Probabilistic Inference by Projected Weighted Model Counting on Horn Clauses

Alexandre Dubray
Institute of Information and Communication Technologies, Electonics and Applied Mathematics (ICTEAM), UCLouvain, Belgium

Pierre Schaus
Institute of Information and Communication Technologies, Electonics and Applied Mathematics (ICTEAM), UCLouvain, Belgium

Siegfried Nijssen
Institute of Information and Communication Technologies, Electonics and Applied Mathematics (ICTEAM), UCLouvain, Belgium

Abstract

Weighted model counting, that is, counting the weighted number of satisfying assignments of a propositional formula, is an important tool in probabilistic reasoning. Recently, the use of projected weighted model counting (PWMC) has been proposed as an approach to formulate and answer probabilistic queries. In this work, we propose a new simplified modeling language based on PWMC in which probabilistic inference tasks are modeled using a conjunction of Horn clauses and a particular weighting scheme for the variables. We show that the major problems of inference for Bayesian Networks, network reachability and probabilistic logic programming can be modeled in this language. Subsequently, we propose a new, relatively simple solver that is specifically optimized to solve the PWMC problem for such formulas. Our experiments show that our new solver is competitive with state-of-the-art solvers on the major problems studied.

2012 ACM Subject Classification Mathematics of computing → Probabilistic inference problems; Computing methodologies → Probabilistic reasoning; Mathematics of computing → Bayesian networks

Keywords and phrases Model Counting, Bayesian Networks, Probabilistic Networks

Digital Object Identifier 10.4230/LIPIcs.CP.2023.11

1 Introduction

Weighted model counters are systems that calculate the weighted number of assignments that satisfy a formula in conjunctive normal form (CNF). Weighted model counting (WMC) is a hard problem: it is #P complete, and hence building efficient systems for this task is a challenge.

An important application of WMC is probabilistic inference, that is, calculating the probability of an observation according to a given probabilistic model. With increasing sizes of probabilistic models, the performance of inference remains important. Model counters have already been used to solve probabilistic inference problems on

- Bayesian networks (BNs) [6, 9, 20];
- Probabilistic networks (PNs) [12, 27]
- Probabilistic logic programs (PLPs) [13].

In each of these studies, approaches were proposed for how to model a probabilistic inference task as a WMC task on a CNF formula. Unfortunately, the resulting models are not always simple. For Bayesian networks the CNF formulas are polynomial in size given the size of the BN, but a number of papers have proposed increasingly complex models to make solving efficient [2, 4, 5]; for PNs and PLPs the CNF can even be exponentially larger than the

original probabilistic model [27]. A challenge remains how to model inference tasks in a simple manner and obtain good performance at the same time.

A promising solution to this modeling problem may be the use of *projected (weighted) model counting* (PWCM) [1] as a general approach for probabilistic inference; it was shown that using PWMC a CNF of polynomial size can be used to solve reachability problems on PNs [12]. Where in weighted model counting a sum is calculated over all models of a formula F, in PWMC the models are *projected* on a subset of the variables (the priority variables) and a weight is given only to each resulting projected model. For example, if $F = (a \vee b) \wedge (b \vee c \vee d)$, then a model counter will sum over the 11 models of F. On the other hand, if the priority variables are $\{a, c\}$, then a projected model counter sums over 4 projected models (as setting $b = \top$ always makes F true).

In this work, we study the challenge of efficient and simple probabilistic inference using PWMC in more detail. We show that the aforementioned probabilistic inference tasks can be modeled as PWMC tasks over a simpler form of CNF formulas: CNF formulas of *Horn* clauses with a specific weighting scheme. We will call this task the task of *projected probabilistic Horn model counting* (PPHMC). We will argue that the resulting weighted CNF models are simple and polynomial in size given the original probabilistic models.

Subsequently, we will introduce a new solver, the Schlandals solver, which takes full advantage of the probabilistic weighting scheme and the fact that all clauses in our formulas are Horn clauses. The main intuition behind our solver is that it exploits the well-known fact that the SAT problem over conjunctions of Horn clauses can be solved in linear time. Using this observation, it is able to efficiently find assignments to the non-priority variables that greatly reduce the number of constrained clauses in the input formula (like $b = \top$ in the example above). At the same time, it is still able to exploit optimizations found in other DPLL-style model counters, such as component caching, unit propagation and branching strategies, to run with a bounded use of memory.

An experimental evaluation of our solver shows that it is competitive with state-of-the-art existing tools. We compare our solver with the best performing solvers of the 2022 projected (weighted) model counting competition on two probabilistic inference tasks: inference in Bayesian Networks and reliability estimation in PNs. Our experiments show that our solver is able to solve most of the instances for the BN task, beating other tools when using simple forms of CNF models, and getting similar performance when compared to optimized complex models. On PNs, our solver even outperforms the state-of-the-art, without any optimization of our model. These results open up future possibilities for the use of PPHMC.

2 Related Work

This work is concerned with the task of *Projected Weighted Model Counting* (PWMC). In the classical *Model Counting* problem, one is interested in finding the number of models of a Boolean formula F, the assignments that makes the formula true. In this work we focus on the most common setting, in which F is a formula in conjunctive normal form (CNF). In *Projected Model Counting*, the goal is to count the assignments to a subset of the variables (the *priority variables*) such that there exists an assignment to the other variables that makes F true. If the set of priority variables contains all variables of F, the projected and unprojected problems are the same. In the rest of this work, we thus assume that the subset of priority variable is not empty. In the weighted version of these problems, each valid assignment is weighted and the goal is to return the sum of the weights of the models.

More formally, let F be a Boolean formula over a set of variables $\mathcal{V}$ with $\mathcal{P} \subseteq \mathcal{V}$ the

priority variables and $\mathcal{D} = \mathcal{V} \setminus \mathcal{P}$ the non-priority variables. In traditional projected weighted model counting each priority variable is given two weights, $w(v)$ and $w(\neg v)$, one for each polarity. We denote by S_X an assignment to the variables in $X \subseteq \mathcal{V}$ and $S_X[x]$ the assigned value of $x \in X$. If $X, Y \subset \mathcal{V}$ are two disjoint sets of variables, then $S_X \cup S_Y$ is defined as the conjunction of the assignments. We define

$$\mathcal{S}_F = \{S_\mathcal{P} \mid \exists S_\mathcal{D} : F[S_\mathcal{P} \cup S_\mathcal{D}] = \top\}$$

as the set of assignments to the priority variables that can be extended with an assignment to the non-priority variables and satisfy the formula. The goal of the traditional PWMC task is then to find

$$\sum_{S \in \mathcal{S}_F} \left(\prod_{v \in \mathcal{V} | S[v] = \top} w(v) \times \prod_{v \in \mathcal{V} | S[v] = \bot} w(\neg v) \right).$$

Note that when weighted model counters are used to model probabilistic inference tasks, the weights of assignments are set such that this sum corresponds to a probability. If for every variable $p \in \mathcal{P}$ it holds that $w(p) + w(\neg p) = 1$ the resulting sum will always be ≤ 1. Unfortunately, the models for some problems require that $w(p) + w(\neg p) > 1$, and hence solvers cannot assume that $w(p) + w(\neg p) = 1$.

Various solvers exist to solve the PWMC problem and its unweighted version. In [22], the authors present `Ganak` is a model counter build upon `sharpSAT`, a DPLL-style model counter with component caching [26]. `Ganak` uses probabilistic component caching while ensuring guarantees on the validity of the returned count. Furthermore, the authors propose to use information about the cache in the branching heuristics and show that this is beneficial to model counters. `Ganak` can also be used for projected model counting, by first branching on the priority variables, but it does not use other specialized techniques. Further modifications of `Ganak` have been proposed and in particular it has been shown that integrating tree decomposition in the branching heuristic can have a positive impact [15]. This has led to the development of `SharpSAT-TD` [15], a (weighted) model counter, and `GPMC` [25], which is also able to solve the PWMC problem.

The `projMC` solver [17] uses another approach to solve the PWMC problem[1]. To compute the count over a formula F they first compute a disjunctive decomposition from a model of F. They then use the pairwise incompatible parts of the decomposition to simplify F, and they recursively solve the new sub-problems.

The aforementioned solvers use a strategy in which a limited amount of memory is used. This is also the focus of our work. An alternative strategy is to use a model compilation in combination with dynamic programming. Recently, Dudek et al. proposed the two-phased `ProCount` solver [11]. In the first phase, the input formula is transformed into a graded project-join tree (by a *planner*). Then, in the second phase, an executor (based on algebraic decision diagrams) is used to compute the count, using a dynamic programming approach.

3 Problem Definition

First, let us note that, since this work focuses on probabilistic problems, the weights on the priority variables are used to model probabilities. Hence, we refer to those as *probabilistic*

[1] Although the original paper only describes the unweighted problem, a parameter in the solver enables the retrieval of the count as a float, taking into account the weights

variables while the set of non-priority variables are called *deterministic variables*. In the rest of this paper, we denote with $P(E)$ the probability that an event E occurs. In this work, we introduce a combination of two novelties in how to model problems using PWMC. First, a constraint on the clauses in F is imposed: we only allow clauses to be *Horn* clauses.

▶ **Definition 1** (Horn clause). *A Horne clause C is a formula of the form*

$$v_1 \wedge \cdots \wedge v_n \Rightarrow v_t$$

where $I = v_1 \wedge \cdots \wedge v_n$ is called the implicant *of the clause and $h = v_t$ is* the head *of the clause. Here $v_i \in \mathcal{V}$ is a variable, and v_t is either a variable in $\mathcal{V}$ or $\bot$. If $n = 0$ (the implicant is empty) then the left-hand side reduces to $\top$.*

It can be observed that when a Horn clause is written as an implication, as above, all the literals in the clause have the same (positive) polarity. Hence, in order to simplify our discussion and notation, we only talk about variables, and not literals. A Horn clause C_i can be identified uniquely by its implicant I_i and its head h_i. In the rest of this paper, we will use the notation C_i and (I_i, h_i) interchangeably. For simplicity of notation, we also denote by $v \in I_i$ the fact that $v \in \mathcal{V}$ is a variable of the implicant of C_i.

Horn clauses have been well studied in the literature. An important result is that the SAT problem over a CNF formula of Horn clauses can be solved in linear time [10]; given that the SAT problem in its general form is NP hard, this is a significant simplification.

Secondly, we add the notion of *distributions over the probabilistic variables*. We assume that each probabilistic variable $p \in \mathcal{P}$ belongs to exactly one partition $P_i \subseteq \mathcal{P}$ of the probabilistic variables. We define a *distribution* over each such partition, in the following simple manner: we require that one weight is specified for each probabilistic variable and require that $\sum_{p \in P_i} w(p) = 1$ for all variables in the partition. Subsequently, we calculate the weight of an assignment S to the probabilistic variables as follows. The weight of a partition, given the assignment S, is defined as follows:

$$w_{P_i}(S) = \begin{cases} w(p) & \text{if there is exactly one } p \in P_i \text{ for which } S[p] = \top \\ 0 & \text{otherwise,} \end{cases}$$

or, in other words, if exactly one variable in the partition is set to $\top$, the weight of that variable is given to that partition; otherwise, the assignment is invalid. Implicitly, we allow only one variable within a partition to be true at the same time. The weight of the assignment S is the product of the partition weights given S and thus the solution of the PPHMC problem is given by

$$\sum_{S \in \mathcal{S}_F} \prod_i w_{P_i}(S),$$

where we assume there is at least one partition of probabilistic variables.

3.1 Models

As our earlier discussion makes clear, in this work we study a simpler modeling language in which non-Horn clauses are not allowed, and we combine this with a different approach to weighting. In this section, we will show that even though we apply the aforementioned restrictions, a number of different problems can still be modeled in our language.

▶ **Example 2** (Bayesian Networks). A *Bayesian Network* (BN) is a probabilistic model that can be represented by a directed acyclic graph. Random variables are represented by nodes

and conditional dependency relationships by edges. Figure 1(a) shows an example of a BN with four nodes. For simplicity, we illustrate the encoding in our language for a network with nodes that have two values, but this generalizes for nodes with more than two values.

In this network, both B and C depend on A, while D depends on B and C. In addition to the network structure, *conditional probability tables* (CPTs) give, for each node, the probability of its values conditioned by its parent's value. These are called the *parameters* of the network and for conciseness we write $P(x \mid \mathbf{u})$ for the parameter corresponding to the probability that a node X takes value x given that its parents take values $\mathbf{u} = u_1, \ldots, u_n$.

Various CNF encodings have been proposed for Bayesian networks [2, 4, 5, 7]. Even though the author of these works use Horn clauses, the un-projected nature of their target solvers imposes additional, non-Horn, clauses.

Let us present our encoding for BN by first defining the logical variables. For every value x of node X in the network, we define one deterministic variable v^x. For the BN in Figure 1, we have the following variables: $v^{a_0}, v^{a_1}, v^{b_0}, v^{b_1}, v^{c_0}, v^{c_1}, v^{d_0}, v^{d_1}$. Moreover for each parameter $P(x \mid \mathbf{u})$ we define a corresponding probabilistic variable $p^x_{\mathbf{u}}$. For the CPT of node B, we introduce four such variables: $p^{b_0}_{a_0}, p^{b_1}_{a_0}, p^{b_0}_{a_1}, p^{b_1}_{a_1}$.

In our approach, we have to define the distributions over the probabilistic variables. The CPTs of the network give a natural partition of the probabilistic variables. That is, for a node X in the network, we define one partition for each line of its CPT. Hence, for node B there are two partitions: $P_{B_1} = \{p^{b_0}_{a_0}, p^{b_1}_{a_0}\}$ and $P_{B_2} = \{p^{b_0}_{a_1}, p^{b_1}_{a_1}\}$. Next, we define the weight on the variables. Contrary to previous encodings, only weights on probabilistic variables are needed, and we use as weight the parameter they represent: $w(p^x_{\mathbf{u}}) = P(x \mid \mathbf{u})$. For example, we have that $w(p^{b_0}_{a_0}) = 0.6$ and $w(p^{b_1}_{a_0}) = 0.4$.

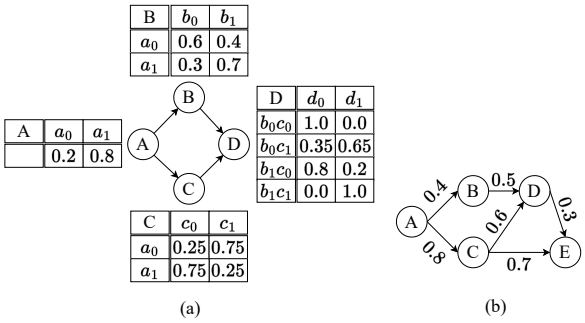
Finally, we define the clauses: for each parameter $P(x \mid \mathbf{u})$ with $\mathbf{u} = u_1, \ldots, u_n$, we introduce one clause $v^{u_1} \wedge \ldots \wedge v^{u_n} \wedge p^x_{\mathbf{u}} \Rightarrow v^x$. The clauses for the BN in Figure 1 are shown below.

$$
\begin{array}{rcl}
p^{a_0} \Rightarrow v^{a_0} & v^{a_0} \wedge p^{c_0}_{a_0} \Rightarrow v^{c_0} & v^{b_1} \wedge v^{c_0} \wedge p^{d_0}_{b_1 c_0} \Rightarrow v^{d_0} \\
p^{a_1} \Rightarrow v^{a_1} & v^{a_0} \wedge p^{c_1}_{a_0} \Rightarrow v^{c_1} & v^{b_1} \wedge v^{c_0} \wedge p^{d_1}_{b_1 c_0} \Rightarrow v^{d_1} \\
v^{a_0} \wedge p^{b_0}_{a_0} \Rightarrow v^{b_0} & v^{a_1} \wedge p^{c_0}_{a_1} \Rightarrow v^{c_0} & v^{b_0} \wedge v^{c_1} \wedge p^{d_0}_{b_0 c_1} \Rightarrow v^{d_0} \\
v^{a_0} \wedge p^{b_1}_{a_0} \Rightarrow v^{b_1} & v^{a_1} \wedge p^{c_1}_{a_1} \Rightarrow v^{c_1} & v^{b_0} \wedge v^{c_1} \wedge p^{d_1}_{b_0 c_1} \Rightarrow v^{d_1} \\
v^{a_1} \wedge p^{b_0}_{a_1} \Rightarrow v^{b_0} & v^{b_0} \wedge v^{c_0} \wedge p^{d_0}_{b_0 c_0} \Rightarrow v^{d_0} & v^{b_1} \wedge v^{c_1} \wedge p^{d_0}_{b_1 c_1} \Rightarrow v^{d_0} \\
v^{a_1} \wedge p^{b_1}_{a_1} \Rightarrow v^{b_1} & v^{b_0} \wedge v^{c_0} \wedge p^{d_1}_{b_0 c_0} \Rightarrow v^{d_1} & v^{b_1} \wedge v^{c_1} \wedge p^{d_1}_{b_1 c_1} \Rightarrow v^{d_1}
\end{array}
$$

The clause $v^{a_0} \wedge p^{b_0}_{a_0} \Rightarrow v^{b_0}$, for instance, represents that if A has value a_0 and we pick the variable $p^{b_0}_{a_0}$ from the distribution $\{p^{b_0}_{a_0}, p^{b_1}_{a_0}\}$, B will have value b_0; we believe this is a natural and simple representation that directly reflects the BN.

Note that we can satisfy all clauses by setting the v variables to true. A common inference problem on Bayesian Networks is that of calculating a probability $P(X = x)$. We can solve this problem by adding a clause $v^{x'} \Rightarrow \bot$ for each value x' of X such that $x' \neq x$. Effectively this removes from the sum those assignments in which $X \neq x$.

This encoding differs in a number of ways from the encodings used in WMC [2, 6]. The general idea is similar: for rows of the CPT, clauses are created; probabilistic variables receive weights that represent entries in the CPTs. Compared to earlier encodings, we do not generate clauses to impose that the indicators variables (v^x) are mutually exclusive for a node X. Our weighting scheme takes care of this. Furthermore, the earlier encodings have the parameter variables as the head of the implications in the CNFs, while in our encoding



(a) (b)

Figure 1 a) An example of Bayesian Network with four binary variables. In this network B and C depends on A and D depends on B and C. The probability tables are given next to the nodes. b) An example of probabilistic network for reliability problems. The numbers labeled on the edges are their probability of being present.

they are in the implicants; while both representations are equivalent, we believe that in our representation the structure of the BN is more closely reflected in the clauses.

▶ **Example 3** (Reliability in Networks)**.** Reliability in network (RN) problems study the connectivity of nodes in probabilistic graphs. In such graphs, as shown in Figure 1(b), each edge has a probability of being present. In this work, we consider the computation of the probability that two nodes are connected.

More formally, let $G = (V, E)$ be a probabilistic graph and $f_w : E \mapsto [0, 1]$ a weighting function that assign to each edge a probability of being present or not. We denote s the source node, t the target node and R_t^s ($\bar{R}_t^s$) the fact that t is (not) reachable from s. The goal is then to compute $P(R_t^s)$.

The encoding of this problem in our language is similar to that in [12], in which the authors propose to compute $P(\bar{R}_t^s)$ and then use the fact that $P(R_t^s) = 1 - P(\bar{R}_t^s)$ to answer the initial query.

Let us first define the logical variables. For each node $X \in V$, we introduce one deterministic variable v_X. For each edge $e \in E$ from u to v, we introduce two probabilistic variables p_{uv} (the edge is present) and $\bar{p}_{uv}$ (the edge is not present). The weighting scheme of the probabilistic variables uses the weighting function of the edges: $w(p_{uv}) = f_w(e)$ and $w(\bar{p}_{uv}) = 1 - f_w(e)$. A distribution is defined for each edge, containing these two variables: $P_e = \{p_{uv}, \bar{p}_{uv}\}$. Since the probabilistic variables for each edge are in their own distribution, and each distribution contains no other variables, it is easy to see that an assignment to the probabilistic variables corresponds to a possible instance of the graph.

We will use the deterministic variables v_X to represent whether in the instance implied by the probabilistic variables, X is reachable from s. The clauses, that we define hereafter, must ensure that an assignment is a model only if $v_t = \bot$ given the choices for the edges.

To ensure that, the clauses use the transitive nature of the connectivity. That is, for each edge e from u to v, a clause $v_u \wedge p_{uv} \Rightarrow v_v$ is created. This can be interpreted as that if u is reachable from s and e is present, then v is also reachable from s. We impose that s is reachable from s by adding the clause $\top \Rightarrow s$ and that t is not reachable by adding the clause $v_t \Rightarrow \bot$. The clauses for the query $P(\bar{R}_E^A)$ for the graph in Figure 1 are shown below.

$$v_A \wedge p_{AB} \Rightarrow v_B \qquad v_B \wedge p_{BD} \Rightarrow v_D \qquad v_C \wedge p_{CE} \Rightarrow v_E \qquad v_E \Rightarrow \bot$$
$$v_A \wedge p_{AC} \Rightarrow v_C \qquad v_C \wedge p_{CD} \Rightarrow v_D \qquad v_D \wedge p_{DE} \Rightarrow v_E \qquad \top \Rightarrow v_A$$

▶ **Example 4** (Probabilistic programming)**.** ProbLog is a probabilistic programming language that extends Prolog with probabilistic predicates [8]. It can be used to represent both the aforementioned inference problems. While a full discussion of this language is beyond the scope of this paper, we wish to illustrate how PPHMC can be used in ProbLog. Consider the following example ProbLog program, taken from the ProbLog website[2]:

```
0.4 :: heads.
0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).
win :- heads, col(_,red).
win :- col(1,C), col(2,C).
query(win).
```

For such a logic program, ProbLog performs *grounding*, creating a propositional version of the program. This propositional version can be represented as follows in our language:

$$\begin{array}{rlll} p_{heads}, p_{col(1,red)} \Rightarrow & v_{win} & p_{col(1,blue)}, p_{col(2,blue)} & \Rightarrow v_{win} \\ p_{heads}, p_{col(2,red)} \Rightarrow & v_{win} & v_{win} & \Rightarrow \bot \\ p_{col(1,red)}, p_{col(2,red)} \Rightarrow & v_{win} & & \end{array}$$

Here we have these partitions: $w(p_{heads}) = 0.4$, $w(\bar{p}_{heads}) = 0.6$; $w(p_{col(1,red)}) = 0.3$, $w(p_{col(1,blue)}) = 0.7$; $w(p_{col(2,red)}) = 0.2$, $w(p_{col(2,green)}) = 0.3$, $w(p_{col(2,blue)}) = 0.5$. The probability of the query is obtained by PPHMC on this ground version. Given that our previous example showed how reliability problems can be modeled in polynomial space, while the model for this problem is exponential in the grounded ProbLog model, we hypothesize that this is possible for all grounded programs without cycle breaking.

4 Algorithm

In the section we present the main algorithms of our solver. At the core, the problem is solved by a backtracking search over the possible assignments for the distributions. When a value is assigned to a variable, we call a new propagator designed specifically for the structure of the clauses. We first present the general algorithm for a search-based solver for the PWMC problem and then the specific propagation algorithm as well as some branching heuristics.

4.1 General Approach

The main algorithm is shown in Algorithm 1. In essence, it is similar to other DPLL-based solvers. The computation of the count for a non-empty formula F starts by looking into a cache (line 7) to determine if the formula has already been counted. If not, then it chooses an unfixed variable (lines 8, 10) and assigns it a value. A residual formula is computed after calling a propagation algorithm (line 11) which is then divided into independent components (line 14). The components are then solved independently (line 17) and their count is stored in the cache (line 18). In order to bound the memory consumption of search based solvers, the cache has a limited number of entries. As cache cleaning techniques are not the focus of this work, we simply fully clear the cache when the limit is reached.

There are a few differences between our solver and other model counters that need to be pointed out. First, let us note that the variables in a distribution are mutually exclusive.

[2] https://dtai.cs.kuleuven.be/problog/tutorial/advanced/00_inference.html

Algorithm 1 General PPHMC search algorithm.

```
1  Function PWMC(F, 𝒫)
       input  : A Horn-CNF formula F with probabilistic variables 𝒫
       output : The projected (on 𝒫) weighted model count of F
2      C ← newCache()
3      return PWMCr(F, 𝒫, C)
4
5  Function PWMCr(F, 𝒫, C)
       input  : F, 𝒫 same as PWMC()
       input  : C the cache of previously found counts
       output : Same as PWMC()
6      if 𝒫 = ∅ then return 1
7      if F ∈ C then return C[F]
8      P ← a distribution of 𝒫 such that ∃v ∈ P | v is not fixed
9      count ← 0
10     foreach v ∈ P | v is not fixed do
           /* Assign v = ⊤ and call the propagation algorithm. Returns the
              residual formula F' and the unconstrained probability of F
              given F'.                                                  */
11         (F', U_{F|F'}) ← Propagate(F, 𝒫, v)
12         proba ← U_{F|F'}
13         if F' is not UNSAT then
14             Components ← all connected components of F'
15             foreach Comp ∈ Components do
16                 𝒫' ← 𝒫 reduced to the variables in Comp
17                 proba_Comp ← PWMCr (Comp, 𝒫', C)
18                 C[Comp] ← proba_Comp
19                 proba ← proba * proba_Comp
20             end
21             count ← count + proba
22         end
23     end
24     return count
```

Indeed, if a distribution does not have exactly one variable set to $\top$ in an assignment, then its weight is 0 and $\prod_i w_{P_i}(S) = 0$, which does not contribute to the count.

A consequence of this is that, unlike classical DPLL-based counters, the branching decision is made on distributions and not variables. Indeed, since the distributions partition of the probabilistic variables (i.e., the variables on which the number of weighted models must be counted), assigning one variable to $\top$ in each distribution means that all other probabilistic variables are set to $\bot$. Moreover, when fixing a variable v in the selected distribution, only the case of fixing to $\top$ needs to be explored. The case $v = \bot$ is explored when the other variables in the distribution are selected for branching. Notice that when there is no distribution left in F, then 1 is returned as the remaining formula is SAT (line 6). Indeed, there are no unit clauses (they are removed during the propagation) and all that remains in F are Horn clauses with deterministic variables. By setting all remaining variables to $\bot$ we obtain a model of this formula, as the implicant of all clauses evaluates to $\bot$. This would not be

possible if F was not solely composed of Horn clauses. In this case, the SAT problem needs to be solved on F, which is NP-hard.

The computation of independent components is also slightly different. In traditional model counters, when the input formula F can be decomposed into multiple sub-formulas that do not share any variable, each sub-formula is solved independently and the count of F is the product of the sub-formulas' count. However, since the probabilistic variables are linked in partitions, we must add as condition that two independent sub-formulas cannot share any variable in the same distribution.

Finally, we also devised a new propagation algorithm, explained in the next section. It returns (line 11) a residual formula F' as well as what we call the *unconstrained probability of F given F'*: $U_{F|F'}$. This probability accounts for the distributions of F that are unconstrained in F'. Indeed, if for a distribution $P_i = \{p_i^1, \dots, p_i^m\}$ occurring in F, there are no clause in F' that contains any of the $p_i^k \in P_i$, P_i will never be selected by the branching heuristic. Moreover, branching on it will not impact F'. Hence, the probability obtained by branching of P_i can be precomputed and is given by $\sum_{p \in P_i | p \text{ is not fixed}} w(p)$. Note that this sum may not be 1 if propagation fixed one of the p_i^k variables to $\bot$ earlier, and $|P_i| > 2$. More generally, for l such distributions $P_{u_1}, \dots, P_{u_l}$ we have $U_F = \prod_{i=1}^{l} \sum_{p \in P_{u_i} | p \text{ is not fixed}} w(p)$.

4.2 Propagation

In this section, we describe the propagation algorithm used by our solver, summarized in Algorithm 2. In brief, we first apply the classical *Boolean Unit Propagation* (BUP), in which the links between probabilistic variables in partitions are also enforced, until a fixed point is reached. Then we remove from the remaining formula the clauses that do not constrain the count anymore. First, let us detail what the BUP does when a variable v is fixed in F:

- If $v = \top$ then
  - If v is a probabilistic variable in a distribution P, apply the BUP on all $v' \in P, v' \neq v$ with $v' = \bot$.
  - Every clause $C = (I, h)$ such that $h = v$ is removed from F. Indeed, for every assignment on its remaining variables, it evaluates to $\top$.
  - For every clause $C = (I, h)$ such that $v \in I$, replace I by $I' = I \setminus \{v\}$.
- If $v = \bot$ then
  - If v is a probabilistic variable in a distribution P and only one variable v' remains unfixed in P, apply BUP with $v' = \top$.
  - Every clause $C = (I, h)$ such that $v \in I$ is removed from F for the same reason as above.
  - The head of every clause $C = (I, h)$ such that $h = v$ is replaced by $\bot$.

There are two cases in which, after a call to the BUP, a variable is forced to take a value, and the BUP algorithm needs to be called again. First, when the last variable of an implicant is removed from it, then the head of the clause must be $\top$. Secondly, when a clause (I, h) has its head set to $\bot$ and there is only one variable left in I, then this variable must be set to $\bot$. In Algorithm 2, the call to BUP at line 2 is executed until no such clauses can be found.

However, in the context of *projected* model counting, a key insight for our work is that further propagation can be done that is not entailed by BUP. The intuition is that the deterministic variables can be used to remove additional clauses from F. For example, if our formula includes a clause $\neg a \vee \neg b \vee \neg c$ with a, b being probabilistic variables and c a deterministic variable that does not appear in any other clauses, then setting c to $\bot$ makes the clause evaluate to $\top$, regardless of the choice for a and b. Since we are only interested in

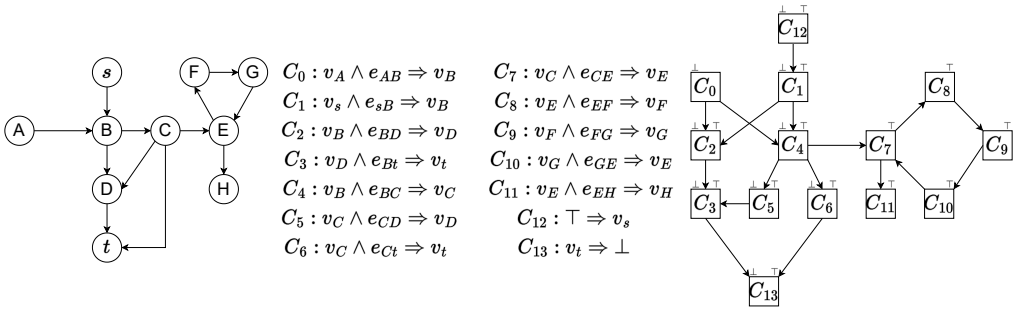


Figure 2 On the left: an instance of a reliability problem on probabilistic networks. In the center: the associated clauses for the query $P(\bar{R}_t^s)$. On the right: the graph of clauses implication: there is one node per clause and a link between C_i and C_j if $h_i \in I_j$.

finding an assignment to the deterministic variables, this assignment does not impact the final count.

Let us show how this works on the clauses in Figure 2, taken from a reliability query in a probabilistic networks, in which we want to compute $P(\bar{R}_t^s)$, the probability that s and t are not connected in the graph on the left. Applying BUP gives $v_s = \top$ and $v_t = \bot$, which removes these variables from C_1, C_3 and C_6. No further propagation can be done with BUP. However, when looking at the graph on the left, it is clear that only the nodes B, C and D impact the connectivity between s and t, but this is not detected by BPU. For instance, let us look at nodes A and the associated clause C_0, which contains the distribution $\{e_{AB}, \bar{e}_{AB}\}$. It can be seen that for both choices for the edge e_{AB}, setting $v_A = \bot$ (a deterministic variable) reduces the clause C_0 to $\top$. Since there are no clauses that have v_A in their head, setting v_A to $\bot$ has no impact on the other variables in F. Moreover, since it is a deterministic variable, it does not impact the projected weighted model count. A similar reasoning can be made for H and C_{11} by setting $v_H = \top$, since v_h does not appear in any implicant in F.

The intuition behind our propagation is the following. In order for an assignment to not be a model of the input formula, it must generate a clause $\top \Rightarrow \bot$. Some clauses cannot contribute to such a contradiction, by setting a deterministic variable to $\bot$ in its implicant or its head to $\top$. We formalize this intuition next.

First, let us define in which case we cannot know if a clause will have its head set to $\bot$ or its implicant set to $\top$.

▶ **Definition 5** ($\{\top, \bot\}$-reachability). *A clause $C_i = (I_i, h_i) \in F$ is $\bot$-reachable if one of the two following conditions is met:*

1. *C_i is of the form $I_i \Rightarrow \bot$ or $I_i \Rightarrow p$ with $p \in \mathcal{P}$*
2. *There exists a clause $C_j = (I_j, h_j) \in F$ such that C_j is $\bot$-reachable and $h_i \in I_j$.*

Similarly, C_i is $\top$-reachable if one of the two following conditions is met:

1. *There exists no deterministic variables in I_i*
2. *There exists a clause $C_j = (I_j, h_j) \in F$ such that C_j is $\top$-reachable and $h_j \in I_i$.*

We say that a clause is *constrained* if it is $\bot$-reachable and $\top$-reachable. The $\{\bot, \top\}$-reachability can be seen for the clauses in Figure 2 on the right, on the implication graph of the clauses. The implication graph of a set of Horn clauses is a graph $G = (V, E)$ such that there is one node per clause and an edge from a clause $C_i = (I_i, h_i)$ to $C_j = (I_j, h_j)$ if $h_i \in I_j$. If a clause is $\top$-reachable ($\bot$-reachable), so are all its descendants (ancestors) in the implication graph.

Algorithm 2 Propagation algorithm of Schlandals.

Function `Propagate`$(F, \mathcal{P}, v)$
  input : A boolean formula F with probabilistic variables $\mathcal{P}$
  input : A variable v set to $\top$
  output : The residual formula F' and a propagation probability p_{prog}
  /* Call the BUP procedure with the initial assignment of $\top$ to v until fix point is reached */
  $F' \leftarrow$ `BUP`$(F, v, \top)$
  foreach $C = (I, h) \in F'$ **do**
    if $h = \bot$ **or** $h \in \mathcal{P}$ **then** `SetFReachable`(F', C)
    if $\nexists v \in I \mid v \notin \mathcal{P}$ **then** `SetTReachable`(F', C)
  end
  foreach $C \in F'$ **do**
    if C *is not* $\top$*-reachable* **or** C *is not* $\bot$*-reachable* **then**
      $F' \leftarrow F' - C$
    end
  end
  $U_{F|F'} \leftarrow 1$
  foreach *distribution* P *such that* $P \in F \wedge P \notin F'$ **do**
    $U_F \leftarrow U_F * \sum_{p \in P | p \text{ is not fixed}} w(p)$
  end
  return $(F', U_{F|F'})$

We have the following theorem, which states that if a clause is not constrained, then it can be removed safely (without impacting the count) from F.

▶ **Theorem 6.** *Let $F = C_1 \wedge \ldots \wedge C_n$ be a formula with n Horn clauses over the variables $\mathcal{V}$, $\mathcal{P} \subseteq \mathcal{V}$ the set of probabilistic variables and $\mathcal{D} = \mathcal{V} \setminus \mathcal{P}$ the set of deterministic variables. Let $C_{u_1}, \ldots, C_{u_k}$ be k unconstrained clauses of F with $C_{u_i} = (I_{u_i}, h_{u_i})$.*

There exists a subset of k deterministic variables $X = \{x_1, \ldots x_k\} \subseteq \mathcal{D}$ with $x_i \in I_{u_i} \cup \{h_{u_i}\}$ and an assignment S_X on X such that

$$\mathcal{S}_F = \mathcal{S}_{F[S_X]}$$

where S_F denotes the set of models of F, projected on $\mathcal{P}$, and $F[S_X]$ the formula obtained by applying the BUP algorithm on F with the assignment S_X.

We now prove this theorem and give the procedure to find the assignment on the deterministic variables.

Proof. Let $G = (V, E)$ be the graph of the implications of F and C_i an unconstrained clause in F. We prove that an assignment can be found for one of the deterministic variables of C_i such that it does not impact the count of F.

First, let us assume that C_i is not $\bot$-reachable. We denote $G^d_{C_i} = (V^d_{C_i}, E^d_{C_i})$ the sub-graph of G that contains C_i and all its descendants. By definition there is no clause $C_j \in V^d_{C_i}$ that is $\bot$-reachable, otherwise C_i would be $\bot$-reachable. Hence, all clauses in $V^d_{C_i}$ are unconstrained. Let $X^d = \cup_{C_j \in V^d_{C_i}} \{h_j\}$ be the set of (deterministic) heads in $G^d_{C_i}$. We define the assignment S_{X^d} such that $S_{X^d}[x] = \top$, $\forall x \in X^d$. This removes all the clauses in $V^d_{C_i}$ from F without impacting the values of the other variables in the clauses.

Next, if C_i is $\bot$-reachable but unconstrained, then it is not $\top$-reachable. Let $G^p_{C_i} = (V^p_{C_i}, E^p_{C_i})$ be the sub-graph of G that contains C_i and all its parents. Since C_i is $\bot$-reachable, so is every clause in $V^p_{C_i}$. Thus we have that every clause $C_j \in V^p_{C_i}$ has at least one deterministic node $d_j \in I_j$, otherwise they would be $\top$-reachable. Let $X^p = \cup_{C_j \in V^p_{C_i}} \{d_j\}$ be the set of such nodes. We set S_{X^p} such that $S_{X^p}[d_j] = \bot$ for all clause $d_j \in X^p$ which removes all clauses in $V^p_{C_i}$ from F without constraining the other variables. ◀

Algorithm 3 Procedure to mark the clauses of a formula F as $\bot$-reachable.

```
1 Function SetFReachable(F, C)
      input : A boolean formula F, a clause C = (I, h) of F
2     if C is not marked as ⊥-reachable then
3         Mark C as ⊥-reachable
4         foreach C' = (I', h') ∈ F | h' ∈ I do SetFReachable(F, C')
5     end
```

This additional propagation is shown in lines 3-11 of Algorithm 2. After the application of BUP until a fix point is reached, the remaining clauses are iterated over. If a clause is of the form $C = (I, \bot)$ or has a head with an unfixed probabilistic variable, then the procedure `SetFReachable` is called, for which the code is shown in Algorithm 3. This algorithm basically traverses the implication graph of F, starting from C and marks every clause it encounters as $\bot$-reachable. Notice that it does not mark multiple times the same clause. Hence, the cost of marking all the $\bot$-reachable clauses is $\mathcal{O}(n)$ with n the number of clauses in F'. A similar procedure is defined for the $\top$-reachability, but we do not include it for conciseness. After marking the clauses (lines 3-5), every unconstrained clause is removed from the formula obtained after BUP (lines 7-11). Finally, the algorithm computes the unconstrained probability $U_{F|F'}$ (lines 12-15).

It can be noted that in some cases our propagation is similar to *Pure Literal Elimination* (PLE) on the deterministic variables. For instance, in Figure 2, v_A never appears in the head of any clause. Hence, it can be set as $\bot$, which is also the value found by our procedure. However, PLE cannot detect that the nodes E, F and G are not useful for the query. It should also be noted that although our propagation implies that PLE is performed, the procedure only reasons about the clauses, which makes it more efficient.

4.3 Branching Heuristic

At line 8 of Algorithm 1, a distribution is selected within the set of distributions for which no element is set to $\top$. In this section we explain the heuristics implemented in our solver to choose the distribution. We must note that our solver is not a conflict-driven clause learning (CDCL) solver, making it impossible to use the heuristics of such solvers [19, 22]. Instead, we implemented some easy to understand, fast to evaluate, but effective heuristics based on the implication graph of a formula F. We provide three heuristics related to the degree of a clause in the implication graph. These heuristics select a clause with i) the lowest in-degree ii) the lowest out-degree and iii) the maximum degree, and then a distribution in this clause. Each time they are called, they re-evaluate the score of each clause in the current sub-formula.

5 Results

In order to evaluate the effectiveness of our approach, we compare our solver[3] with `GPMC` [25] and `ProjMC` [17] on Bayesian networks and reliability queries in probabilistic networks. We chose these solvers as they were the best performing on the Projected Model Counting and Projected Weighted Model Counting tracks of the 2022 model counting competition. We did not include the `proCount` solver [11] as it is a knowledge compilation based solver. `Ganak` [22] also was not used as we found it was the worst performing on the projected tasks (as also observed in the 2022 competition). For the reliability queries, we also ran `ApproxMC` [3, 23, 24] as it is known to perform well on this problem, although it should be stressed `ApproxMC` solves an *approximate* model counting problem, while we solve an exact model counting problem. For each of the two problems we first present the methodology used to generate the instances and then present the results[4]. For each instance, a timeout of 600 seconds and a memory limit of 15 GB were set.

5.1 Baysian Networks

The Bayesian networks come from the bnlearn R package [21] repository and range from small networks (fewer than 20 nodes) to large (up to hundreds of nodes). We selected all the networks in the repository for which at least one solver did not time out. For each of these networks, the queries are done on the leaves of the networks without any evidence. Hence, for a leaf L of the network that takes values $l_1, \ldots, l_n$, we create n instances for the queries $P(L = l_1), \ldots, P(L = l_n)$.

The goal of this experiment is to answer the following questions: i) how do the evaluated PWMC solvers perform using our simple encoding? ii) How does our encoding perform compared to other state-of-the-art, but more complex encodings for BNs, designed for weighted model counters? For i) we use a CNF formula in other PWMC solvers similar to our model, where we add additional clauses to enforce the distribution constraints that we have in our weighting scheme. For ii), we follow the nomenclature in [6], and we compare our encoding with ENC1 [7], ENC3 [4], and ENC4 [5]. We also add the encoding recently presented in [2], which we denote ENC4linp. Briefly, ENC1 is the simplest encoding: there is one indicator variable λ^x for each value x of node X and one parameter variable $\theta^x_{\mathbf{u}}$ per parameter $P(x \mid \mathbf{u})$ of the network. For the clauses, the indicator variables of the same node are mutually exclusive; a clause $\lambda^{u_1} \wedge \ldots \wedge \lambda^{u_n} \wedge \lambda^x \Leftrightarrow \theta^x_{\mathbf{u}}$ is created for each parameter $P(x \mid \mathbf{u})$. The main addition of ENC3 is that, for a given CPT in the network, the same variable $\theta^x_{\mathbf{u}}$ is reused for all the entries having the same probability. In ENC4, The CPTs are simplified using a modified version of Quine/McCluskey algorithm for finding prime implicants, resulting in a better decomposition of the input CNF. Finally, in ENC4linp, the authors propose two novelties for encoding BNs in a CNF: the use of log-encoding for the elementary assignment of a variable and one parameter variable per CPT is kept implicit. For these encodings, `GPMC` and `projMC` are run in WMC mode (which is the `d4` [16] model counter for the latter).

Figure 3 shows the percentage of instances solved within the time limit.

First, it can be seen that if we use our model in other PWMC solvers, their performance is much worse; these solvers solve a few instances.

[3] Available at `https://github.com/aia-uclouvain/schlandals`
[4] The instances can be found here `https://github.com/AlexandreDubray/pwmc_benchmarks`

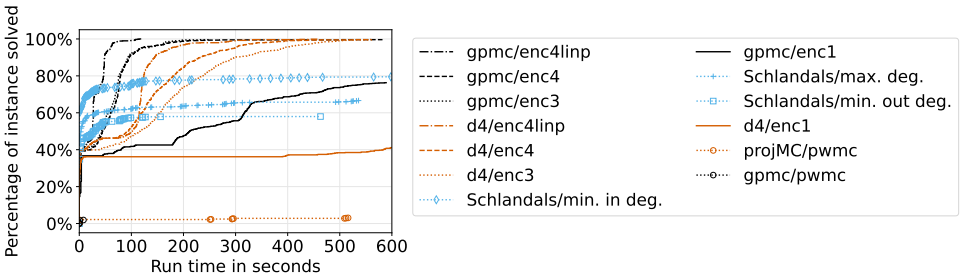


Figure 3 Percentage of instances solved in 600 seconds for the Bayesian Network data sets.

Compared to the ENC1 model used in traditional WMC, which is arguably the model for traditional WMC most similar to our model, we can still observe that our approach performs significantly better, solving 80% of the instances for the minimum in degree heuristic. We believe it is highly encouraging that our approach works so well for such a simple model.

When evaluating the more complex, optimized models ENC3, ENC4 and ENC4linp developed for WMC, we can see that these have increasingly better results. This confirms the benefit of the various improvements made, over the years, to the encodings. Overall, the `GPMC` solver is the best performing of all, even when using only the ENC3 encoding. The biggest gap between the ENC encodings is between ENC1 and ENC3. Indeed, using the same parameter variables for multiple parameters of a CPT gives a huge drop in the number of variables, especially for the hardest instances, which contain large CPTs. The benefits of the Quine/McCluskey reduction are clearly visible for the `d4` solver.

The nature of our modeling language is such that the optimizations of ENC3, ENC4 and ENC4linp cannot be directly applied to our models. Unlike the ENC encodings, we require that one variable per distribution is set to $\top$. However, we hypothesize that similar optimizations can also be developed for our solver in future work.

5.2 Connectivity in Probabilistic Networks

For this problem, the data sets used come from two sources. First, we used the power grid network of Europe and USA as extracted by the GridKit tool [18, 28]. The extracted graphs represent the electric power system in these geographical areas. In order to have various sizes of instances, both networks are divided by country (Europe) or by state (USA). Then for each subnetwork, five random pairs of nodes are selected and the probability that they are connected is computed. Notice that since the edges in the power-grid network have no orientation (the graph is undirected), we transformed it into a directed graph by replacing each edge between u and v by one edge from u to v and one edge from v to u. We assume that each edge has a probability of 0.125 to be down, as done in [12].

Secondly, we used (oriented) graphs representing water distribution systems from the WNTR Python package [14]. We considered as sources (sinks) nodes having no parents (children). Then, we compute $P(\bar{R}_t^s)$ between each source-target pair (s, t) such that there exists a path between s and t. As for the power grid network data set, we assume a fix probability of 0.125 that each link is down.

Unlike for Bayesian Networks, our encoding and the one proposed in [12] are almost identical and can be reused for all the evaluated solvers. The only difference is that we do not create two variables per edge for the encoding passed to `GPMC` and `projMC`. Figure 4 shows the percentage of instances solved in the given amount of time for both of these data sets.

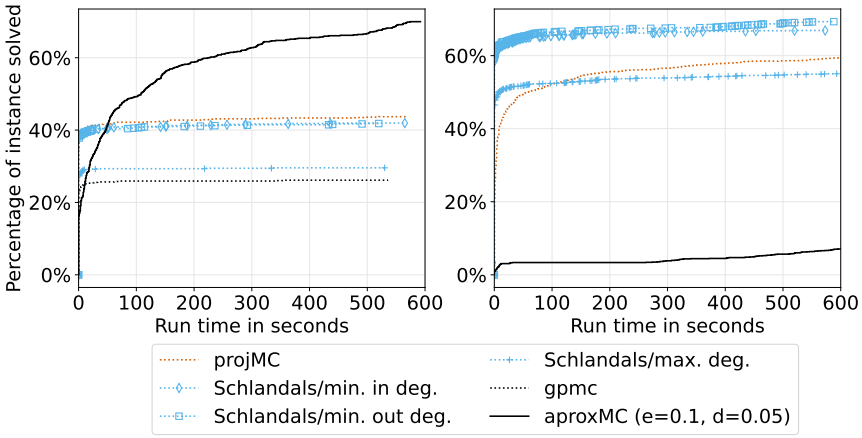


Figure 4 Percentage of instances solved in 600 seconds for the power-grid networks (left) and the water networks (right).

First, let us note that `GPMC` performs the least well on both data sets. While it is able to solve up to 25% of the instances for the power grid network, it is unable to solve the largest instances of the water networks. For the latter data set, it quickly reaches the memory limit (15 GB) and stagnates until the time-out. On the other hand, it can be seen that `projMC` and `Schlandals` solve roughly the same number of instances on the power grid networks, while `Schlandals` solves 10% more instances on the water networks. On these types of data sets, the min-in degree and min-out degree heuristics work best for our solver. It can be seen that for the water networks, `Schlandals` outperforms `projMC`.

Interestingly, on the power grid data sets, our solver is still able to match the performance of `projMC` even though our propagation is not used to its full capacity. Indeed, since the original graphs are not directed, they were transformed by adding two directed edges for each undirected edge. As a result, either all clauses in a component are constrained, or no clause is constrained. Indeed, by doubling the edges in the original graph, the edges in the implication graph of the resulting CNF are also doubled. Hence, every clause which is a descendant of a $\bot$-reachable clause is also one of its ancestors and a similar argument can be made for $\top$-reachable clauses. Hence, our propagation algorithm has less pruning power. On the other hand, for the water network data set, our solver uses the full strength of our propagation and is more efficient than `projMC`.

Let us briefly comment on the performance of `approxMC` [12]. Again, it should be noted that `approxMC` is an approximate model counter that provides provides $(\delta - \epsilon)$-guarantees and does not consider weights on the literals. Hence, the problem it solves is quite different from the other solvers compared in this work: if the count returned by `approxMC` is C, and the exact count $C^\star$, it ensures that $\frac{C^\star}{1+\epsilon} < C < C^\star(1+\epsilon)$ with a confidence of $1-\delta$ ($\delta, \epsilon \in [0,1]$). In Figure 4 the results are shown for $\epsilon = 0.1$ and $\delta = 0.05$. This is an example of configuration in which the solver is quite confident in its solution, and hence the results are more comparable to that of the other solvers. It can be seen that it performs very well on the power grid network instances, but poorly on the water supply network ones. Using $\epsilon = 0.8$ and $\delta = 0.2$, as reported in [12], the solver is able to solve all instances on both data sets, but it is much less confident in its solution. Overall the performance of `approxMC` heavily depends on the acceptable margin of error.

6 Conclusion

Weighted model counters have become an essential tool in probabilistic reasoning, but the CNF models have grown more and more complex. A step towards simplicity for some problems has been taken by the introduction of projected weighted model counting. In this work, we propose the Projected Probabilistic Horn model counting (PPHMC) problem, in which only Horn clauses are allowed, making the modeling language simpler. We have shown that with an appropriate weighting scheme it is possible to model important probabilistic problems as PPHMC problems. In particular, we provide an encoding for Bayesian networks, probabilistic networks and probabilistic logic programs. We also introduced a new tool, the Schlandals solver, specifically designed for our language. Our experiments show that our solver is competitive with state-of-the-art solvers and opens the path to further work on PPHMC.

As we have seen for the Bayesian Networks, the encoding can have a great impact on the performance. An interesting line of work would be to investigate how the optimizations developed during the past twenty years, for the encoding of Bayesian Networks into a logical formula, can be applied with our weighting schema, and how our solver can be integrated in probabilistic logic programming systems. On the other hand, a lot of work has been done to make model counters efficient. In the future, integrating such techniques (probabilistic caching, tree decomposition, symmetry breaking, advanced branching heuristics, etc.) with the specificity of PPHMC can also increase the performance of our solver. Finally, this paper has been focused on PWMC within bounded memory. However, the core of our solver can be reused in a knowledge compiler or an approximate solver.

References

1 Rehan Abdul Aziz, Geoffrey Chu, Christian Muise, and Peter Stuckey. #∃ SAT: Projected model counting. In *International Conference on Theory and Applications of Satisfiability Testing 2015*. Springer, 2015.

2 Anicet Bart, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. An improved CNF encoding scheme for probabilistic inference. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 2016.

3 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic sat calls. In *IJCAI*, 2016.

4 Mark Chavira and Adnan Darwiche. Compiling Bayesian networks with local structure. In *IJCAI*, volume 5, 2005.

5 Mark Chavira and Adnan Darwiche. Encoding CNFs to empower component analysis. In *Theory and Applications of Satisfiability Testing-SAT 2006: 9th International Conference, Seattle, WA, USA, August 12-15, 2006. Proceedings 9*. Springer, 2006.

6 Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7), 2008.

7 Adnan Darwiche. A logical approach to factoring belief networks. *KR*, 2, 2002.

8 Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7. Hyderabad, 2007.

9 Paulius Dilkas and Vaishak Belle. Weighted model counting with conditional weights for Bayesian networks. In *Uncertainty in Artificial Intelligence*. PMLR, 2021.

10 William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3), 1984.

11 Jeffrey M. Dudek, Vu H. N. Phan, and Moshe Y. Vardi. ProCount: Weighted Projected Model Counting with Graded Project-Join Trees. In Chu-Min Li and Felip Manyà, editors, *Theory*

and Applications of Satisfiability Testing – SAT 2021, Lecture Notes in Computer Science, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-80223-3_11.

12 Leonardo Duenas-Osorio, Kuldeep Meel, Roger Paredes, and Moshe Vardi. Counting-based reliability estimation for power-transmission grids. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

13 Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15(3), 2015.

14 Katherine A. Klise, Michael Bynum, Dylan Moriarty, and Regan Murray. A software framework for assessing the resilience of drinking water systems to disasters with an example earthquake case study. *Environmental modelling & software*, 95, 2017.

15 Tuukka Korhonen and Matti Järvisalo. Integrating tree decompositions into decision heuristics of propositional model counters. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

16 Jean-Marie Lagniez and Pierre Marquis. An Improved Decision-DNNF Compiler. In *IJCAI*, volume 17, 2017.

17 Jean-Marie Lagniez and Pierre Marquis. A recursive algorithm for projected model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2019.

18 Wided Medjroubi, Ulf Philipp Müller, Malte Scharf, Carsten Matke, and David Kleinhans. Open data in power grid modelling: New approaches towards transparent grid models. *Energy Reports*, 3, 2017.

19 Tian Sang, Paul Beame, and Henry Kautz. Heuristics for fast exact model counting. In *Theory and Applications of Satisfiability Testing: 8th International Conference, SAT 2005, St Andrews, UK, June 19-23, 2005. Proceedings 8*. Springer, 2005.

20 Tian Sang, Paul Beame, and Henry Kautz. Solving Bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 1. AAAI Press, 2005.

21 Marco Scutari. Learning Bayesian networks with the bnlearn R package. *arXiv preprint arXiv:0908.3817*, 2009. arXiv:0908.3817.

22 Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A Scalable Probabilistic Exact Model Counter. In *IJCAI*, volume 19, 2019.

23 Mate Soos, Stephan Gocht, and Kuldeep S. Meel. Tinted, detached, and lazy cnf-xor solving and its applications to counting and sampling. In *International Conference on Computer Aided Verification*, 2020.

24 Mate Soos and Kuldeep S. Meel. Bird: engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In *AAAI*, 2019.

25 Ryosuke Suzuki, Kenji Hashimoto, and Masahiko Sakai. Improvement of projected model-counting solver with component decomposition using SAT solving in components. Technical report, JSAI Technical Report, SIG-FPAI-506-07, 2017.

26 Marc Thurley. sharpSAT-counting models with advanced component caching and implicit BCP. *SAT*, 4121, 2006.

27 Jonas Vlasselaer, Angelika Kimmig, Anton Dries, Wannes Meert, and Luc De Raedt. Knowledge compilation and weighted model counting for inference in probabilistic logic programs. In *Proceedings of the First Workshop on Beyond NP*. AAAI Press, 2016.

28 Bart Wiegmans. Gridkit: European And North-American Extracts, March 2016. doi:10.5281/ZENODO.47317.