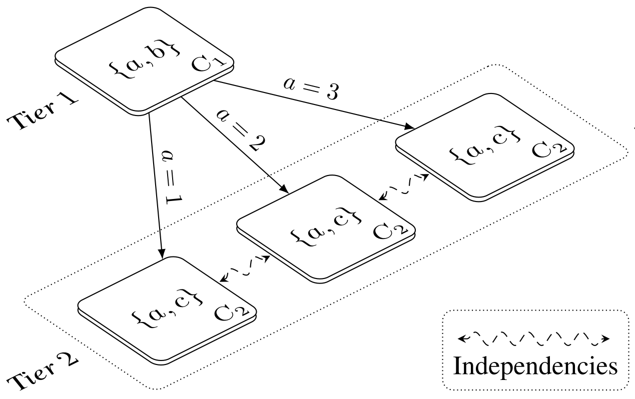


Figure 4 Tiered architecture for the example in Figure 2

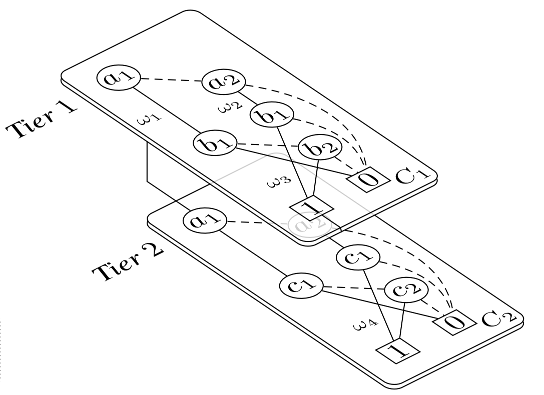


Figure 5 Connected representation of the architecture in Figure 4

When disregarding conditional independence, observe that parallel traversals of compiled representations under different valuations has a practical limitation, as independence is reduced the more evidence we introduce. This does not lead to concern in practice however, because although performance gains through parallelism are reduced, inference itself becomes easier as there are literally less paths and nodes we require to traverse in order to perform inference. The intention the behind proposed parallelization methods is still achieved by reducing inference cost in more complex scenarios.

To introduce more parallelism, we relax the total ordering τ to a partial order by taking its transitive closure. The resulting architecture resembles a DAG, much like a decision diagram. This weakened architecture allows a parent tier node to be immediately traversed when its children have been traversed. We exploit the additional independence for more parallelism. We spawn a task for each traversal of a compiled subproblem representation. A computed table keeps track of the different traversals by mapping a subproblem and a possible valuation for its context to either a task or a result. Before creating a task for traversing a component, the computed table is queried. If it does not contain an entry, the traversal task is created and the new task is immediately stored in the computed table to be later overwritten with its result. If the entry is already mapped to a task, then we yield and wait for its result. Otherwise the result in the computed table is used immediately.

To summarize, (1) compiled subproblems under different valuations of its context can be traversed in parallel, (2) independent subgraphs of the tiered architecture can be traversed in parallel, and (3) posteriors with distinct evidence can be computed in parallel.

5. Empirical Evaluation and Discussion

We have implemented a parallel WMC framework[1] we refer to as PAR and TDPAR, exploiting opportunities for parallelism outlined in Section 4. The general setup for both compilation and inference is the same in that we use a task scheduler and a thread pool (using pthreads), where each thread has its own task queue, to which the scheduler adds work. The scheduler is responsible for resolving dependencies. Performance was measured using several publicly available Bayesian networks on a system with 64 AMD Opteron 6376 processors and 500+ Gb of RAM.

1. Available at `https://github.com/gisodal/wmc`

Bayesian			Compilation time (ms)						Inference time (ms)			
network	$\lvert\mathcal{A}\rvert$	$\lvert P\rvert$	TDPAR	PAR	BNC	CUDD	SDD	ACE	PAR	PAR1	ACE	DLIB
asia	16	36	0.07	0.09	0.53	0.83	1.94	85.90	0.006	0.004	2.340	0.570
sachs	24	228	0.11	0.13	1.85	1.60	31.08	100.49	0.009	0.009	3.149	4.555
std. farm	25	70	0.13	0.14	1.03	1.12	4.94	97.05	0.012	0.008	3.075	2.129
year2000	26	255	0.19	0.17	2.77	1.95	19.57	101.73	0.020	0.015	4.193	18.569
golf	37	388	0.15	0.15	1.55	2.20	16.43	104.45	0.015	0.011	3.065	22.659
poker	43	748	0.17	0.15	2.74	4.29	44.41	105.73	0.014	0.008	2.992	20.029
printer ts	58	272	0.24	0.29	1.80	1.64	7.47	109.26	0.006	0.004	3.067	8.565
boblo	60	316	0.22	0.28	3.73	3.37	39.12	101.29	0.018	0.030	3.912	28.056
child	60	344	0.27	0.32	4.96	4.72	117.16	119.26	0.032	0.035	5.564	27.697
insurance	89	1419	10.10	34.43	107.14	359.29	37149.08	618.06	1.055	2.580	35.325	11685.817
weeduk	90	22611	3.34	2.62	1159.82	425.45	-	3489.63	0.217	0.336	31.141	3155.015
alarm	105	752	0.50	1.32	14.11	19.48	199.85	149.84	0.062	0.211	6.419	35.125
water	116	13484	1485.11	1742.27	766.39	65668.09	-	968.77	-	-	-	-
powerpl	120	432	0.31	0.58	7.73	7.50	159.65	130.35	0.027	0.088	6.740	7.623
carpo	122	554	0.38	0.65	8.04	8.50	63.36	112.27	0.032	0.076	5.959	16.968
win95pts	152	1148	1.33	6.37	31.43	176.65	653.93	154.06	0.160	0.624	9.677	220.080
hepar2	162	2139	1.20	4.66	88.24	449.76	9990.12	281.36	0.165	0.919	18.084	118.217
fungiuk	165	43007	9.83	12.13	3286.51	2010.44	-	12064.18	1.033	1.580	42.140	4681.968
hailfinder	223	3741	2.45	44.46	313.61	3725.44	142839.43	297.19	1.384	6.780	19.801	2751.509
3nt	228	4546	11.99	8.76	46.90	472.41	4288.17	384.06	0.736	1.052	20.618	4817.178
4sp	246	6496	10.62	26.42	119.29	1460.97	60921.00	531.35	1.502	3.236	29.129	-
barley	421	130180	10933.66	19640.55	367274.15	-	-	-	-	-	*	-
andes	440	2308	283.24	1138.58	-	-	-	6918.69	42.195	80.331	132.801	-
pathfinder	520	106432	12.36	62.14	1391.82	19475.49	19739.38	2748.09	0.630	5.729	31.767	4359.722
mildew	616	547158	707.20	661.41	229981.46	-	-	857534.35	72.121	256.653	196.992	-
munin1	992	19226	40180.25	-	-	-	-	-	*	*	*	-
pigs	1323	8427	526.99	-	-	-	-	20019.39	*	*	169.610	-
link	1793	20462	60263.09	-	-	-	-	-	*	*	*	-
diabetes	4682	461069	36120.00	-	-	-	-	941554.61	*	*	1104.04	-

Table 1 All times are in milliseconds. Compilation times are limited to 20 minutes and inference times to 15 seconds (- denotes timeout, * denotes unavailable compiled representation). $|\mathcal{A}|$ is the number of encoding variables and $|P|$ the number of probabilities per BN

Table 1 compares state-of-the-art compilers to two parallel implementations, TDPAR and PAR, producing WPBDDs that are driven by tree and chain (or total) orderings, respectively. They were faster in nearly all cases. Comparisons are made with state-of-the-art compilers SDD[2], CUDD[3], BNC[1] and ACE[4], producing SDDs, OBDDs, WPBDDs and d-DNNFs, respectively. The compilation step crucially depends on the ordering used. To ensure fair comparison, a single ordering was created for each BN with min-hill climbing Chavira and Darwiche (2008), and used by all compilers. SDDs were compiled using a balanced vtree induced by this ordering. Only ACE produced its own orderings.

The compiler underwent several trial and improve cycles. Most notably, we have found that while distributing conjoin operations across available threads in the thread pool, grouping cubes that collectively describe a CPT reduces intermediate representation size. This places a partial order on the conjoin operations, which improved compilation time despite the added dependencies. Secondly, a sizable thread pool suffers from great overhead with such fine grained task scheduling when using a synchronized computed table Nielsen et al. (2000). Each thread thus has its own computed table. This requires more memory, but we did not encounter memory limitations potentially caused by this during experimentation.

2. Available at `http://reasoning.cs.ucla.edu/sdd`
3. Available at `http://vlsi.colorado.edu/~fabio`
4. Available at `http://reasoning.cs.ucla.edu/ace`

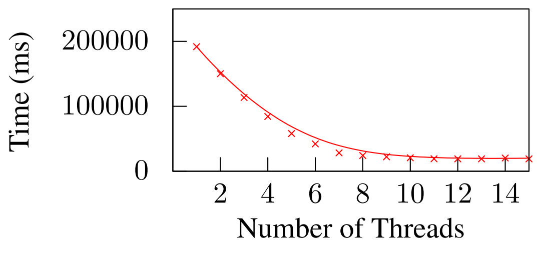


Figure 6 Compilation results for Barley

Compared to single core executions we have achieved 15-fold speedups in compilation, where greater speedups are achieved with mid- to large sized networks. Figure 6 provides a closer examination that shows scaling efficiency is bounded to a certain number of cores, which is to be expected for parallel computations that need to deal with interfering dependencies. Scaling efficiency thus depends on the network, more specifically its size, the dependencies it models and the ordering in which conjoin operations are performed. We hypothesize that a particularly bad order of these operations was chosen for network `water` resulting in increased parallel compilation times. Parallelizing individual conjoin operations in future work might alleviate or remove aforementioned restricting factors to scaling efficiency Dijk et al. (2013).

To evaluate inference performance, we have implemented a parallel model counter for compiled representations produced by PAR. We compare runtimes to model counter ACE and junction tree implementation DLIB[5]. Comparisons to parallel inference methods could unfortunately not be made, as they were not available to us at the time of experimentation. We chose DLIB from among the freely available junction tree implementations for being one of the few written in c++, and additionally having extensive documentation. The use of conditional independence to speedup inference, e.g., through lazy propagation used by the junction tree algorithm or conditioning used during the WMC approach to inference, is not employed by any of the inference methods to facilitate fair comparison.

To compare runtimes, each inference method processed an identical set of probabilistic queries, of which the average runtime is reported in Table 1, where PAR_1 is a single core execution. Partitions are found by recursively decomposing the BN whilst minimizing cutset size and balancing partition size, using simulated annealing and a scoring function. A suited number of partitions thus depends on the network, but is always no less than 2. We observe that small networks have a difficult time dealing with threading overhead, but as networks increase in size, we obtain up to 9-fold speedups.

Tree driven representations as produced by TDPAR and ACE enjoy tighter size upper bounds based on treewidth, compared to pathwidth upper bounds for BDDs Choi et al. (2013), likely resulting in smaller representations and compilation/inference times. It has shown to be difficult to find good total orderings for large networks, but competitive orderings are found for the remaining networks. Single core runtimes for `pigs` and `diabetes` for TDPAR are 91.74ms and 542.29ms compared ACE's 169.61ms and 1104.04ms, respectively. Future work on a parallel model counter for TDPAR's output will certainly provide even greater gains. Overall, for both compilation and inference, significant speedups were attainable through parallelism.

6. Conclusion

Several parallelization opportunities are explored in order to decrease the cost of knowledge compilation and inference. Additional exploitable independencies are introduced by partitioning BNs and using a compositional approach during inference. The benefit of exploited parallelism has been empirically verified. Proposed methods have all been implemented and show 15-fold speedups are possible for compilation. Speedups were achieved with the most efficient implementation compared to SDD, OBDD, WPBDD and d-DNNF compilers. For inference, speedups are less pronounced, still achiev-

5. available at `http://dlib.net/`

ing up to 9-fold speedups. Combined, parallel compilation and inference have shown to provide great improvement among current state-of-the-art methods.

Acknowledgments. The second author is funded by the research program VENI with project number 639.021.649 of the Netherlands Organization for Scientific Research (NWO).

References

R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *Transactions on Computers*, 100: 677–691, 1986.

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172:772–799, 2008.

A. Choi, D. Kisa, and A. 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.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence*, 42:393–405, 1990.

G. H. Dal and P. J. Lucas. Weighted positive binary decision diagrams for exact probabilistic inference. *International Journal of Approximate Reasoning*, 90:411–432, 2017.

G. H. Dal, W. A. Kosters, and F. W. Takes. Fast diameter computation of large sparse graphs using GPUs. In *International Conference on Parallel, Distributed and Network-Based Processing*, pages 632–639, 2014.

G. H. Dal, S. Michels, and P. J. Lucas. Reducing the cost of probabilistic knowledge compilation. In *Proc. of Machine Learning Research*, volume 73, pages 141–152, 2017.

B. D'Ambrosio, T. Fountain, and Z. Li. Parallelizing probabilistic inference: Some early explorations. In *Uncertainty in artificial intelligence*, pages 59–66, 1992.

A. Darwiche. A logical approach to factoring belief networks. *KR*, 2:409–420, 2002.

T. v. Dijk, A. Laarman, and J. v. d. Pol. Multi-core BDD operations for symbolic reachability. *Electronic Notes in Theoretical Computer Science*, 296:127–143, 2013.

S. Funiak, C. Guestrin, R. Sukthankar, and M. A. Paskin. Distributed inference in dynamical systems. In *Advances in Neural Information Processing Systems*, pages 433–440, 2007.

J. Gonzalez, Y. Low, and C. Guestrin. Residual splash for optimally parallelizing belief propagation. In *International Conference on Artificial Intelligence and Statistics*, pages 177–184, 2009.

A. Hommersom, P. J. Lucas, M. Velikova, and G. H. Dal. MoSHCA - my mobile and smart health care assistant. In *e-Health Networking, Applications & Services*, pages 188–192, 2013.

A. V. Kozlov and J. P. Singh. A parallel Lauritzen-Spiegelhalter algorithm for probabilistic inference. In *Proc. of the conference on Supercomputing*, pages 320–329, 1994.

S. L. Lauritzen. *Graphical Models*. Oxford Science Publications, 1996.

A. Mendiburu, R. Santana, J. A. Lozano, and E. Bengoetxea. A parallel framework for loopy belief propagation. In *Genetic and evolutionary computation*, pages 2843–2850, 2007.

V. K. Namasivayam and V. K. Prasanna. Scalable parallel implementation of exact inference in Bayesian networks. In *PDMC*, volume 1, pages 8–pp, 2006.

D. Newman, P. Smyth, M. Welling, and A. U. Asuncion. Distributed inference for latent dirichlet allocation. In *Advances in neural information processing systems*, pages 1081–1088, 2008.

T. D. Nielsen, P.-H. Wuillemin, F. V. Jensen, and U. Kjærulff. Using ROBDDs for inference in Bayesian networks with troubleshooting as an example. In *Uncertainty in artificial intelligence*, 2000.

S. Russell and P. Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall, 1995.

R. D. Shachter, B. D'Ambrosio, and B. Del Favero. Symbolic probabilistic inference in belief networks. In *Proc. of the international conference on Artificial Intelligence (AAAI)*, volume 90, pages 126–131.