Anytime Weighted Model Counting with Approximation Guarantees for Probabilistic Inference

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 (WMC) plays a central role in probabilistic reasoning. Given that this problem is $\#P-$hard, harder instances can generally only be addressed using approximate techniques based on sampling, which provide statistical convergence guarantees: the longer a sampling process runs, the more accurate the WMC is likely to be. In this work, we propose a deterministic search-based approach that can also be stopped at any time and provides hard lower- and upper-bound guarantees on the true WMC. This approach uses a value heuristic that guides exploration first towards models with a high weight and leverages *Limited Discrepancy Search* to make the bounds converge faster. The validity, scalability, and convergence of our approach are tested and compared with state-of-the-art baseline methods on the problem of computing marginal probabilities in Bayesian networks and reliability estimation in probabilistic graphs.

2012 ACM Subject Classification Mathematics of computing → Probabilistic inference problems; Theory of computation → Probabilistic computation; Mathematics of computing → Approximation

Keywords and phrases Projected Weighted Model Counting, Limited Discrepancy Search, Approximate Method, Probabilistic Inference

Digital Object Identifier 10.4230/LIPIcs.CP.2024.10

Supplementary Material

Software (Source Code): `https://github.com/aia-uclouvain/schlandals` [8]
archived at `swh:1:dir:3ffca0d07dbd88cffbbdfda6e3f0ae09a9e77ac0`

Dataset: `https://github.com/AlexandreDubray/bn-benchmarks`
archived at `swh:1:dir:e4644057447aa2f4b9579b41744a6dede462da6a`

Dataset: `https://github.com/AlexandreDubray/probabilistic-graph-benchmarks`
archived at `swh:1:dir:fc1586f609a54f32b09e4d71eda2eb6457f00d20`

1 Introduction

Model counting, the problem of counting the number of satisfying assignments of a propositional formula, is a canonical $\#P-$hard problem with many applications. In particular, (projected) weighted model counting ((P)WMC) has been extensively used for probabilistic reasoning problems, such as computing the marginals in Bayesian Networks [4, 5, 20, 9] or reliability estimation in probabilistic graphs [10, 9]. While model counters are becoming increasingly efficient, computing an exact probability is not always possible for large probabilistic models. Methods computing an approximation of the probability offer better

scalability, but their guarantees are often statistical [3, 1, 12, 11, 13, 18]. Such methods are often based on sampling, meaning they can be stopped anytime. Moreover, the more the sampling process runs, the more accurate the estimated probability will likely be. For example, model counters providing so-called $(\varepsilon, \delta)-$guarantees return an approximation whose relative error is bounded by a parameter ε with probability $1-\delta$ [3, 1]. Some methods offer weaker guarantees; they ensure that the probability of returning a wrong lower bound on the probability decreases with the number of iterations [13, 18].

This work proposes a new search-based method for solving PWMC problems. It can be stopped at any time and returns hard bounds on the true count. We show that a DPLL-style model counter can maintain a lower and an upper bound on the true count at each search tree node. This modified version of the classical DPLL-style algorithm can be stopped at any time during its execution, providing deterministic guarantees on the true count. Then, we observe that in the specific context of *weighted* model counting, *Limited Discrepancy Search* [14] (LDS) offers advantages compared to classical depth-first-search methods. Based on this observation, we propose a simple value-selection heuristic that favors the most likely model and use it in an LDS-based search. Our final contribution is to link the lower and upper bound on the true count to the well-known ε-guarantee. We show that bounds can be used to compute an approximate probability that satisfies an ε requirement and to determine the quality of an approximate model count returned by our algorithm when interrupted by a time-out. Using this theoretical result and the LDS-based search, we provide the first anytime method that can return, during its execution, approximate weighted counts with minimal ε-guarantees. We implemented this method, named `Schlandals-LDS-`ε, in `Schlandals` [9], a recently proposed projected weighted model counter specialized for probabilistic inference. We compare our method against state-of-the-art solvers on two major probabilistic reasoning tasks: computing marginal probabilities in Bayesian networks and reliability estimation in probabilistic graphs. Our experiments show that `Schlandals-LDS-`ε performs better on both tasks when looking at the number of solved instances and bound convergence, validating the effectiveness of our bound calculation and LDS-based weighted model counting.

The rest of this paper is organized as follows. We review the technical background for model counting and `Schlandals` in Section 2, followed by a review of the related work in Section 3. In Section 4, we present `Schlandals-LDS-`ε We first introduce how to compute bounds in Section 4.1 and then the search based on Limited Discrepancy Search in Section 4.2. We show the link between the bounds and ε-guarantees in Section 4.3. We compare our work with existing methods and evaluate the convergence of our bounds in Section 5 before concluding in Section 6.

2 Technical Background

Let F be a boolean formula over a set of variables $\mathcal{V}$. An *interpretation* I for a formula F is a complete assignment to all variables in F, and a *partial* interpretation is an interpretation on a subset of all the variables in F. We use $I = \{a = \top, b = \bot\}$ to denote an interpretation in which a is true ($\top$) and b is false ($\bot$). By $F[I]$, we denote the evaluation of F under I, using the standard way of interpreting logical formulas. If $F[I] = \top$, I is a *model* of F. Let $\mathcal{M}_F = \{I \in \{0,1\}^{|\mathcal{V}|} \mid F[I] = \top\}$ be the set of models of F. The goal of a model counter is to compute $|\mathcal{M}_F|$, the number of models of F. In weighted model counting, each variable $v \in \mathcal{V}$ has two weights $\omega(v^+)$ and $\omega(v^-)$; the weight of an interpretation is defined as $\omega(I) = \prod_{v \in I | v=\top} \omega(v^+) \prod_{v \in I | v=\bot} \omega(v^-)$. A weighted model counter computes $\sum_{I \in \mathcal{M}_F} \omega(I)$, the weighted sum of F's models. Let $\mathcal{P} \subseteq \mathcal{V}$ be a subset of the variables

and $\pi_{\mathcal{P}} : \{0,1\}^{|\mathcal{V}|} \mapsto \{0,1\}^{|\mathcal{P}|}$ the function that keeps, from an assignment to all variables of F, only the assignment to the variables in $\mathcal{P}$. A projected model counter computes the number of projected models, defined by $|\pi_{\mathcal{P}}(\mathcal{M}_F)| = |\{\pi(I) \mid I \in \mathcal{M}_F\}|$. For example, let $F = (a \vee \neg b) \wedge (b \vee c \vee d) \wedge (\neg a \vee d)$. This formula has 7 models, but if restricted to $\mathcal{P} = \{a, b\}$, then it only has 3 models. The interpretations $I_1 = \{a = \top, b = \top, c = \top, d = \top\}$ and $I_2 = \{a = \top, b = \top, c = \bot, d = \top\}$ are both models of F but, when restricted on the variables $\{a, b\}$, they are both equal: $\pi_{\mathcal{P}}(I_1) = \pi_{\mathcal{P}}(I_2) = \{a = \top, b = \top\}$. A projected weighted model counter computes $PWMC(F) = \sum_{I \in \pi_{\mathcal{P}}(\mathcal{M}_F)} \omega(I)$. Notice that in such a case, a weight only needs to be defined for an interpretation projected on the variables $\mathcal{P}$.

An approximate model counter returns an ε-approximation ($\varepsilon > 0$) if, given a true count $c^\star$, it returns a count $\hat{c}$ and the following inequality holds:

$$\frac{c^\star}{1+\varepsilon} \leq \hat{c} \leq c^\star(1+\varepsilon), \tag{1}$$

that is, the relative error of the approximation is bounded by a factor $1 + \varepsilon$, and we say that such model counters provide ε-guarantees. Approximate model counters providing (ε, δ)-guarantees return an approximation that respects Equation (1) with a probability $1 - \delta$.

`Schlandals` is a search-based projected weighted model counter specialized for probabilistic inference problems [9]. Its input language has some specific features. It requires, in particular, that the projected variables are partitioned into discrete probability distributions $D_1, \dots, D_n$.

▶ **Example 1** (Example of an input for Schlandals)**.** Below is an example of a formula F in Schlandals, where all variables v are in $\mathcal{P}$:

$$\begin{array}{lll} d_2^1 \wedge d_2^2 \Rightarrow d_1^1 & d_1^1 \wedge d_2^1 \Rightarrow d_4^1 & d_1^1 \wedge d_3^2 \wedge d_3^3 \Rightarrow d_5^2 \\ d_4^1 \Rightarrow d_2^2 & d_1^1 \wedge d_3^1 \Rightarrow d_5^1 & d_1^1 \wedge d_5^1 \Rightarrow d_3^2 \end{array}$$

with distributions $D_1 = \{d_1^1, d_1^2\}, D_2 = \{d_2^1, d_2^2\}, D_3 = \{d_3^1, d_3^2, d_3^3\}, D_4 = \{d_4^1, d_4^2\}$, and $D_5 = \{d_5^1, d_5^2\}$.

We denote by $dom_F(D_i)$ the domain of a distribution D_i (i.e., the set of probabilistic variables in its partition). For each variable $v \in dom_F(D_i)$, *one* weight $P(v)$ needs to be specified, in such a manner that $\sum_{v \in dom_F(D_i)} P(v) = 1$. When determining which interpretations are models in `Schlandals`, an implicit constraint is added: if an interpretation I does not fix *exactly* one variable $v \in dom_F(D_i)$ to $\top$ in each distribution D_i, then we have $F[I] = \bot$.

▶ **Example 2.** Continuing our example, an interpretation I setting $d_1^1 = \top$ and $d_1^2 = \top$ is not a model as two variables of the same distribution are set to $\top$. On the contrary, $I = \{d_1^2, d_2^2, d_3^1, d_4^1, d_5^1\}$ is a model of the formula. The weight of this model is obtained by calculating $P(d_1^2) \times P(d_2^2) \times P(d_3^1) \times P(d_4^1) \times P(d_5^1)$, reflecting the probabilities of the choices made for each of the distributions in the interpretation. For conciseness, we describe a partial interpretation I over the weighted variables by the choices (variables set to $\top$) in the distributions.

The importance of the `Schlandals` language is that it allows several probabilistic inference problems to be modeled in a concise manner, among which inference problems in Bayesian Networks and probabilistic graphs; here, the ability to perform *projected* model counting is critical for efficient encoding of inference problems on probabilistic graphs [27, 10]. The fact that weighted variables are part of *distributions* ensures that weighted model counts always correspond to probabilities.

Algorithm 1 PWMC as done in `Schlandals` [9].

```
1  Function PWMCr(F)
       input  : A boolean formula F with distributions D_F
       output : P[F] the weighted model count of F
2      if |D_F| = 0 then return 1  // F is SAT, return 1.  See [9]
3      if F in cache with value p then return p
4      P[F] ← 0
5      D_i ← choose_distribution()                    // For heuristics, see [9]
6      foreach v ∈ dom_F(D_i) do
7          if TIMEOUT then break
8          if F[v] = ⊥ then continue // Applies propagation
           // prop(F[v]) are the choices, for some distributions, forced by
              propagation
9          p ← ∏_{v'∈prop(F[v])} P(v')
10         C ← components of F[v]                      // Independent components
11         foreach F_c ∈ C do p^c ← PWMCr(F_c)
12         P[F] += p × ∏_c p^c
13     end
14     Adds F ↦ p in cache and return p
15 return PWMCr(F)
```

The `Schlandals` language, as presented in [9], also requires that all clauses are Horn. This allows `Schlandals` to solve satisfaction problems involving only non-weighted variables in polynomial time by means of propagation. While this restriction may seem limiting, it was shown that many problems can be modeled even under this restriction; moreover, the ideas presented below can also be extended to solvers for formulas without this restriction.

The `Schlandals` solver implements a variant of component caching DPLL search, presented in Algorithm 1. Given how a model's weight is defined, `Schlandals` does not branch like classical model counters. It starts from a formula F and selects a distribution not yet assigned (line 5). The variable selection heuristic is replaced by a *distribution* selection heuristic, and the value selection heuristic selects which variable in the distribution must be set to true. Then, for each value of its domain (lines 6-13), it sets it to true, applies propagation, and gets the residual formula $F[v]$ (line 8). Here `Schlandals` performs traditional forms of propagation, such as unit propagation, but also a specific form of propagation for Horn clauses. If a formula remains after propagation, it is decomposed into independent components (line 9) that are solved independently (line 11). The counts of the independent components are multiplied with each other and added to the count of the formula (line 12), multiplied by the probability of the distributions assigned during propagation (computed at line 9).

3 Related Work

Most other approximate model counters are sampling-based; these model counters provide statistical guarantees at best. *Hashing-based* approximate model counters use special classes of hash functions to sample partitions of the search space. They count the models in these small parts and estimate the overall count from these parts. Usually, such solvers provide $(\varepsilon, \delta)-$guarantees: they return an approximation whose relative (to the true count)

error is bounded by a factor ε, as specified by Equation (1), with a probability of $1 - \delta$. `ApproxMC` [3, 24, 23] is one such solver, designed for unweighted (projected) model counting. Based on the same idea, `WeightMC` [1] targets weighted model counting and sampling.

Another class of methods is that of the bounding counters. `SampleCount` [13] uses a sampler to estimate which variables, after being assigned, divide the solution space in half. When the formula is small enough, it counts exactly its number of models and multiplies the result by a constant factor to obtain an estimated count. This procedure obtains a lower bound with probabilistic guarantees when repeated multiple times. `PartialKC` [18] is another sampling-based model counter based on partial compilation. Both of these approaches are designed for unweighted model counting. Note that such sampling-based model counters can usually be turned into anytime model counters by estimating model counts even before the predefined approximation guarantee is ensured. In particular, the partial representations produced by `PartialKC` can be used to compute lower and upper bounds on the true count. However, given that `PartialKC` works on unweighted formulas, their computations differ from ours. Moreover, `PartialKC`'s compilation process does not aim to fasten the bounds convergence.

Model counting problems are related to several other problems. One such problem is calculating the partition function, Z, of discrete stochastic graphical models, such as Markov Random Fields, Markov Networks, or Cost Function Networks. An algorithm, called Z^*_ε, for calculating an estimate of $\hat{Z}$ with $\frac{Z}{1+\varepsilon} \leq \hat{Z} \leq Z$ guarantees was proposed by Viricel et al. [26] and is implemented in the `Toulbar2` [21] system. Also Z^*_ε calculates upper and lower bounds on the final model count, pruning the search based on ε. However, there are several differences between our algorithm and this algorithm: 1) our algorithm is defined over a different form of model that includes projected variables; 2) our algorithm is any time and operates such that an upper-bound on the final model count can be calculated during the search; 3) our upper-bound calculation is optimized for the type of model we work with: it relies on the fact that only variables have weights, integrates the bound calculation with domain propagation, and can be integrated into any DPLL-style algorithm with little overhead; 4) it does not have the behavior stated by Viricel et al. [26] that a larger ε can lead to less pruning; 5) we combine our method with LDS. We will demonstrate the benefits of our approach experimentally.

Finally, weighted model counting has been used as a probabilistic inference mechanism in probabilistic programming languages such as `Problog` [7, 27]. In particular, anytime methods have been developed specifically for `Problog` [28], where a TP-compilation incrementally calculates model counts by combining SDD diagrams [6].

4 Anytime Projected Weighted Model Counting with Bounds

This section describes `Schlandals-LDS-`ε, our anytime search-based approach for computing bounds on $P[F]$. First, we describe how a simple modification of a DPLL search can compute bounds at each search tree node. Then, we present how Limited Discrepancy Search (LDS) can be applied to weighted model counting. Finally, we show the relationship between the computed bounds and ε-approximations, and we show that LDS-based search can be used to compute approximate weighted model counts with deterministic guarantees.

4.1 Computing Bounds During the Search

Our lower and upper bounds are based on calculating which interpretations are (not) models, accounting for how propagation reduces the domain of the distributions. Let F be the (sub-)formula being solved and $\mathcal{D}_F = \{D_1, \ldots, D_n\}$ the distributions in F. We denote by

$dom_F(D_i) = \{d_i^1, \ldots, d_i^{k_i}\}$ the domain of D_i. During the search propagation might have previously removed some variables from $dom_F(D_i)$; hence, $\sum_{v \in dom_F(D_i)} P(v) \leq 1$ generally for subformulas considered during the search.

Lower bound

Our lower bound is relatively simple and amounts to maintaining a sum for models seen till a certain moment. Without loss of generality, let us assume that the solver decides to branch on D_1. We denote by $F[v]$ the formula that remains from F after deciding that, for D_1, variable $v \in dom_F(D_i)$ is true, and after propagation has been performed. Let $prop(F[v])$ be the set of probabilistic variables that are set to true during this propagation. Essentially our algorithm calculates the probability $P[F]$ recursively by exploring all possibilities for D_1, using:

$$P[F] = \sum_{v \in dom_F(D_1)} \left(P[F[v]] \times \prod_{v' \in prop(F[v])} P(v') \right)$$

A lower bound is obtained by executing this sum over a subset of $dom_F(D_1)$; hence, we can incrementally maintain a lower bound while considering the possibilities in $dom_F(D_1)$.

Upper bound

Maintaining an upper bound is conceptually more complex, but we propose an upper bound that can be calculated with relatively little overhead in a DPLL-based solver. A first naïve upper bound, which we will improve afterwards, is $P(\mathcal{D}_F)$, defined as follows:

$$P(\mathcal{D}_F) = \prod_{D_i \in \mathcal{D}_F} \left(\sum_{v \in dom_F(D_i)} P(v) \right) \tag{2}$$

i.e., $P(\mathcal{D}_F)$ represents the maximum probability that F can obtain if all remaining interpretations are models. The intuition behind this formula is that an upper bound can be obtained by calculating a weighted sum over the interpretations that are in the cartesian product of the remaining domains of the remaining distributions. The product calculates this sum efficiently. As illustrated in the following example, this upper bound can be tightened during the search when the domains of the distributions are pruned by propagation.

▶ **Example 3.** Continuing Example 1, let us consider the case when, at the root, the solver decides to branch on D_4 and assigns $d_4^1 = \top$. In that case, the clause $d_4^1 \implies d_2^2$ becomes $\top \implies d_2^2$, forcing $d_2^2 = \top$. Since exactly one variable must be true in each distribution, we have that $d_2^1 = \bot$. This means that no interpretation containing $d_4^1 = \top$ and $d_2^1 = \top$ can be a model of F, and the upper bound for the formula F can be improved. Assuming that no other variable is set to $\bot$ during that propagation, the weight of the removed interpretations is equal to $P(d_4^1) \times P(d_2^1)$, and the upper bound for F can be decreased to $1.0 - P(d_4^1) \times P(d_2^1)$.

This improved upper bound calculation is formalized in Algorithm 2, which is an adaptation of Algorithm 1 to compute bounds on $P[F]$. Intuitively, this algorithm computes, for any search tree node with sub-formula F', a sum of probabilities of models of F', denoted p_{in}, and a sum of probabilities of interpretations that are not models, denoted p_{out}. From these two values, the lower and upper bounds can be inferred. The structure of Algorithm 2 is the same as Algorithm 1. The search stops when there are no more distributions in F (line 2) or

Algorithm 2 PWMC with Bounds Computaton.

```
1  Function PWMC-Bounds(F)
       input  : A boolean formula F with distributions D_F
       output : The probability of the considered models (p_in) and non-models (p_out)
2      if |D_F| = 0 then return (1, 0)
3      if F in cache with values (p'_in, p'_out) then return (p'_in, p'_out)
4      p_in ← 0; p_out ← 0
5      D_i ← choose_distribution()                          // For heuristics, see [9]
6      foreach v ∈ dom_F(D_i) do
7          if TIMEOUT then break
8          if F[v] = ⊥ then
9              p_out += P(v) × P(D_F\{D_i})                 // Uses Equation (2)
10         else
11             p ← ∏_{v'∈prop(F[v])} P(v')
12             p_out += P(v) × (P(D_F\{D_i}) − P(D_F[v]))   // Uses Equation (2)
13             C ← components of F[v]                       // Independent components
14             foreach F_c ∈ C do
15                 (p^c_in, p^c_out) ← PWMC-Bounds(F_c)
16             end
17             p_in += p × ∏_c p^c_in
18             p_out += p × (P(D_F[v]) − ∏_c(P(D_{F_c}) − p^c_out))   // Uses Equation (2)
19         end
20     end
21     Add F ↦ (p_in, p_out) in cache and return (p_in, p_out)
22 (p_in, p_out) ← PWMC-Bounds(F, ε)
23 return (p_in, 1 − p_out)
```

the formula is in the cache (line 3). Otherwise, it selects a distribution to branch on (line 5). Then, it iterates over its domain (lines 6-20), applies `Schlandals`' propagation (line 8), and recursively explores the independent components (lines 13-16). Moreover, the computation of p_{in}, the sum of F models' probability (line 17), is the same as the computation of $P[F]$ in Algorithm 1.

Hence, the main difference between Algorithm 2 and Algorithm 1 is the computation of p_{out}. When exploring sub-problems of F (i.e., branching for some value v of a distribution), $F[v]$, the formula obtained after applying `Schlandals`' propagation, might turn out to be unsatisfiable. Then, all interpretations containing $v = \top$ cannot be models of F, and p_{out} is increased (line 8). This rule also applies after propagation: propagation may remove some variables from a distribution's domain, making some interpretations non-models. The probability of the interpretations setting one of these variables to $\top$ can be added to p_{out} (line 10). This computation is based on the difference in the maximum probability of the distributions before ($P(\mathcal{D}_F \setminus \{D_i\})$) and after the propagation ($P(\mathcal{D}_{F[V]})$). Finally, the probabilities of the non-models of each independent component are combined and added to p_{out} (line 18). The intuition for this formula is that if one partial interpretation in one of the components evaluates to false, then the whole interpretation evaluates to false regardless of the partial interpretation in the other components. Hence, if U_c represents the random event that the c-th component evaluates to $\bot$ (with probability p^c_{out}), we wish to compute $P(\vee_{i=1}^{|C|} U_i) = P(\neg(\wedge_{i=1}^{|C|} \neg U_i))$; this gives the indicated formula as for each component the probability that it is satisfiable is given by $P(\mathcal{D}_{F_c}) - p^c_{out}$.

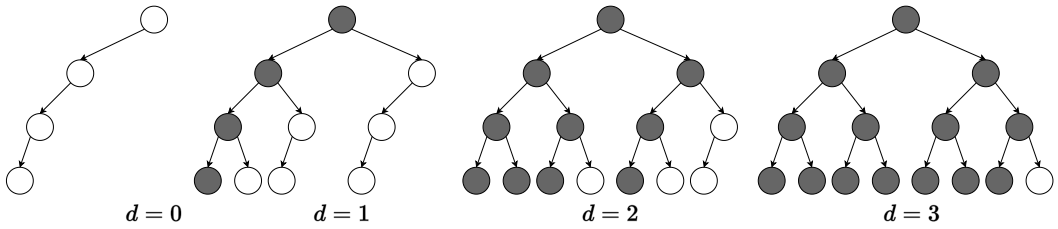

Figure 1 Example of iterative (from left to right) exploration of the search space with Limited Discrepancy Search. At each iteration, d is the current discrepancy, starting from 0. The white nodes represent newly explored nodes, while the grey ones represent nodes explored in previous iterations.

4.2 Limited Discrepancy Search for Weighted Model Counting

Limited Discrepancy Search [14] is a search procedure initially designed to solve constraint satisfaction problems in situations where it is reasonable to assume that a branching heuristic will often be correct. An essential intuition behind LDS is that if a branching heuristic is always correct, a solution can be found by following the leftmost branch of the search tree, that is, the branch preferred by the heuristic. If the number of times the heuristic is wrong is limited, and the lefthand branch of a node does not provide a solution, then the heuristic should be incorrect less often in the righthand subtree. Limited Discrepancy Search builds on this intuition; it iteratively explores the search space, deviating more and more from the heuristic, as illustrated in Figure 1. Initially, it follows exactly the branching heuristic, exploring the leftmost branch of the search tree. If no solution is found, it allows deviating from the heuristic at one node per branch: the discrepancy is 1. This process continues, incrementing the maximum number of discrepancies each time until a solution is found or the whole search space is explored. This intuition naturally extends to optimization problems: a good heuristic will guide the search toward a good solution and provide tight bounds. Hence, large parts of the search space can be pruned in subsequent iterations of the LDS. However, it is not obvious that counting problems benefit from such a search scheme; the whole search space must be explored to calculate the exact model count.

However, we argue that LDS offers advantages compared to Algorithm 2 in the specific setting of *anytime weighted* model counting, where the interest is to converge the bounds as rapidly as possible. Indeed, the interpretations of a formula F are unlikely to have uniform weights. If LDS' assumptions are true, then the most likely interpretations can be found using a small discrepancy, which is very fast for LDS, as it ignores large parts of the search space. On the other hand, methods based on depth-first-search, as presented in Algorithm 1 must explore the whole sub-tree, including unlikely interpretations, before switching to a more promising part of the search space; moreover, when decomposing a formula in components, DFS-based algorithms face the problem that they cannot raise the lower-bound before all components have been considered. Hence, for the hardest problems, DFS-based methods might even time out before exploring likely interpretations and considering all components. Limited Discrepancy Search has the potential not to suffer from these problems, if the branching heuristic is reasonable.

Algorithm 3 gives the modified procedure to perform an LDS-based DPLL search in `Schlandals`. The structure of the algorithm is the same as Algorithm 2: a recursive procedure (lines 1-18) explores the search space by selecting a distribution (line 5), exploring its values (lines 6-17) and recursively solving the sub-problems (line 12-14) while maintaining the p_{in} and p_{out} counts (lines 11,15-16). However, there are some key differences, which we will describe next.

Algorithm 3 PWMC with Bounds Computation and Limited Discrepancy Search.

```
1  Function Schlandals-LDS(F, d)
       input  : A boolean formula F with distributions 𝒟_F and a discrepancy budget d
       output : The probability of the considered models (p_in) and non-models (p_out)
2      if |𝒟_F| = 0 then return (1, 0)
3      if F in cache with values (p_in, p_out, d') with d' ≤ d or p_in + p_out = P(𝒟_F) then
           return (p_in, p_out)
4      p_in ← 0; p_out ← 0
5      D_i ← choose_distribution()                          // For heuristics, see [9]
6      for k ← 1 to min(d + 1, |dom_F(D_i)|) do
7          if TIMEOUT then break
8          v ← k−th values of dom_F(D_i)                    // Has the k−th highest weight
9          if F[v] = ⊥ then p_out += P(v) × P(𝒟_F\{D_i}); continue // Equation (2)
10         p ← ∏_{v'∈prop(F[v])} P(v')
11         p_out += P(v) × (P(𝒟_F\{D_i}) − P(𝒟_{F[v]}))
12         C ← components of F[v]
13         foreach F_c ∈ C do
               // Discrepancy not decrement for first child
               (p^c_in, p^c_out) ← Schlandals-LDS(F_c, d − (k − 1))
14         end
15         p_in += p × ∏_c p^c_in
16         p_out += p × (P(𝒟_F) − ∏_c(P(𝒟_{F_c}) − p^c_out))
17     end
18     Add (p_in, p_out, d) in cache and return (p_in, p_out)
19 d ← 0; lb ← 0; ub ← 0
20 while not TIMEOUT do
21     (p_in, p_out) ← Schlandals-LDS(F, d)
22     lb ← max(p_in, lb), ub ← min(1 − p_out, ub)
23     output (lb, ub)
24     d += 1
25 end
26 return (lb, ub)
```

First, when a formula is found in the cache (line 3), its result is not automatically returned. Indeed, if a node in the search tree is only partially explored when first encountered, its bounds are not tight, and $p_{in} + p_{out} \neq P(\mathcal{D}_F)$. Such a formula must still be explored, but only if the discrepancy budget is higher than the one stored in the cache; otherwise, no new interpretation can be found.

Then, when a distribution D_i is selected for branching, the iteration on its domain (line 6) differs from classical DPLL-search: it is limited by the discrepancy budget. Moreover, the effectiveness of LDS heavily depends on the value selection heuristic. We propose a simple heuristic that favors the most likely interpretation first. Let $dom_F(D_i) = \{v_1, \ldots, v_n\}$ be the domain of D_i. Let us assume that the domain is iterated from v_1 to v_n; given a discrepancy budget d, the values $v_1, \ldots, v_d$ are explored. We propose to order the values of the domain such that $P(v_1) \geq P(v_2) \geq \ldots \geq P(v_n)$. The rationale behind this heuristic is the following. The weight of an interpretation of F is computed as $\prod_{D_i \in \mathcal{D}_F} P(v_i^\top)$ with $v_i^\top$ being the variable set to $\top$ for distribution D_i. Hence, selecting the most likely variable first aims to favor the most likely interpretations.

Finally, the outer loop calling the recursive function (lines 20-25) is specific to LDS. The bounds are initialized (line 19), and while the time limit is not reached, an iteration of LDS is done (line 21). The bounds are updated (line 22), output to the user (line 23), and the discrepancy is incremented (line 24). Notice that the bounds are updated if they are better than the previously found bounds. Indeed, if the time limit is reached, the last iteration of LDS may explore only a small part of the search space, leading to worse bounds than in the previous iteration. When the whole search space has been explored or the time limit is reached, the bounds are returned to the user (line 26).

4.3 From Bounds to an Epsilon Guarantee

We have shown that it is possible to modify a DPLL-style search algorithm to compute a lower and upper bound on the true probability. We now show a relationship between these bounds and an ε-guarantee. Remember that a model counter computes an approximate probability $\hat{p}$ of p with ε-guarantee on its error if $\frac{p}{1+\varepsilon} \leq \hat{p} \leq p(1+\varepsilon)$.

▶ **Theorem 4.** *Let F be a formula with true probability $P[F]$, $\varepsilon \geq 0$ an error factor. If $P_l[F]$ is a lower bound on $P[F]$, $P_u[F]$ an upper bound on F, and $P_u[F] \leq P_l[F] \times (1+\varepsilon)^2$ then we have that*

$$\frac{P[F]}{1+\varepsilon} \leq \sqrt{P_l[F] \times P_u[F]} \leq P[F](1+\varepsilon).$$

Proof. Let us first prove the left part of the inequality. By definition, we have that $P[F] \leq P_u[F]$ and, by assumption, $\frac{P_u[F]}{(1+\varepsilon)^2} \leq P_l[F]$, hence

$$P[F]^2 \leq P_u[F]^2 \Leftrightarrow \frac{P[F]^2}{(1+\varepsilon)^2} \leq \frac{P_u[F]^2}{(1+\varepsilon)^2} = \frac{P_u[F]P_u[F]}{(1+\varepsilon)^2} \leq P_l[F] \times P_u[F]$$
$$\Leftrightarrow \sqrt{\frac{P[F]^2}{(1+\epsilon)^2}} = \frac{P[F]}{(1+\varepsilon)} \leq \sqrt{P_l[F] \times P_u[F]}$$

We prove the second inequality in a similar manner.

$$P[F] \geq P_l[F] \Leftrightarrow P[F]^2(1+\varepsilon)^2 \geq P_l[F]^2(1+\varepsilon)^2 = P_l[F]P_l[F](1+\varepsilon)^2 \geq P_l[F]P_u[F]$$
$$\Leftrightarrow \sqrt{P[F]^2(1+\varepsilon)^2} = P[F](1+\varepsilon) \geq \sqrt{P_l[F] \times P_u[F]}$$ ◀

This theorem tells us two things. First, if the bounds on a probability $P[F]$ are close enough, with respect to an allowed ε error factor, the search can be stopped before a time-out is reached and we can return a probability that respects a required ε-guarantee. Second, however, if a time-out is reached, we can also still estimate the quality of the model count at that moment, with respect to an $\varepsilon-$guarantee: given a lower bound $P_l[F]$ and an upper bound $P_u[F]$, Theorem (4) allows us to compute the *minimum required* ε that would stop the search at that moment. Algorithm 4 shows an adaptation of the outer loop of Algorithm 3 to perform LDS with an optional error factor ε. The algorithm runs until the condition of Theorem 4 is met or until the time limit is reached. At each iteration, it outputs an approximate probability with the ε error factor that would stop the search at that moment.

We believe that this setting can be useful for instance in experiments where a large number of probabilities need to be calculated; using our solver this can be done with a time out limit on each query, while at the end of the experiment statistics can be calculated on the approximation quality of the probabilities calculated in the experiment.

Algorithm 4 Approximate PWMC with LDS.

$d \leftarrow 0$; $lb \leftarrow 0$; $ub \leftarrow 0$
while not *TIMEOUT* **and not** $ub \leq lb(1+\varepsilon)^2$ **do**
  $(p_{in}, p_{out}) \leftarrow$ `Schlandals-LDS(`F, d`)`
  $lb \leftarrow \max(p_{in}, lb)$, $ub \leftarrow \min(1 - p_{out}, ub)$
  $\varepsilon_d \leftarrow \sqrt{\frac{ub}{lb}} - 1$
  output $(\sqrt{lb \times ub}, \varepsilon_d)$
  $d \mathrel{+}= 1$
end
return $(\sqrt{lb \times ub}, \sqrt{\frac{ub}{lb}} - 1)$

5 Experimental Results

In this section, we evaluate the effectiveness of our LDS-based[1] search method against state-of-the-art methods. All methods are evaluated on two axes: i) How many instances they can solve and ii) how fast they converge toward the true probability. Moreover, we evaluate the effectiveness of our value-selection heuristic.

We evaluate our methods against the following state-of-the-art methods. We compare against `PartialKC` [18], a recently proposed anytime *unweighted* model counter[2] and `ApproxMC` [3, 24, 23], which is a popular hashing-based model counter providing $(\varepsilon, \delta)-$guarantees. We omit `WeightMC` [1] in our experiments as the available code was not updated recently and did not scale on our instances. We also compare our LDS search against the TP-compilation algorithm of `Problog`[7, 28], which provides a lower and an upper bound on the probability in an anytime fashion. Unfortunately, the upper bound is not available to users, and there is no easy way to extract it[3]. The Toulbar2 [21] solver was also run with the algorithm presented in [26]. Finally, using Theorem 4, we evaluate an anytime version of Algorithm 2, which can be seen as the DFS alternative to our LDS-based search. To do so, we ran Algorithm 2 with increasingly high timeouts, storing for each timeout the returned bounds. In the rest of this section, we denote by `DFS` this approach, and by `LDS` the approach presented in Algorithm 4. To better evaluate these approaches, we also ran `D4` [15, 16], `GPMC` [25] and `ExactMC` [17] on our benchmarks. All methods ran with a timeout of 600 seconds and a memory limit of $15Gb$ when the option was available with the solver. Except for parameters related to approximations, all methods have been executed with their default parameters. We evaluated these methods on two problems: computing the marginals in Bayesian networks without evidence and computing reachability queries in probabilistic graphs.

The *Bayesian networks* have various sizes, ranging from a few parameters to tens of thousands, and originated from the bnlearn R package [22]. For each network, we use one query per value of the leaves: if a leaf node in the network has four values, we create four instances that compute the marginal probability of each value. Overall, there are 2749

[1] The source code is available at `https://github.com/aia-uclouvain/schlandals`.
[2] At the time of the writing of this work, there exists a parameter in `PartialKC` to perform weighted model counting. Unfortunately, the feature is not yet implemented
[3] We checked this with the maintainers of the Problog implementation.

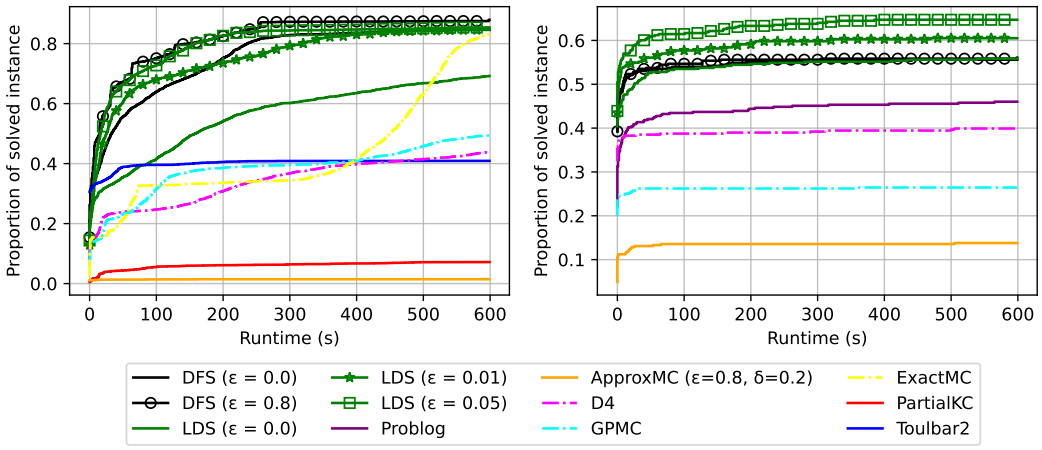

Figure 2 Proportion of solved instances over time, with a timeout of 600 seconds, for Bayesian networks (left) and reliability estimation (right) problems.

instances[4]. For `Schlandals`, we use the encoding presented in [9], and for `PartialKC` and `ApproxMC`, we use the encoding from [4]. Since these solvers work on unweighted formulas, we transformed the weighted instances into unweighted ones using the approach proposed in [2], which encodes the weight with additional variables and clauses. Then, the unweighted count can be divided by a normalization factor to obtain the weighted model count. A precision parameter can adjust the number of clauses and variables added at the cost of a less precise weighted model count. We chose a precision of 10 so that the result after normalization is similar to the true probability up to five decimals. However, using a lower precision did not change the results significantly. For `Toulbar2`, we used the UAI format with, as evidence, the pair node-value to query. Since `Problog` is not designed to work directly on CNF formulas, implementing the various optimizations for encoding Bayesian networks is impossible. Hence, we decided not to run it on these benchmarks as it would only scale to the easiest instances.

For the *reachability queries*, we took the graphs representing power grid networks in Europe and the USA extracted by the GridKit tool [19, 29]. For each of these graphs, sub-graphs were created by splitting the nodes and edges by country (for Europe) and state (for the USA). Following the description in [10], we assign each edge a probability of 0.125 of being down. Finally, five queries are created for each sub-graph by taking random pairs of nodes and computing their connection probability[5]. For `Schlandals`, we again used the encoding presented in [9], which is based on the ones presented in [10] and used for the other solvers. Unfortunately, `PartialKC` and `Toulbar2` are not projected model counters and cannot be launched on these instances. For `ApproxMC`, the instances were made unweighted using the same approach as for the Bayesian networks but with a precision of 3.

Evaluation of Solving Instances

First, we analyze how many instances each method can solve. Here, we consider an instance solved if it respects the approximation contract, if any, or finds the true probability. For example, an instance with a probability of 0.5 would be solved by a solver returning 0.55

[4] The scripts to generate the instances can be found at `https://github.com/AlexandreDubray/bn-benchmarks`

[5] The scripts to generate the instances can be found at `https://github.com/AlexandreDubray/probabilistic-graph-benchmarks`

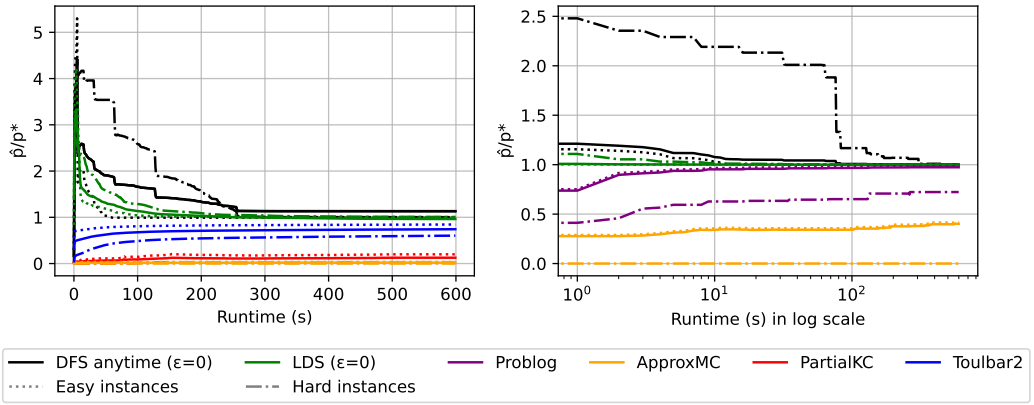

Figure 3 Average convergence of the estimated probability ($\hat{p}$) towards the true probability ($p^\star$) for Bayesian networks (left) and reliability estimation (right) problems. For `Schlandals`, the estimated probability is given by $\sqrt{lb \times ub}$; the other methods directly output an estimated probability. The continuous lines represent the average over all instances, the dotted lines are the average over the easy instances (can be solved by `Schlandals` in less than 60 seconds), and the dashed lines are the average over the hard instances.

when required to produce an ε-approximation with $\varepsilon = 0.2$. Figure 2 shows the proportion of solved instances within the time limit for Bayesian networks (left) and reachability queries (right). It can be seen that `Schlandals` performs the best regardless of the problem and the type of solver (exact, approximate, or anytime). The unweighted model counters are performing less well. Adding variables and clauses for the weights heavily impacts their performance, particularly for large probabilistic models with thousands of weights. In our experiments, on most instances, `Toulbar2` was not able to improve its performance when calculating an ε-approximation using the algorithm presented in [26]. Hence, this graph only includes the result for $\varepsilon = 0$.

It can be seen that when solving the problems exactly ($\varepsilon = 0$), the LDS version of `Schlandals` performs less well than its DFS version; this is to be expected as parts of the search space are traversed multiple times. However, when a small error (ε) is acceptable, then `LDS` performs roughly the same as `DFS` on Bayesian networks and surpasses it on reachability queries (with $\varepsilon = 0.01$, the DFS does not solve more instances within the time limit, so we did not include this curve in the graph). Even more striking, `LDS` can solve at least as many instances, in the same amount of time, as `DFS` and produce probabilities under much stronger guarantee requirements ($\varepsilon = 0.05$ instead of $\varepsilon = 0.8$). Although `LDS` has some overhead by exploring some parts of the search space, and it can be seen when the whole search space must be explored, when computing ε-approximation, it seems that it allows to converge much faster towards the true probability. We confirm this intuition in the next section by analyzing the convergence of the various methods.

Convergence of the Anytime Methods

Figure 3 shows the average convergence of the methods towards the true probability when computing it exactly (i.e., with $\varepsilon = 0$). For the two versions of `Schlandals`, we use Theorem 4 and the bounds to produce an approximation. We modified the outer loop of `ApproxMC` (see Algorithm 1 in [3]) to output the estimated probability at each iteration. While these intermediate solutions have weaker guarantees, considering `ApproxMC` as an anytime model

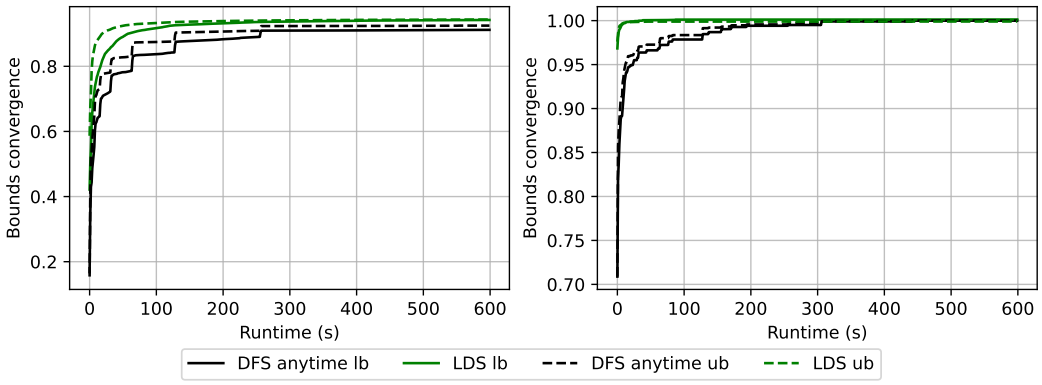

Figure 4 Lower and upper bound convergence towards the true probability $p^\star$ on Bayesian networks (left) and reachability queries (right) for the two versions of `Schlandals`.

counter is possible. All other methods provide a lower bound on the probability during their execution, which is used as a comparison in the figure. Figure 3 shows the average convergence on all instances (solid lines), but also on easy (dotted lines) and hard instances (dashed lines). We use a simple threshold to consider an instance difficult or not: if `Schlandals` can solve the instance in less than a minute, we consider the instance as easy. For the Bayesian networks, the true probability is known for each instance, but this is not true for reachability queries. Hence, for the later problem, we only consider, in this graph, the instances for which at least one solver can produce an exact solution. Such instances are mainly the easiest ones, explaining why the methods converge faster than for Bayesian networks.

As previously noted, `Schlandals` is the best-performing method. Hence, its estimated probability converges, on average, to the true probability. However, the gap between the convergence of the easy and hard instances is much larger for the DFS version of `Schlandals` than for any other method. Due to its depth-first nature, it stays in parts of the search space containing small-weight interpretations. On the other hand, `LDS` suffers much less from this problem, and the convergence gap between the easy and the hard instances is smaller. Due to its inability to solve the hardest instances, `Toulbar2` does not converge, on average, toward the true probability. We hypothesize that it gets trapped in part of the search space where it accumulates little probability mass before being timed out. Finally, due to their poor scalability on weighted problems, `PartialKC` and `ApproxMC` converge poorly. Indeed, they can not produce a first solution for many hard instances. Hence, the default lower bound on these instances is 0, pulling their average convergence towards that value.

Finally, we evaluate if limited discrepancy search effectively allows finding likely models first. To do so, we analyze how the lower and upper bounds converge toward the true probability. If our search-based methods find the most likely model first, the bounds should converge quickly towards the probability, then plateau until the final convergence or timeout. To assess the convergence of a lower bound $P_l[F]$, we use the metric $P_l[F]/P[F] \in [0, 1]$; when no solution has been found, it is 0 and when all solutions have been found it is 1. For the upper bound $P_u[F]$, we use $(1 - P_u[F])/(1 - P[F])$, which exhibits the same characteristics. Figure 4 shows, for both problems, the average convergence, using the metrics defined above, for `DFS` and `LDS`. It can be seen that both methods quickly accumulate probability mass for models *and non-models* of F, validating that our value-selection heuristic favors the most likely models. Moreover, `LDS` performs significantly better than `DFS`, reaching an average of more than 0.9 for both metrics in less than 100 seconds.

6 Conclusion

In this work, we proposed a new method for performing (projected) weighted model counting that can be stopped at any time during its execution. Unlike other anytime methods, the proposed approach provides deterministic lower *and upper* bounds on the true count. We have shown that the propagators of DPLL-style weighted model counters can be leveraged to maintain such an upper bound during exploration. We proposed a simple heuristic that favors exploring the highly weighted interpretations first, fastening the convergence of the bounds. We enhanced this convergence further using *Limited Discrepancy Search* (LDS). Finally, we have shown that these lower and upper bounds can be linked to the well-known notion of ε-approximations; hence, the proposed method can be used seamlessly to perform an interruptable computation for either the true weighted model count or an ε-approximation. We implemented this method in the `Schlandals` solver, specialized for probabilistic inference, and evaluated it against state-of-the-art methods on probabilistic reasoning tasks. Our experiments show that `Schlandals` is the best-performing solver on the studied tasks and that the LDS-based search outperforms classical DFS when computing ε-approximations.

The method presented in this work uses a simple strategy for the discrepancy, and more work could be done to enhance the performance of the LDS-based search. Moreover, the method presented here could be applied to other weighted model counters, including those that support arbitrary CNF formulas, such as provided in model counting competitions; moreover, an interesting research direction is to apply LDS to unweighted model counting. In this case, our simple heuristic can not be applied since all interpretations have the same weight, and another heuristic would be necessary.

References

1 Supratik Chakraborty, Daniel Fremont, Kuldeep Meel, Sanjit Seshia, and Moshe Vardi. Distribution-aware sampling and weighted model counting for SAT. In *AAAI*, 2014.

2 Supratik Chakraborty, Dror Fried, Kuldeep S. Meel, and Moshe Y. Vardi. From Weighted to Unweighted Model Counting. In *IJCAI*, 2015.

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. 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.

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

6 Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

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

8 Alexandre Dubray. Schlandals. Software, version 1.0.3., swhId: `swh:1:dir:3ffca0d07dbd88cffbbdfda6e3f0ae09a9e77ac0` (visited on 2024-08-19). URL: `https://github.com/aia-uclouvain/schlandals`.

9 Alexandre Dubray, Pierre Schaus, and Siegfried Nijssen. Probabilistic Inference by Projected Weighted Model Counting on Horn Clauses. In *DROPS-IDN/v2/Document/10.4230/LIPIcs.CP.2023.15*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPIcs.CP.2023.15`.

10 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.

11 Vibhav Gogate and Rina Dechter. SampleSearch: Importance sampling in presence of determinism. *Artificial Intelligence*, 175(2), 2011.

12 Vibhav Gogate and Rina Dechter. Importance sampling-based estimation over and/or search spaces for graphical models. *Artificial Intelligence*, 184, 2012.

13 Carla P. Gomes, Jörg Hoffmann, Ashish Sabharwal, and Bart Selman. From Sampling to Model Counting. In *IJCAI*, volume 2007, 2007.

14 William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *IJCAI (1)*, 1995.

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

16 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.

17 Yong Lai, Kuldeep S. Meel, and Roland HC Yap. The power of literal equivalence in model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 2021.

18 Yong Lai, Kuldeep S. Meel, and Roland HC Yap. Fast Converging Anytime Model Counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 2023.

19 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.

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 T. Schiex, S. de Givry, and M. Sanchez. Toulbar2—an open source weighted constraint satisfaction solver. *URL http://mulcyber. toulouse. inra. fr/projects/toulbar2*, 2006.

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

23 Mate Soos, Stephan Gocht, and Kuldeep S. Meel. Tinted, Detached, and Lazy CNF-XOR solving and its Applications to Counting and Sampling. In *CCAV*, 2020.

24 Mate Soos and Kuldeep S. Meel. BIRD: Engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 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 Clément Viricel, David Simoncini, Sophie Barbe, and Thomas Schiex. Guaranteed Weighted Counting for Affinity Computation: Beyond Determinism and Structure. In Michel Rueher, editor, *CP*, 2016.

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 Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, and Luc De Raedt. Anytime inference in probabilistic logic programs with Tp-compilation. In *IJCAI*, 2015.

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