ParaGnosis: A Tool for Parallel Knowledge Compilation

Giso H. Dal[1(✉)], Alfons Laarman[2], and Peter J. F. Lucas[1]

[1] Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Enschede, The Netherlands
{g.h.dal,peter.lucas}@utwente.nl

[2] Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands
a.w.laarman@liacs.leidenuniv.nl

Abstract. ParaGnosis (https://doi.org/10.5281/zenodo.7312034, https://zenodo.org/badge/latestdoi/560170574, Alternative url: https://github.com/gisodal/paragnosis, Demo url: https://github.com/gisodal/paragnosis/blob/main/DEMO.md) is an open-source tool that supports inference queries on Bayesian networks through weighted model counting. In the knowledge compilation step, the input Bayesian network is *encoded* as propositional logic and then *compiled* into a knowledge base in decision diagram representation. The tool supports various diagram formats, including the Weighted-Positive Binary Decision Diagram (WPBDD) which can concisely represent discrete probability distributions.

Once compiled, the probabilistic knowledge base can be queried in the inference step. To efficiently implement both steps, ParaGnosis uses simulated annealing to split the knowledge base into a number of partitions. This further reduces the decision diagram size and crucially enables parallelism in both the compilation and the inference steps. Experiments demonstrate that this partitioned approach, in combination with the WPBDD representation, can outperform other approaches in the knowledge compilation step, at the cost of slightly more expensive inference queries. Additionally, the tool can attain 15-fold parallel speedups using 64 cores.

1 Introduction

Hazard and safety analysis are important tools to mitigate risks and prevent disasters in many industries. The complicated interactions between industrial processes and production chains are often modeled in fault trees [31], Bayesian networks [26], and other graphical models [15], that support reasoning under uncertainty. Probabilistic reasoning is however computationally hard. To remedy this problem, *knowledge compilation* aims to find a concise representation that supports getting fast results on queries regarding the same probabilistic model.

Many representation languages [5,14,22,29,32] have been studied for this purpose, analytically as well as experimentally, demonstrating a clear tradeoff

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
G. Caltais and C. Schilling (Eds.): SPIN 2023, LNCS 13872, pp. 22–37, 2023.
https://doi.org/10.1007/978-3-031-32157-3_2

between the succinctness of the language —with often exponential separations— and the tractability of important operations on them. These operations can roughly be divided into *manipulation operations* and *queries*. The former plays an important role in the compilation step, which builds the knowledge base, while the latter are used to query it. For this reason, the budget for manipulation operations is often greater, since this step needs to happen only once.

To the best of our knowledge, ParaGnosis is the first parallel knowledge compilation and inference tool for Bayesian networks with discrete variables. It compiles networks into the Weighted-Positive Binary Decision Diagram (WPBDD). Our chosen parallelization approach through partitioning can reduce the effort spent on compilation because the decomposition of a propositional theory is known to yield smaller symbolic representations, which has previously been shown in model checking [15,17,25,28]. Empirical results with ParaGnosis confirm this [13]. This improvement is offset by a potential increase in the time spent on inference, although our experiments still demonstrate good performance due to the smaller representations. The parallelization approach is orthogonal with regard to target representation languages [11], as we demonstrated with four different representations.

The performance of ParaGnosis compares favorably [10,11] against other knowledge compilers, like SDD, CUDD and ACE, which target SDD [14], OBDD [3] and d-DNNF [16], respectively. The scalability of the tool is good for larger networks and for both compilation and inference, exhibiting over tenfold speedups. ParaGnosis achieves this through its unique compositional approach and use of parallelism.

In this paper, we present the tool ParaGnosis. To help readers understand how the tool works, a high-level overview of the theoretical concepts is given (Sect. 2 and 3) that underly its implementation (Sect. 4). We only offer a user-oriented description of the used BDD, and partitioning and parallelization concepts (see [10,11] for in-depth descriptions and core algorithms). Performance results are presented in Sect. 5. Some examples, figures and definitions are borrowed from previously published works that describe the theoretical foundation of ParaGnosis [10,12]. We finally discuss how this tool relates to others in its field (Sect. 6). Compared to previous work, we have improved ParaGnosis to handle queries other than marginalization, including conditional probabilities and automatic posterior computations of all unobserved variables. We also added several inference query optimizations through parallelism, and a graphical visualization is provided of compiled representations.

2 Background

2.1 Bayesian Networks

A Bayesian network (BN) $\mathcal{B}$ is a probabilistic graphical model that represents a joint probability distribution over its variables. Let $X = \{X_1, \ldots, X_n\}$ be a set of random variables.

Values of a variable X_1 are denoted in lowercase. We denote with $P(X = x)$ the probability that $(X_1, \ldots, X_n) = (x_1, \ldots, x_n)$, i.e. $X_i = x_i$, for $i = 1, \ldots, n$. Let $I \subseteq [n]$, then $X_I = \{X_i \mid i \in I, X_i \in X\}$.

Definition 1 (Bayesian Networks). *A Bayesian network $\mathcal{B} = (\mathcal{G}, P)$ is a DAG $\mathcal{G} = (V, E)$, with nodes V and edges $E \subseteq V \times V$, that models a factorization of joint probability distribution $P(X_V)$ defined over random variables X_V as:*

$$P(X_V = x_V) = \prod_{v \in V} P(X_v = x_v \mid X_{\mathsf{pa}(v)} = x_{\mathsf{pa}(v)}), \tag{1}$$

such that there is a one-to-one correspondence between nodes V and variables X_V, and the conditional probability distribution of $X_v \in X_V$ given its parents $X_{\mathsf{pa}(v)}$ is specified as $P(X_v \mid X_{\mathsf{pa}(v)})$. A Conditional Probability Table (CPT) displays the conditional probabilities $P(X_v \mid X_{\mathsf{pa}(v)})$ of a single variable X_v with respect to its mutually dependent random variables $X_{\mathsf{pa}(v)}$.

Example 1 (Bayesian Network). Fig. 1 shows a BN $\mathcal{B}$ defined over variables $X = \{A, B\}$ (Fig. 1b), its CPTs (Fig. 1a)

$P(A=1)$	$P(A=2)$	$P(A=3)$
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

(a) Conditional probability tables.

$$P(X) = P(B|A)P(A)$$

(b) Bayesian network.

Fig. 1. Bayesian network with local structure.

Posterior probabilities can be computed —a process called inference— using well-known lemmas in probability theory, such as Bayes' theorem $P(X|Y) = \frac{P(X|Y)P(Y)}{P(Y)}$, marginalization $P(X) = \sum_y P(X, Y = y)$, and the factorization property (Definition 1).

2.2 Knowledge Compilation

Bayesian networks represent concise factorizations of probability distributions by using conditional independence assumptions. The size of the factorization has direct implications toward the cost of reasoning, i.e., probabilistic inference. A more expressive model must be used to further improve a BN's factorization in order to exploit additional independences [5]. A prominent way of achieving this is to find a more concise and canonical representation, called a *knowledge base*, such as a Binary Decision Diagram (BDD) [3]. Compiling a BN into a decision diagram (DD) representation is commonly referred to as *knowledge compilation* [16], or simply compilation.

Encoding. BNs are defined over multi-valued domains. Prior to compiling it to a DD, we require an encoding to transition from the multi-valued domain to the Boolean domain. There are multiple ways to do this. We choose to first translate a BN into a Boolean formula with dedicated variables to represent probabilities [5,12].

Conjunctive Normal Form (CNF) is commonly used to facilitate compilation. This CNF is constructed as follows. We create for every $X_v \in X$ a set of atoms $\mathsf{at}(X_i) = \{x_1, ..., x_n\}$. Semantically, $x_i \in \mathsf{at}(X_v)$ represents X_v being equal its i^{th} value. An atom ω_j is introduced for every unique probability in X_v's CPT, i.e., ω_j can refer to multiple distinct entries in X_v's CPT if they represent the same probability. This associated probability is obtained by $\mathsf{pr}(\omega_i) \in [0, 1]$. Function pr returns 1 if no probability is associated with its argument. Finally, a clause is created for each unique valuation of X_v CPT (e.g. for each probability) disjoined with a probability associated ω_j that belongs to that valuation. Also, clauses are added to prevent inconsistent valuation representations, e.g., a variable having multiple values at the same time. We now show this by example, but detailed descriptions can be found in [12]

Example 2 (Bayesian Network encoding).

Let BN $\mathcal{B} = (\mathcal{G}, P)$, with $\mathcal{G} = (V, E)$, be defined over variables $X_V = \{A, B\}$ as described in Example 1. For simplicity, we will focus on just variable A. To encode the BN we create atoms $\mathsf{at}(A) = \{a_1, a_2, a_3\}$. A's CPT has 3 distinct entries and only two distinct probabilities. We create ω_1 for valuations $A = 1$ and create ω_2 for $A = 2$ and $A = 3$, with $\mathsf{pr}(\omega_1) = 0.8$, $\mathsf{pr}(\omega_2) = 0.1$.

The CNF representation follows:

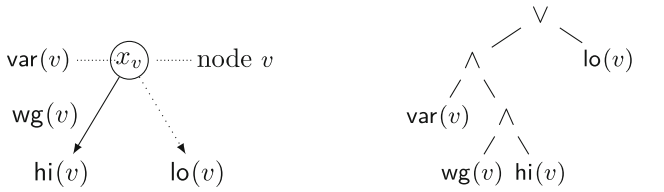
$$(a_1 \vee a_2 \vee a_3) \wedge (\overline{a_1} \vee \overline{a_2}) \wedge (\overline{a_1} \vee \overline{a_3}) \wedge (\overline{a_2} \vee \overline{a_3}) \wedge$$
$$(\overline{a_1} \vee \omega_1) \wedge (\overline{a_2} \vee \omega_2) \wedge (\overline{a_3} \vee \omega_2)$$

The first row is solely concerned with preserving valuation consistency (A can only have one value). The second row has a clause for each valuation/probability. The encoding includes the following models for variable A:

	Models					Weights
1	a_1	$\overline{a_2}$	$\overline{a_3}$	ω_1	$\overline{\omega_2}$	$\mathsf{pr}(\omega_1) \cdot \mathsf{pr}(\overline{\omega_2}) = 0.8 \cdot 1 = 0.8$
2	$\overline{a_1}$	a_2	$\overline{a_3}$	$\overline{\omega_1}$	ω_2	$\mathsf{pr}(\overline{\omega_1}) \cdot \mathsf{pr}(\omega_2) = 1 \cdot 0.1 = 0.1$
3	$\overline{a_1}$	$\overline{a_2}$	a_3	$\overline{\omega_1}$	ω_2	$\mathsf{pr}(\overline{\omega_1}) \cdot \mathsf{pr}(\omega_2) = 1 \cdot 0.1 = 0.1$

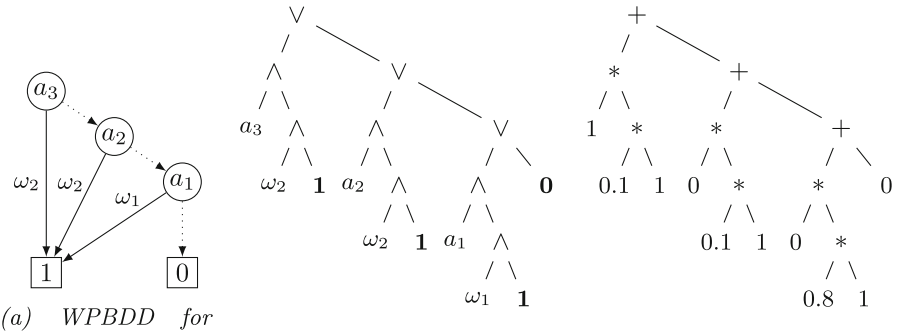
Note that the weighted model count sums to 1.0 for this selection of models. However, there are more models, e.g., model a_1,$\overline{a_2}$,$\overline{a_3}$,ω_1,ω_2, model $\overline{a_1}$,a_2,$\overline{a_3}$,ω_1,ω_2, etc. Only minimal models sum to 1.0, i.e., models with the most amount of negations.

Compilation. Now that we have an encoding, we can look at its compilation to a WPBDD in particular. A WPBDD is an ordered BDD that represents a concise factorization of a Boolean formula f as a (rooted) directed acyclic graph with decision nodes, and two terminal nodes labeled with 1 and 0. Each non-terminal node v is labeled with a Boolean variable $\mathsf{var}(v) = x_v$ and has two children, $\mathsf{hi}(v)$ and $\mathsf{lo}(v)$, with a set of weight variables $\mathsf{wg}(v)$ at the edge to node $\mathsf{hi}(v)$. Edges to nodes $\mathsf{hi}(v)$ and $\mathsf{lo}(v)$ are solid and dotted, respectively, as shown in Fig. 2a. Its logical equivalent is shown in Fig. 2b. Each root-terminal path contains a variable at most once, and in a particular total or partial order.



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

Fig. 2. The semantics of a WPBDD node.



(a) WPBDD for variable A. *(b) Logical circuit.* *(c) Arithmetic circuit.*

Fig. 3. Performing inference in Example 3.

A CNF encoding as described above acts as an entry point for the language compiler [20]. Such compilers target different variations of DDs.

The respective DD is built using the typical bottom-up strategy [3], by applying DD operations to construct a DD representing the encoded formula from the previous step. The process of compiling into a respective DD is by far the most expensive operation, compared to the inference step, which is linear in the size of the knowledge base as desired.

Inference. Inference is performed through *Weighted Model Counting* on the DD, WMC for short [5,17]. This process sums the weight of every variable assignment. In the decision diagram, these assignments are represented by paths and the weights by edge labels (see the example WPBDD in Fig. 1 (c)). Since these paths often overlap in the DD structure, inference through model counting is linear in the size of the target representation [16].

Let's look at a WPBDD compilation and inference example. A WPBDD exactly represents the encoding provided. In order to perform inference we can trivially transform the logical circuit that the WPBDD represents into an arithmetic circuit.

Example 3 (Compilation and inference). Consider only variable A in Example 1. The compiled representation is shown in Fig. 3a for variable ordering $a_3 \prec a_2 \prec a1$. We have not optimized the representation in order to make the upcoming discussion easier. Reduction rules specific to WPBDDs allow the removal of the a_2 node to further reduce its size. Each path from the root to the **1**-terminal semantically implies evidence. There are three possible paths shown below. If we have evidence prior to traversing the compiled representation, we only consider the paths that are consistent with the evidence.

Path	Logic	Semantics
$a_3 \rightarrow \mathbf{1}$	$\overline{a_1} \wedge \overline{a_2} \wedge a_3$	$A = 3$
$a_3 \dashrightarrow a_2 \rightarrow \mathbf{1}$	$\overline{a_1} \wedge a_2 \wedge \overline{a_3}$	$A = 2$
$a_3 \dashrightarrow a_2 \dashrightarrow a_1 \rightarrow \mathbf{1}$	$a_1 \wedge \overline{a_2} \wedge \overline{a_3}$	$A = 1$

The underlying logical circuit is shown in Fig. 3b (obtained with the circuit in Fig. 2b). To perform inference, we need to instantiate the equivalent arithmetic circuit. Figure 3c shows the instantiated circuit that allows us to compute $P(A = 3) = 0.1$, by setting valuation $(a_1, a_2, a_3) = (0, 0, 1)$.

3 Weighted Model Counting Methodologies

We distinguish between two methodologies within the field of probabilistic inference through WMC. Traditional (or monolithic) Weighted Model Counting and Compositional Weight Model Counting. The former is described in Sect. 2. The latter is introduced by ParaGnosis, and expands traditional compilation with partitioning. The purpose of partitioning is to alleviate high compilation costs. Figure 4 shows an overview of the underlying principles behind the two methodologies, divided into two steps: *compilation* and *inference*. On the left, we have the monolithic and on the right, we have the compositional approach.

In-depth descriptions of compositional WMC in scrupulous detail can be found in [10], however, we consider it to be out of scope for this article and will proceed to give a more high-level description here. The concrete implementation of ParaGnosis is described in Sect. 4.

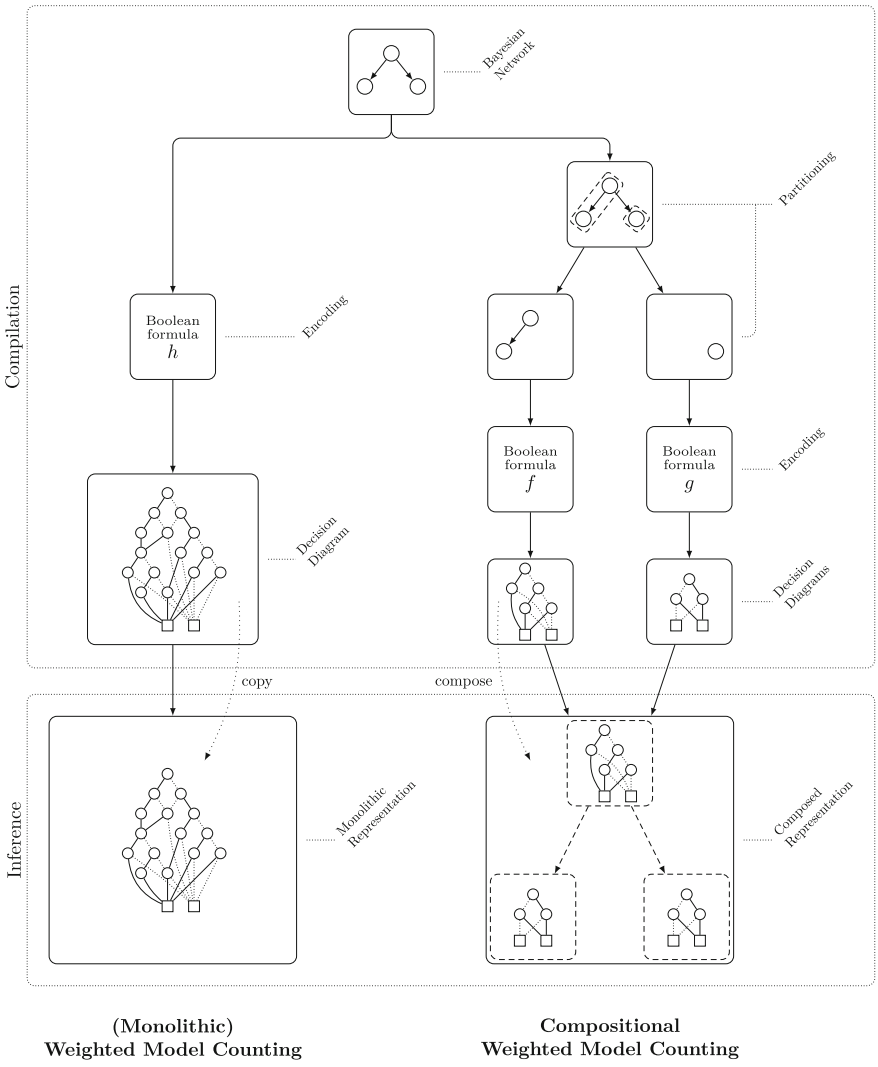


Fig. 4. Compilation and inference process. ParaGnosis uses the compositional approach depicted on the right.

3.1 Compilation to a Compositional Knowledge Base

We first describe compilation as all steps required to obtain a DD. In addition to traditional compilation, ParaGnosis introduces partitioning to further improve overall performance [13]. It finds a partitioning that decomposes the BN into independent components. A component is a set of nodes that are not connected to nodes outside of the component by removing edges from the BN. The fewer removals the better, with regard to inference complexity. This method thus keeps

CPTs intact, as the partitioning happens on a BN node level, not deeper. We give a partitioning demonstration later, using Example 4. With the partitioning in hand, the following steps can be performed independently, per partition. Each partition is considered an independent BN from this point on.

With monolithic compilation, we would only be able to amortize the cost of compilation by performing many inference queries. With partitioned compilation, we shift some of this cost over to the inference side, yielding overall performance improvements in cases where we would traditionally not be able to compile a knowledge base due to time or memory restrictions [13].

3.2 Inference

After compilation, we arrive at the inference step.

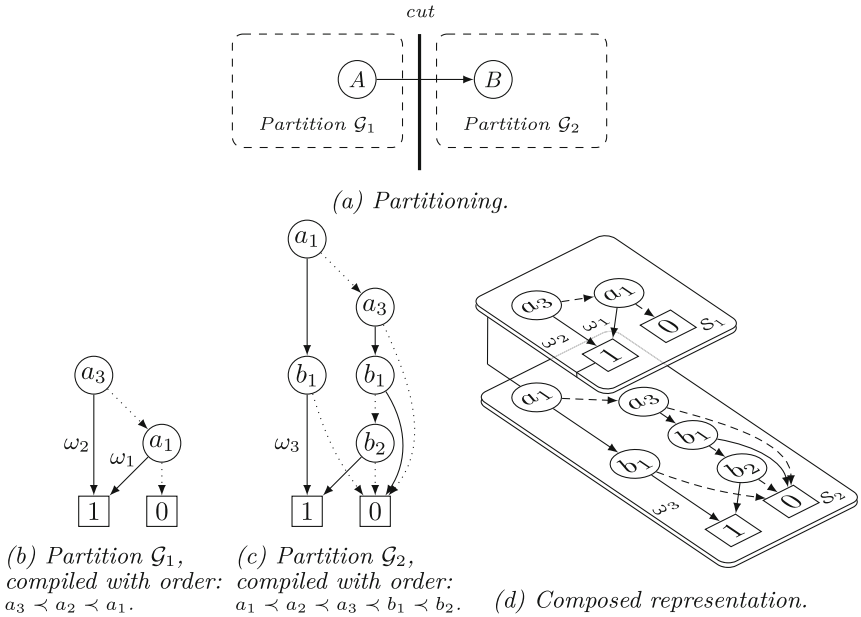
In case the user chose to partition the BN during the compilation step, ParaGnosis performs an adapted WMC step to recombine the compiled DDs. A partition's representation should be connected to another when they share a common variable. This implies that we need to traverse partitions according to some partial order. It can be represented by a tree, which we suitably refer to as a composition-tree [10].

The order in which we choose to traverse partitions (combined with common variable relations) determines how they are connected. As we traverse one partition, its sink is connected to the next partition's root. Now that all partitions form one connected component, we can proceed as previously described with a traversal we are already familiar with from WMC. We discuss how partitioning influence inference complexity in Sect. 6.

Example 4 (Partitioned compilation and composition). Consider the BN from Example 1. Figure 5a shows a partitioning. Each partitioning can then independently be compiled to a target representation, i.e. a WPBDD, OBDD, SDD, etc. Figure 5b and 5c show the WPBDD representations of the partitions. Note that partition $\mathcal{G}_2$ also includes $\mathtt{at}(A) = \{a_1, a_2, a_3\}$ in its compiled representation, because B depends on A in the BN. Figure 5d shows how the compiled representations are connected to form one connected component. This facilitates its traversal as a monolithic structure and enables WMC.

4 ParaGnosis

We present the overview of ParaGnosis's architecture in Fig. 6, including its inputs and outputs. It implements the principles outlined in Fig. 4 and Sect. 3. ParaGnosis is a collection of two applications written in C++. The compilation step is implemented by the `COMPILER`, whereas the inference step is implemented by the INFERENCE ENGINE. We provide a high-level view of the implementations in the following sections. In-depth descriptions on the used partitioning technique are provided in [10,13], and how they created independencies are exploited through parallelism in [11].



(a) Partitioning.

(b) Partition $\mathcal{G}_1$, compiled with order: $a_3 \prec a_2 \prec a_1$.

(c) Partition $\mathcal{G}_2$, compiled with order: $a_1 \prec a_2 \prec a_3 \prec b_1 \prec b_2$.

(d) Composed representation.

Fig. 5. Partitioning a BN for Example 4.

ParaGnosis comes with a wrapper script called `pg`, simplifying compilation and inference steps. Compiling the 'icy roads' network to a WPBDD and directly visualizing it (using DOT) is as easy as running the following command:

```
> pg compile icy_roads --method wpbdd --dot
```

Computing posteriors for variables 'Holmes' and 'Icy' can be achieved by:

```
> pg inference --posteriors="Holmes,Icy" icy_roads
```

An extended tool demo is also available[1].

4.1 The `COMPILER`

As shown in Fig. 6, the `COMPILER` takes as input a discrete BN in the original HUGIN's `.net` format [23]. The compiler is responsible for creating a decision diagram from the provided BN and writes these to output files. The principles outlined in Sect. 3 provide an orthogonal framework with regard to the target representation. However, ParaGnosis chooses to target *Weighted Positive Binary Decision Diagrams* (WPBDD), because it is a dedicated representation for probabilistic inference. (We discuss differences with other representations in Sect. 6.)

[1] Demo: https://github.com/gisodal/paragnosis/blob/main/DEMO.md.

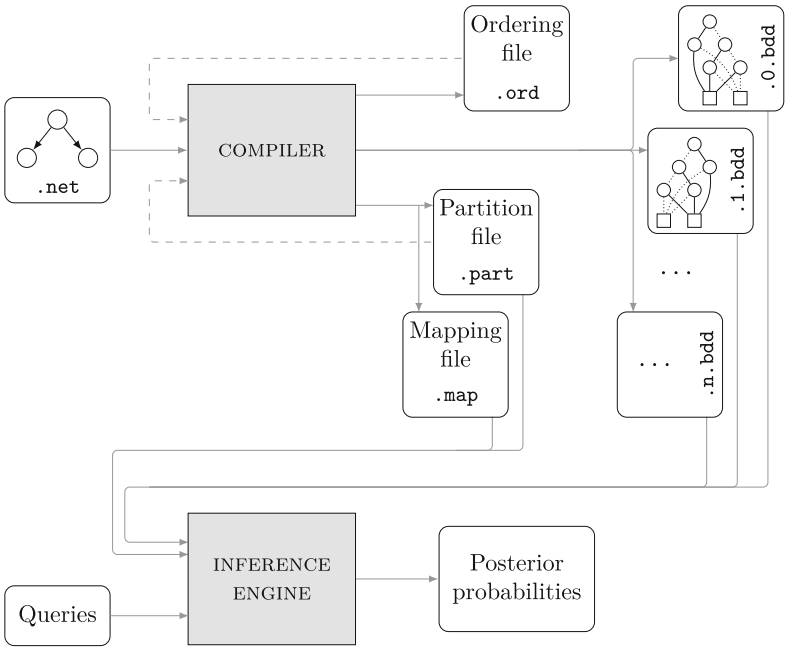


Fig. 6. ParaGnosis's architecture.

The compiler can introduce a partitioning to further improve overall performance [13]. With a user-provided number of partitions, a partitioning is found for the BN. Using simulated annealing, this partitioning is optimized by minimizing the sum of the tree-width of all partitions. Tree-width is a metric commonly used to indicate the complexity of BNs [6]. Optionally, a partitioning can also be provided by the user. With the partitioning in hand, the following steps can be performed in parallel, per partition. Theoretically, compilation is as fast as the slowest compiling partition [11].

Compilation is critically dependent upon a good variable ordering. The size of the resulting WPBDD (or any other ordered DD) is determined by it. The compiler can optionally be provided with a variable ordering as input. When not provided by one, an ordering is created automatically using the min-fill heuristic [5]. Further refinement can be attained when the user requires it. It is achieved by minimizing the tree-width of candidate orderings, using simulated annealing.

4.2 The INFERENCE ENGINE

The inference engine takes as input probabilistic queries, and the compiler's output (Fig. 6). At its core, the inference engine is able to perform probabilistic inference through marginalization. It does so using WMC (Sect. 3.2. Given a user-provided set of probabilistic queries, the corresponding marginalizations

are performed. Prior to every WMC run, we need to instantiate the variables in the WPBDD, reflecting the marginalization we want to achieve. Proper variable instantiations in the WPBDD are attained by using the COMPILER's variable mapping to map BN variables to WPBDD variables.

Notice that each marginalization is an independent run of WMC, just with different variable instantiations. A trivial optimization here is that we can run every call to WMC in *parallel*. Theoretically reducing the cost of computing the posterior probabilities for all instantiations of unobserved variables combined to the single WMC call that requires the most resources.

Parallelism is notoriously difficult to exploit in sparse graphs such as BNs [4,9]. Introducing a partitioning during the compilation step can achieve the required independence such that parallelism can be exploited at the level of individual WMC runs [13]. Compiled partitions are composed using a composition-tree. This tree is the structure by which we traverse each partition's corresponding WPBDD [10]. The independence among a parent's children in the composition-tree can be used to run WMC in those parts in parallel, i.e., independent sub-trees can run in parallel [11].

5 Performance of ParaGnosis

We re-report some experimental results on the performance of ParaGnosis to substantiate our claims on its performance. All above experiments ran on a system with AMD Opteron 6376 processors, with 500+ Gb of RAM.

We have previously [10,12] compared WPBDD compilation costs to those of OBDDs and ZBDDs (using the CUDD 3.0.0 library), SDDs [14] (using the SDD 1.1.1 library), and OBDDs.[2,3,4] Much care is taken to create a fair comparison between libraries. We set up a head-to-head procedure, where each compiler was swapped-in and out. This ensures that the same steps are performed to produce each respective representation. The same ordering is thus tested with compiler and representation. Details of the experimental setup can be found in [10,13]. The results are again reported in Table 1 and show favorable runtimes for the WPBDD approach.

For inference, in previous work [10], we chose to compare to Ace (version 3.0) and to the junction tree algorithm using the publicly available Dlib library (version 18.18).[5,6] In previous works, ParaGnosis has also compared favorably to the HUGIN library (version 8.4).[7] [12]

We compared the aforementioned methods by measuring the speed by which they could solve the same set of probabilistic queries. Queries are created randomly, i.e., with a random number of observed variables and a random configuration. A query is created and fed to each method [10].

[2] CUDD is available at http://vlsi.colorado.edu/~fabio/.
[3] The SDD compiler is available at http://reasoning.cs.ucla.edu/sdd/.
[4] The WPBDD compiler is available at https://github.com/gisodal/paragnosis/.
[5] Ace is available at http://reasoning.cs.ucla.edu/ace/.
[6] Dlib is available at http://dlib.net/.
[7] HUGIN is available at http://www.hugin.com/.

Table 1. Compilation runtime (milliseconds), where $|\mathsf{at}(X)|$ are the number encoding variables for BN variables X, - implies compilation failure by exceeding 15 min or 500 Gb of RAM memory.

Bayesian Network	$\lvert\mathsf{at}(X)\rvert$	Partitioned t-WPBDD	t-WPBDD	OBDD	SDD	d-DNNF
sachs	24	0.148	**0.100**	1.932	29.119	92.179
student farm	25	0.117	**0.106**	1.403	4.646	118.641
printer ts	58	0.230	**0.198**	1.757	6.628	97.956
boblo	60	**0.213**	**0.213**	3.792	27.920	118.202
child	60	**0.195**	0.331	4.564	96.344	117.620
insurance	89	**0.494**	20.365	267.967	12337.980	680.771
weeduk	90	18.415	**6.091**	429.110	-	3472.012
alarm	105	**0.407**	0.467	10.085	400.158	157.163
water	116	**5.185**	1635.935	16034.149	-	1009.578
powerplant	120	**0.268**	0.361	9.409	119.856	159.193
carpo	122	0.426	**0.420**	13.910	119.122	137.955
win95pts	152	**0.874**	1.386	193.919	902.473	173.762
hepar2	162	**1.444**	1.567	414.316	31119.984	287.980
fungiuk	165	**22.186**	45.559	1667.940	-	12193.593
hailfinder	223	**1.061**	3.748	422.270	14350.353	354.151
3nt	228	**0.696**	2.397	344.902	4259.798	424.939
4sp	246	**0.849**	5.090	991.545	7041.476	573.015
barley	421	**611.830**	23290.743	-	-	-
mainuk	421	**584.409**	23443.483	-	-	-
andes	440	**3.267**	224.648	-	-	7785.916
pathfinder	520	**17.279**	18.057	22741.434	137591.643	2813.821
mildew	616	**42.611**	576.852	244920.444	-	885305.099
munin1	992	**11.929**	53899.548	-	-	-
pigs	1323	**4.444**	348.872	-	-	20623.511
link	1793	**174.897**	19412.863	-	-	-
diabetes	4682	**1297.221**	2622.924	-	-	-
munin2	5376	**96.809**	926.789	-	-	235544.805
munin3	5601	**52.350**	1088.710	-	-	102338.718
munin4	5645	**718.358**	2565.931	-	-	162054.255
munin	5651	**1407.531**	2360.196	-	-	161133.160

The results are again reported in Table 2 and show favorable runtimes for the WPBDD approach, even for the partitioned approach. It shows the average runtime of a query across the aforementioned set of tested queries. While partitioning in theory can shift some workload from compilation to inference, it can also further reduce the size of the knowledge base, which explains these results.

In future experimental work, it would be interesting to compare ParaGnosis against Mora [2] which takes the new approach by reducing the inference problem to a probabilistic program. This new approach has not yet been compared to any of the above tools.

Table 2. Inference runtime averaged per query (milliseconds), where - implies inference failure by exceeding 15 min, or compilation failure.

Bayesian Network	Partitioned t-WPBDD	t-WPBDD	d-DNNF	DLIB
sachs	0.023	**0.011**	2.975	?
student farm	0.035	**0.016**	2.813	2627.214
printer ts	0.007	**0.006**	2.852	?
boblo	0.054	**0.034**	3.713	3545.921
child	0.345	**0.036**	5.695	3545.921
insurance	7.486	**1.874**	36.884	?
weeduk	0.607	**0.262**	30.908	3586.316
alarm	0.187	**0.115**	6.513	3547.504
water	**25.176**	74.135	33.512	-
powerplant	**0.025**	0.032	6.249	3393.056
carpo	0.138	**0.037**	5.739	3515.298
win95pts	**0.371**	0.635	9.680	3220.060
hepar2	**0.247**	1.133	18.659	3463.509
fungiuk	18.822	**5.290**	42.814	3567.136
hailfinder	25.137	**1.747**	19.618	2448.653
3nt	18.411	**8.250**	21.559	?
4sp	5.748	**1.277**	30.043	?
barley	**1399.542**	1798.278	-	-
mainuk	**1377.117**	1782.512	-	-
andes	178.610	185.205	**144.691**	?
pathfinder	5.394	**0.639**	30.686	?
mildew	351.788	552.496	**208.582**	-
munin1	7183.857	**6836.045**	-	-
pigs	248.866	**70.266**	179.088	-
link	**9893.431**	-	-	-
diabetes	968.839	**618.687**	-	-
munin2	**107.788**	207.768	384.055	-
munin3	828.535	**140.398**	263.751	-
munin4	**280.275**	318.687	402.651	-
munin	377.117	**302.675**	416.733	-

6 Discussion

ParaGnosis shows that parallelism can benefit knowledge compilation and inference for Bayesian networks. Our chosen parallelization approach is based

on partitioning. This approach has previously been used to speed up symbolic model checking [19,25,28]. We additionally find that the partitioning introduces a tradeoff between compilation times and inference times, sacrificing some performance in inference to gain parallel scalability.

The computational complexity of inference is linear in the size of the target representation [16]. However, it increases when partitioning is employed. A compiled partition in a composition-tree must be exponentially traversed in the size of the cutset that separates it from its parent. One traversal for each instantiation of the cutset variables. However, this is compensated by a number of principles. (1) the combined size of all partition BDDs is significantly reduced compared to the monolithically compiled BDD [13]; (2) partition BDDs only represent a portion of its total, reducing traversal resources; (3) each child instantiation can be traversed in parallel, potentially reducing traversal cost of exponential traversals to the cost of 1 [10]; (4) as partition BDDs are small, cache locality start to play an important role, giving an advantage over monolithic BDDs [11]. Beyond the complexity introduced by cutsets, inference remains linear. Small cutsets can therefore play a crucial role in performance. Separate empirical investigations on partitioning and parallelism show their respective contribution to ParaGnosis's performance [10,11].

The parallelization approach of ParaGnosis is orthogonal to the chosen target representation of the knowledge base, which we demonstrated by using it for four different target representations [11]. As a consequence, the approach is to a certain extent orthogonal to the exploitation of local structure [30] by those representations, as local structure can still be exploited within the partitioned subproblems. For instance, we showed that causal dependence is fully exploited when using decision diagram representations in the partitions.

For target representations, many choices exist [5,7,14]. Since our partitioning technique exploits the treewidth [6] of the representation [10, §5], and the algorithms are based on message passing [10, §4], other representations like ADD and d-DNNF can be parallelized alike. While our earlier work [11] compared the performance of ParaGnosis against various of these other representations, showing competitive performance, here we will point out some differences between the other representations and suggest future work.

Like AADD [29], and its similar cousins SLDD [34], FEVBDD [32] and QMDD [24], our target representation WPBDD factors out probabilities on the edges of the diagram, resulting in more succinctness than for instance achieved with ADD [1], QuiDD [33] and MTDD [8]. However, unlike AADD, it only factors according to the structure of the Bayesian network, which sacrifices succinctness but ensures exact representation of the floating point numbers. The latter can be quite important in practice, as rounding errors from manipulation operations can rapidly propagate in the discrete data structure, resulting in numerical instability [21,27,35].

The effects of different variable orders are known to be crucial in many representation languages. Representations like d-DNNF [5] and SDD [14], allow more freedom in the order and could potentially improve the results of ParaGnosis.

Early versions of ParaGnosis also tried different parallelization approaches, like the fine-grained task-based scheduling of Sylvan [18], which has shown that good parallel scalability is possible for model checking problems in BDDs, ADDs (also called Multi-Terminal DDs), and MDDs (Sylvan uses a version called List DD [19]). In future work, we hope to establish why this approach did not yield good performance for knowledge compilation as well.

References

1. Bahar, R.I., et al.: Algebraic decision diagrams and their applications. In: Proceedings of 1993 International Conference on Computer Aided Design, pp. 188–191 (1993)
2. Bartocci, E., Kovács, L., Stankovič, M.: Mora - automatic generation of moment-based invariants. In: TACAS 2020. LNCS, vol. 12078, pp. 492–498. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45190-5_28
3. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. Trans. Comput. **100**, 677–691 (1986)
4. Ceylan, I.I., Darwiche, A., Van den Broeck, G.: Open-world probabilistic databases: semantics, algorithms, complexity. Artif. Intell. **295**, 103–123 (2021)
5. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artif. Intell. **172**, 772–799 (2008)
6. Chen, Y., Darwiche, A.: On the definition and computation of causal treewidth. In: The 38th Conference on Uncertainty in Artificial Intelligence (2022)
7. Choi, Y.K., Santillana, C., Shen, Y., Darwiche, A., Cong, J.: FPGA acceleration of Probabilistic Sentential Decision Diagrams with high-level synthesis. ACM Trans. Reconfigurable Technol. Syst. **16**(2), 1–22 (2022)
8. Clarke, E.M., Fujita, M., McGeer, P.C., McMillan, K., Yang, J.C.-Y., Zhao, X.: Multi-terminal binary decision diagrams: An efficient data structure for matrix representation (2001)
9. Dal, G.H., Kosters, W.A., Takes, F.W.: Fast diameter computation of large sparse graphs using GPUs. In: International Conference on Parallel, Distributed and Network-Based Processing, pp. 632–639 (2014)
10. Dal, G.H., Laarman, A.W., Hommersom, A., Lucas, P.J.: A compositional approach to probabilistic knowledge compilation. Int. J. Approximate Reason. **138**, 38–66 (2021)
11. Dal, G.H., Laarman, A.W., Lucas, P.J.F.: Parallel probabilistic inference by weighted model counting. In: International Conference on Probabilistic Graphical Models, pp. 97–108 (2018)
12. Dal, G.H., Lucas, P.J.F.: Weighted positive binary decision diagrams for exact probabilistic inference. Int. J. Approx. Reason. **90**, 411–432 (2017)
13. Dal, G.H., Michels, S., Lucas, P.J.F.: Reducing the cost of probabilistic knowledge compilation. Mach. Learn. Res. **73** 141–152 (2017)
14. Darwiche, A.: SDD: A new canonical representation of propositional knowledge bases. In: International Joint Conference on Artificial Intelligence, vol. 22, pp. 819 (2011)
15. Darwiche, A., Hirth, A.: On the (complete) reasons behind decisions. J. Logic, Lang. Inform. Ages **18**, 1–26 (2022)
16. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. **17**, 229–264 (2002)

17. Darwiche, A., Marquis, P.: On quantifying literals in Boolean logic and its applications to explainable AI. J. Artif. Intell. Res. **72**, 285–328 (2021)
18. van Dijk, T., Laarman, A., van de Pol, J.: Multi-core BDD operations for symbolic reachability. Electron. Notes Theor. Comput. Sci. **296**, 127–143 (2013)
19. van Dijk, T., van de Pol, J.: Sylvan: multi-core framework for decision diagrams. Int. J. Softw. Tools Technol. Transfer **19**, 675–696 (2017)
20. Dudek, J., Phan, V., Vardi, M.: ADDMC: Weighted model counting with algebraic decision diagrams. In: International Conference on Artificial Intelligence, pp. 1468–1476 (2020)
21. Hillmich, S., Burgholzer, L., Stogmuller, F., Wille, R.: Reordering decision diagrams for quantum computing is harder than you might think. In: International Conference on Reversible Computation, pp. 93–107 (2022)
22. Hitzler, P., Sarker, M.K.: Tractable Boolean and arithmetic circuits. Neuro-Symbolic Artif. Intell.: State Art **342**, 146 (2022)
23. Madsen, A.L., Lang, M., Kjaerulff, U.B., Jensen, F.: The Hugin tool for learning Bayesian networks. In: European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty, pp. 594–605 (2003)
24. Miller, D.M., Thornton, M.A.: QMDD: A decision diagram structure for reversible and quantum circuits. In: 36th International Symposium on Multiple-Valued Logic, pp. 30–36 (2006)
25. Narayan, A., Jain, J., Fujita, M., Sangiovanni-Vincentelli, A.: Partitioned ROBDDs-a compact, canonical and efficiently manipulable representation for Boolean functions. In: Proceedings of International Conference on Computer Aided Design, pp. 547–554 (1996)
26. Judea Pearl. Bayesian networks (2011)
27. Peham, T., Burgholzer, L., Wille, R.: Equivalence checking paradigms in quantum circuit design: A case study. In: Proceedings of the 59th Design Automation Conference, pp. 517–522 (2022)
28. Sahoo, D.: A partitioning methodology for BDD-based verification. In: International Conference on Formal Methods in Computer-Aided Design, pp. 399–413 (2004)
29. Sanner, S., McAllester, D.: Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In: International Joint Conference on Artificial Intelligence 2005, pp. 1384–1390 (2005)
30. Shih, A., Choi, A., Darwiche, A.: Compiling Bayesian network classifiers into decision graphs. In: International Conference on Artificial Intelligence, vol. 33, pp. 7966–7974 (2019)
31. Sokukcu, M., Sakar, C.: Risk analysis of collision accidents during underway sts berthing maneuver through integrating fault tree analysis (FTA) into Bayesian network. Appl. Ocean Res. **126**, 103–124 (2022)
32. Tafertshofer, P., Pedram, M.: Factored edge-valued binary decision diagrams. Formal Methods Syst. Design **10**(2), 243–270 (1997)
33. Viamontes, G.F., Markov, I.L., Hayes, J.P.: Improving gate-level simulation of quantum circuits. Quant. Inform. Process., **2**, 347–380 (2003)
34. Wilson, N.: Decision diagrams for the computation of semiring valuations. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, pp. 331–336 (2005)
35. Zulehner, A., Hillmich, S., Wille, R.: How to efficiently handle complex values? Implementing decision diagrams for quantum computing. In: 2019 International Conference on Computer-Aided Design, pp. 1–7 (2019)