A New Exact Solver for (Weighted) Max#SAT

Gilles Audemard
CRIL, Univ. Artois & CNRS, Lens, France

Jean-Marie Lagniez
CRIL, Univ Artois & CNRS, Lens, France

Marie Miceli
CRIL, Univ Artois & CNRS, Lens, France

Abstract

We present and evaluate `d4Max`, an exact approach for solving the Weighted Max#SAT problem. The Max#SAT problem extends the model counting problem (#SAT) by considering a tripartition of the variables $\{X, Y, Z\}$, and consists in maximizing over X the number of assignments to Y that can be extended to a solution with some assignment to Z. The Weighted Max#SAT problem is an extension of the Max#SAT problem with weights associated on each interpretation. We test and compare our approach with other state-of-the-art solvers on the challenging task in probabilistic inference of finding the marginal maximum a posteriori probability (MMAP) of a given subset of the variables in a Bayesian network and on exist-random quantified SSAT benchmarks. The results clearly show the overall superiority of `d4Max` in term of speed and number of instances solved. Moreover, we experimentally show that, in general, `d4Max` is able to quickly spot a solution that is close to optimal, thereby opening the door to an efficient anytime approach.

2012 ACM Subject Classification Computing methodologies → Search methodologies; Software and its engineering → Software notations and tools

Keywords and phrases Max#SAT, EMaj-SAT, Weighted Projected Model Counting, SSAT

Digital Object Identifier 10.4230/LIPIcs.SAT.2022.28

Funding This work has benefited from the support of the AI Chair EXPEKCTATION (ANR-19-CHIA-0005-01) of the French National Research Agency. It was also partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

1 Introduction

Weighted model counting is a fundamental task that consists in computing the number of weighted models of a given propositional formula, typically in CNF. Model counting is computationally hard (#P-complete), and actually much harder in practice than satisfiability (the SAT problem). Weighted projected model counting is an extension of weighted model counting, that also considers a set of propositional variables Y to be forgotten. The objective is then to compute the weighted model count of the quantified Boolean formula $\exists Y.\Phi$ over its variables (i.e., the variables occurring in Φ but not in Y). These tasks are of tremendous importance to many problems, including probabilistic inference [52, 13, 2], explainable AI [43], planning [44, 19, 6], reliability [21], verification [22, 29], among others.

Real world scenarios often require reasoning in an uncertain environment, which leads to the use of PP oracles [24]. PP is the class of decision problems solvable by a probabilistic Turing machine in polynomial time, with an error probability of less than one half for all instances. Toda [57] showed that #P and PP are equally powerful when used as oracles, indicating that the class PP is closely related to #P. For many relevant problems in AI such as calculating maximum a posteriori hypotheses (MAP) [45], explainable AI [61], and probabilistic conformant planning [38], $\mathrm{NP}^{\mathrm{PP}}$ oracles are often necessary. As highlighted by Torán [58], $\mathrm{NP}^{\mathrm{PP}}$ class can be thought as the class of problems which consists in guessing a proof of polynomial size and verifying it using a PP oracle.

E-MajSAT [38] is one prototypical example of the NP^{PP}problems. It asks for a given CNF formula Φ over $X \cup Y$ whether there exists an assignment to variables in X such that half of its extensions satisfies Φ. In [48], Knot Pipatsrisawat and Adnan Darwiche introduced an optimization version of E-MajSAT, called functional E-MajSAT, that asks for a complete assignment τ of X such that the weighted model count of Φ conditioned by τ is maximal. On top of considering an optimization version of E-MajSAT, functional E-MajSAT also considers weights over variables of Y. Recently, Fremont *et al.* [23] proposed an unweighted extension of E-MajSAT, called Max#SAT, that introduces a set of variables to be forgotten. Given a CNF formula Φ over $X \cup Y \cup Z$, Max#SAT asks for a complete assignment τ of X so that the projected model count of $\exists Z.\Phi$ conditioned by τ is maximal.

In this paper we introduce the WeightedMax#SAT problem, which generalizes both functional E-MajSAT and Max#SAT problems and can be formulated as an Exist-Random-Exist quantified SSAT problem. Given a CNF formula Φ over $X \cup Y \cup Z$ and a weight function over literals of Φ, it asks for a complete assignment τ of X such that the weighted projected model count of $\exists Z.\Phi$ conditioned by τ is maximal. It extends both Max#SAT and functional E-MajSAT, as it considers a set of variables to be forgotten, as well as a weight function that is not limited to probabilities on variables. Considering unrestricted weight functions (weights still represented with a linear number of bits) increases the expressiveness of the language, and consequently allows the use of different encodings [8].

To tackle the WeightedMax#SAT problem in practice, we present a top-down tree-search procedure based on the model counter `d4`, called `d4Max`. The solver given in this paper extends the preliminary Max#SAT solver introduced in [3] by considering a weight function in the problem definition, which is necessary for several concrete applications s.t. probabilistic database, verification, XAI, ... We also add the requirements to efficiently turn `d4Max` into an anytime approach. These conditions can also be used to answer the decision version of WeightedMax#SAT, that asks whether there exists an assignment τ of X such that the weighted projected model count of $\exists Z.\Phi$ conditioned by τ is greater than some threshold.

For our experiments we first consider as a case study the inference query of MMAP on Bayesian Networks (BN) [47] (and Markov Networks (MN)), which asks to find the most probable combination of values of a set of random variables in a BN (or MN). In order to evaluate how efficient is `d4Max` on this problem, we first encode MMAP benchmarks into WeightedMax#SAT instances. We use a BN to CNF translator, based on the encoding presented in [13], which associates with an input graphical model (either a BN or a MN) a CNF formula and a weight map. For comparison purpose we use the state-of-the-art inference tool `merlin`. Because MMAP instances can be naturally encoded as Exist-Random SSAT problems, we also consider the solvers `erSSAT` [35] and `ClauSSat` [15] in our evaluation. The empirical results obtained show that `d4Max` is quite competitive on the MMAP instances in comparison to the SSAT solvers and the dedicated tool `merlin`. As a second test bed, we consider Exist-Random SSAT application formulas used in [35]. We show that `d4Max` outperforms `erSSAT` and `ClauSSat` on these benchmarks (which `merlin` can not handle). Finally, we experimentally show that in general, `d4Max` is able to quickly spot a solution close to the optimal, thereby being a quite efficient anytime solver.

The rest of the paper is organized as follows. After some formal preliminaries (Section 2), the WeightedMax#SAT problem is presented in Section 3. Section 4 describes `d4Max`, our exact WeightedMax#SAT solver, alongside an anytime version. An empirical evaluation is provided in Section 5, before the conclusion section (Section 6).

2 Preliminaries

Let X be a set of *Boolean variables* $\{x_1, \ldots, x_m\}$. A *literal* ℓ is a variable x or its negation. A *clause* is a disjunction of literals $(\ell_1 \vee \ldots \vee \ell_c)$ and a *term* is a conjunction of literals $(\ell_1 \wedge \ldots \wedge \ell_t)$. A *unit clause* contains only one literal. A propositional formula in *Conjunctive Normal Form* (CNF) is a conjunction of clauses. Clauses and terms will be denoted by lower greek letters $(\alpha, \tau, \ldots)$ while CNF formulas will be represented by upper ones $(\Phi, \Psi, \ldots)$. $Var(\Phi)$ is the set of variables occurring in Φ and $Lit(\Phi)$ is its set of literals. $Lit(X)$ would also denote the set of literals we can build over the set of variables X. For convenience, we will write $\Phi(X)$ to represent that Φ is defined on the set of variables X.

▶ **Example 1.** Let $\Phi = (x_1 \vee \neg y_1) \wedge (x_1 \vee \neg y_2) \wedge (x_1 \vee y_1 \vee y_2 \vee \neg y_3) \wedge (\neg x_1 \vee \neg y_1 \vee y_2) \wedge (\neg x_1 \vee y_1 \vee \neg y_2) \wedge (\neg x_1 \vee x_2 \vee \neg y_3) \wedge (\neg x_1 \vee \neg z_1)$ be a CNF formula. $Var(\Phi) = \{x_1, x_2, y_1, y_2, y_3, z_1\}$ and $Lit(\Phi) = \{x_1, x_2, y_1, y_2, \neg x_1, \neg y_1, \neg y_2, \neg y_3, \neg z_1\}$.

An *interpretation* or an *assignment* τ is a mapping from variables to $\{true, false\}$ and can be represented either as a set of literals or as a term. A literal ℓ (resp. $\neg\ell$) with $Var(\ell) = x$ is satisfied when $\tau(x) = true$ (resp. $\tau(x) = false$). An interpretation is *complete* when all variables from a formula are assigned, otherwise it is *partial*. A clause is satisfied when one of its literals is satisfied. An interpretation that satisfies all clauses from a CNF formula is called a *model*. The Boolean Satisfiability Problem (SAT) is the decision problem determining whether a model of a CNF formula exists [1]. We denote by $2^{Var(\Phi)}$ the set of all interpretations of a CNF formula Φ, and by $Mod(\Phi)$ the set of all models, with $Mod(\Phi) = \{\tau \in 2^{Var(\Phi)} \mid \tau \models \Phi\}$, $\models$ denoting logical entailment.

▶ **Example 2** (Example 1 cont'd)**.** The models of Φ are:

$\{\neg x_1, \neg x_2, \neg y_1, \neg y_2, \neg y_3, \neg z_1\}$ $\{\neg x_1, \neg x_2, \neg y_1, \neg y_2, \neg y_3, z_1\}$ $\{x_1, \neg x_2, y_1, y_2, \neg y_3, \neg z_1\}$
$\{\neg x_1, x_2, \neg y_1, \neg y_2, \neg y_3, z_1\}$ $\{\neg x_1, x_2, \neg y_1, \neg y_2, \neg y_3, \neg z_1\}$ $\{x_1, x_2, y_1, y_2, y_3, \neg z_1\}$
$\{x_1, \neg x_2, \neg y_1, \neg y_2, \neg y_3, \neg z_1\}$ $\{x_1, x_2, \neg y_1, \neg y_2, y_3, \neg z_1\}$ $\{x_1, x_2, y_1, y_2, \neg y_3, \neg z_1\}$
$\{x_1, x_2, \neg y_1, \neg y_2, \neg y_3, \neg z_1\}$

Given a CNF formula $\Phi(X, Y)$, with X and Y two disjoint sets of variables, $\exists Y.\Phi$ is a quantified Boolean formula denoting (up to logical equivalence, denoted by $\equiv$) the most general consequence of Φ which is independent from the variables of Y [37]. In consequence, $Mod(\exists Y.\Phi) = \{\tau \in 2^X \mid \exists \tau' \in 2^Y \text{ and } \tau \wedge \tau' \models \Phi\}$, i.e., the number of assignments to X such that there exists an extension to Y that satisfies Φ.

▶ **Example 3** (Example 1 cont'd)**.** The models of $\exists z_1.\Phi$ are listed below:

$\{\neg x_1, \neg x_2, \neg y_1, \neg y_2, \neg y_3\}$ $\{x_1, \neg x_2, y_1, y_2, \neg y_3\}$ $\{\neg x_1, x_2, \neg y_1, \neg y_2, \neg y_3\}$
$\{x_1, x_2, y_1, y_2, y_3\}$ $\{x_1, \neg x_2, \neg y_1, \neg y_2, \neg y_3\}$ $\{x_1, x_2, \neg y_1, \neg y_2, y_3\}$
$\{x_1, x_2, y_1, y_2, \neg y_3\}$ $\{x_1, x_2, \neg y_1, \neg y_2, \neg y_3\}$

If a variable $x \in X$ is not in $Var(\Phi(X))$, then it is said to be *free*. The conditioning of a CNF formula Φ by a consistent term τ is the CNF formula $\Phi|_\tau$, obtained from Φ by removing all satisfied clauses (which contain a literal $\ell \in \tau$) and all occurrences of $\neg\ell$. If a clause becomes empty, then Φ is falsified. The unit propagation of a unit clause (ℓ) is the conditioning of Φ on ℓ, which results into an equisatisfiable formula Φ_ℓ, meaning that $Mod(\Phi)$ and $Mod(\Phi|_\ell)$ have the same size. The *Boolean Constraint Propagation* (BCP) [64, 41] is the algorithm that, given a CNF formula, returns an equivalent CNF closed under propagation, i.e., it does not contain unit clauses.

▶ **Example 4** (Example 1 cont'ed)**.** $\Phi_{|\neg x_1} = (\neg y_1) \wedge (\neg y_2) \wedge (\neg y_3)$ is the formula obtained after conditioning Φ by the literal $\neg x_1$. In $\Phi_{|\neg x_1}$ the variables x_2 and z_1 are free.

Given a CNF formula $\Phi(X)$, the counting problem #SAT [26] returns the number of models of Φ over X, denoted by $\| \Phi \|$. Given a quantified CNF formula $\exists Y.\Phi(X,Y)$, Y being the set of variables to be forgotten, the projected model counting problem #$\exists$SAT [63] is to compute the number of assignments over X that have at least one extension to Y that satisfies $\Phi(X,Y)$, which we denote by $\| \exists Y.\Phi(X,Y) \|$. Naturally, if all variables from Φ are in Y, then the problem #$\exists$SAT boils down to solve the SAT problem.

▶ **Example 5** (Example 1 cont'd). $\| \Phi \| = 10$ and $\| \exists z_1.\Phi \| = 8$.

To distinguish relevant literals from irrelevant ones, a weight function $\omega : 2^{Lit(\Phi)} \rightarrow \mathbb{R}_*^+$ gives for each literal a weight. In unweighted CNF formulas, all literals have a weight of one, meaning that they have the same relevance. Given a weighted CNF formula Φ, the weight of a model τ of Φ, denoted $\omega(\tau)$, is the product of weights of $Lit(\tau)$. The problem of Weighted Model Counting (WMC) [52], denoted by $\| \Phi \|_\omega$, is to compute the weighted model count $\| \Phi \|_\omega = \sum_{\tau \in 2^{Var(\Phi)}} \omega(\tau)$, with $\tau \models \Phi$. The Weighted Projected Model Counting (WPMC) problem, denoted by $\| \exists Y.\Phi(X,Y) \|_\omega$, extends WMC by computing the weighted model count regarding a set of variables Y to forget, i.e., $\| \exists Y.\Phi(X,Y) \|_\omega = \sum_{\tau \in 2^X} \omega(\tau)$ with $\tau \models \exists Y.\Phi$. Let us remark that for existentially quantified variables Y, the weight function assigns 1 to $Lit(Y)$.

▶ **Example 6** (Example 1 cont'd). Let us consider the following weight function ω:

$\omega(x_1) = 1 \quad \omega(\neg x_1) = 1 \quad \omega(x_2) = 1 \quad \omega(\neg x_2) = 2 \quad \omega(y_1) = 0.5 \quad \omega(\neg y_1) = 0.5$
$\omega(y_2) = 0.5 \quad \omega(\neg y_2) = 0.5 \quad \omega(y_3) = 2 \quad \omega(\neg y_3) = 1 \quad \omega(z_1) = 1 \quad \omega(\neg z_1) = 1$

Given the weight function ω, we have $\omega(\neg x_1, x_2, \neg y_1, \neg y_2, \neg y_3, z_1) = 0.25$, $\| \Phi \|_\omega = 4$ and $\| \exists z_1.\Phi(\{x_1, x_2, y_1, y_2, y_3\}, \{z_1\}) \|_\omega = 3.25$.

Weighted model counting is computationally hard (#P-complete [59]), and actually much harder in practice than satisfiability (the SAT problem). Despite this difficulty, much effort has been spent in the last decade in developing new algorithms for model counting (either exact [51, 27, 56, 16, 7, 31, 42, 40, 17, 33, 50, 53, 34, 30, 20, 9] or approximate [55, 25, 11, 12, 60, 54]) which prove practical for larger and larger instances (see `https://mccompetition.org/`). Most exact counters are tree-search algorithms exploring the whole space of propositional interpretations and take advantage of several dedicated techniques. More precisely, these counters work by recursively branching on a variable x until either a contradiction (non-chronological backtracking) is reached or the formula is satisfied, relying on the simple observation that $\Phi = (x \wedge \Phi_{|x}) \vee (\neg x \wedge \Phi_{|\neg x})$ (Shannon expansion), which translates in terms of model counting into $\| \Phi \| = \| \Phi_{|x} \| + \| \Phi_{|\neg x} \|$. To avoid visiting all possible branches, the computed values are memorized so that if the algorithm is recursively called on a subformula that has already been seen during computation, the number of models of this subformula is directly returned by the cache (component caching). Another improvement of this algorithm is based on the observation that if $\Phi = \Phi_1 \wedge \Phi_2$ with $Var(\Phi_1) \cap Var(\Phi_2) = \emptyset$ (disjoint component analysis), then $\| \Phi \| = \| \Phi_{|x} \| \times \| \Phi_{|\neg x} \|$, which allows to significantly reduce the number of recursive calls. This algorithm can be turned into a projected model counter algorithm by only branching on projected variables and by associating the constants 0 or 1 to the subformulas that only contain existentially quantified variables (0 if the subformula is unsatisfiable, 1 otherwise). For weighted model counting, it is enough to slightly modify the branching rule by considering weights, i.e., $\| \Phi \|_\omega = \omega(x) \times \| \Phi_{|x} \|_\omega + \omega(\neg x) \times \| \Phi_{|\neg x} \|_\omega$.

3 The Weighted Max#SAT problem

In this section we propose to unify both the functional E-MajSAT [48] and the Max#SAT [23] problems under a new problem, called the Maximum Weighted Model Counting problem (WeightedMax#SAT for short). Let us first introduce what are functional E-MajSAT and Max#SAT problems.

Given a CNF formula Φ, MajSAT(Φ) asks whether a propositional formula is satisfied by a majority of assignments ($2^{n-1}+1$ for a formula containing n variables). This problem, an extension of the SAT problem, is the PP-complete problem of reference. Given a CNF formula $\Phi(X,Y)$, X and Y being disjoint, E-MajSAT($\Phi(X,Y)$) [38] asks whether there exists an assignment τ to X such that the majority of complete assignments to Y satisfies the formula Φ conditioned on τ. E-MajSAT is an $\mathrm{NP}^{\mathrm{PP}}$-complete problem [38, 46] and is a special case of the stochastic satisfiability (SSAT) problem [39].

▶ **Example 7** (Example 1 cont'd). Φ is satisfied by 10 assignments, which do not constitute the majority of assignments (that is 33 models). Thus, MajSAT(Φ) is false. Given $X = \{y_1, y_2, y_3\}$ and $Y = \{x_1, x_2, z_1\}$, $\tau = \{\neg y_1, \neg y_2, \neg y_3\}$ is a solution for E-MajSAT($\Phi(X,Y)$). Indeed, $\Phi_{|\tau} = (\neg x_1 \vee \neg t_1)$ is satisfied by 6 assignments (the majority being 5 assignments).

In [48] the authors propose an optimization version of E-MajSAT, called *functional* E-MajSAT. In this extension, variables are partitioned into choice variables X and chance variables Y, that are assigned respectively by the user and by the nature. Given a weight function ω that assigns probabilities on chance variables, functional E-MajSAT($\Phi(X,Y),\omega$) asks for the assignment τ to X that maximizes the weighted model count of $\Phi_{|\tau}$, which corresponds to the maximum probability of the functional E-MajSAT problem on $\Phi(X,Y)$.

▶ **Example 8** (Example 1 cont'd). Let us split $Var(\Phi)$ into $X = \{x_1, x_2, z_1\}$ and $Y = \{y_1, y_2, y_3\}$. ω is the weight function such that $\omega(y_1) = 0.6$, $\omega(y_2) = 0.5$, $\omega(y_3) = 0.1$ (we have $\omega(\neg y_i) = 1 - \omega(y_i)$). $\| \Phi_{|\{x_1,x_2,\neg z_1\}} \|_\omega = 1.5$ is the maximum probability of the functional E-MajSAT problem on $\Phi(X,Y)$ over ω.

In [23] the authors propose an extension of functional E-MajSAT, called Max#SAT. Given a formula Φ over disjoint sets of variables X, Y, and Z, the Max#SAT problem is to maximize over X the number of assignments to Y that have an extension over Z which satisfies Φ. Contrarily to functional E-MajSAT, variables are all unweighted and some variables can be forgotten, which can be indispensable to model some concrete problems.

We can now formally define the Maximum Weighted Model Counting problem (WeightedMax#SAT), which generalizes both functional E-MajSAT and Max#SAT problems.

▶ **Definition 9** (WeightedMax#SAT problem). *Given a propositional formula $\Phi(X,Y,Z)$ over disjoint sets of variables X, Y, and Z, and a weight function ω over $Lit(X \cup Y)$, the WeightedMax#SAT problem is to determine an assignment τ to X that maximizes $\| \exists Z.\Phi(X,Y,Z)_{|\tau} \|_\omega$.*

▶ **Example 10** (Examples 1 and 6 cont'd). Let us split $Var(\Phi)$ into $X = \{x_1, x_2\}$, $Y = \{y_1, y_2, y_3\}$ and $Z = \{z_1\}$. $\tau = \{x_1, x_2\}$ is an optimal solution for both Max#SAT($\Phi(X,Y,Z)$) ($\| \exists z_1.\Phi_{|\tau} \| = 4$) and WeightedMax#SAT($\Phi(X,Y,Z)$, ω) ($\| \exists z_1.\Phi_{|\tau} \|_\omega = 1.5$).

For a given functional E-MajSAT problem $(\Phi(X,Y),\omega)$, it is easy to determine, using a WeightedMax#SAT oracle, a solution by considering the propositional formula $\Phi(X,Y,\emptyset)$ and the exactly same weight function ω. Similarly, we can translate a given Max#SAT

problem $\Phi(X, Y, Z)$ into a WeightedMax#SAT problem by considering the weight function that assigns 1 on each literal. A solution for a given WeightedMax#SAT problem $(\Phi(X, Y, Z), \omega)$ will be represented by a couple $\langle \tau, c \rangle$, where τ is an assignment of X that maximizes $\| \exists Z.\Phi(X, Y, Z)_{|\tau} \|_\omega$ and c is the weighted model count associated to τ. When convenient, the notation WMax#SAT$(\Phi(X, Y, Z), \omega)$ will also be used to represent a solution. We introduce three operators to combine solutions together: $\times$, $\div$, $>$, and the function `max`:

- $\langle \tau_1, c_1 \rangle \times \langle \tau_2, c_2 \rangle = \langle \tau_1 \cup \tau_2,\ c_1 \times c_2 \rangle$;
- $\langle \tau_1, c_1 \rangle \div \langle \tau_2, c_2 \rangle = \langle \tau_1 \setminus \tau_2,\ c_1 / c_2 \rangle$;
- $\langle \tau_1, c_1 \rangle > \langle \tau_2, c_2 \rangle$ is true when $c_1 > c_2$;
- $\texttt{max}(\langle \tau_1, c_1 \rangle, \langle \tau_2, c_2 \rangle) = (c_1 > c_2)\ ?\ \langle \tau_1, c_1 \rangle : \langle \tau_2, c_2 \rangle$.

4 An Exact Weighted Max#SAT Approach

In this section, we propose an algorithm to solve exactly the WeightedMax#SAT problem in practice. First, we focus on the case where an optimal solution is required. Afterwards, we show how this algorithm can be enhanced to become an anytime solver able to handle the decision problem associated to the WeightedMax#SAT problem.

4.1 Solving Exactly the WeightedMax#SAT Problem

Given a propositional formula $\Phi(X, Y, Z)$ over disjoint sets of variables X, Y, and Z, and a weight function ω over $Lit(X \cup Y)$, our approach takes advantage of the following observations:

1. If $x \in X$ then
   $\text{WMax\#SAT}(\Phi(X, Y, Z),\ \omega) = \texttt{max}(\text{WMax\#SAT}((\Phi \wedge x)(X, Y, Z),\ \omega),$
   $\text{WMax\#SAT}((\Phi \wedge \neg x)(X, Y, Z),\ \omega))$
2. If Φ is not satisfiable, then $\text{WMax\#SAT}(\Phi(X, Y, Z), \omega) = \langle \tau \in 2^X,\ 0 \rangle$
3. If $X = \emptyset$, then $\text{WMax\#SAT}(\Phi(X, Y, Z), \omega) = \langle \emptyset,\ \| \exists Z.\Phi \|_\omega \rangle$
4. If ℓ is unit in Φ and $Var(\ell) \in X$, then
   $\text{WMax\#SAT}(\Phi(X, Y, Z), \omega) = \langle \ell,\ \omega(\ell) \rangle \times \text{WMax\#SAT}(\Phi_{|\ell}(X \setminus \{Var(\ell)\}, Y, Z), \omega)$
5. If $x \in X$ is free in $\Phi(X, Y, Z)$, then
   $\text{WMax\#SAT}(\Phi(X, Y, Z), \omega) = \langle \ell,\ \omega(\ell) \rangle \times \text{WMax\#SAT}(\Phi(X \setminus \{x\}, Y, Z), \omega)$
   with $\ell = x$ if $\omega(\ell) \geq \omega(\neg\ell)$ and $\ell = \neg x$ otherwise
6. If ℓ is unit in Φ and $Var(\ell) \in Y$, then
   $\text{WMax\#SAT}(\Phi(X, Y, Z), \omega) = \langle \emptyset,\ \omega(\ell) \rangle \times \text{WMax\#SAT}(\Phi_{|\ell}(X, Y \setminus \{Var(\ell)\}, Z), \omega)$
7. If $y \in Y$ is free in $\Phi(X, Y, Z)$, then
   $\text{WMax\#SAT}(\Phi(X, Y, Z), \omega) = \text{WMax\#SAT}(\Phi(X, Y \setminus \{y\}, Z), \omega) \times \langle \emptyset,\ \omega(y) + \omega(\neg y) \rangle$
8. If $\Phi(X, Y, Z) \equiv \Phi_1(X_1, Y_1, Z_1) \wedge \Phi_2(X_2, Y_2, Z_2)$ with $Var(\Phi_1) \cap Var(\Phi_2) = \emptyset$, then
   $\text{WMax\#SAT}(\Phi(X, Y, Z), \omega) = \text{WMax\#SAT}(\Phi_1(X_1, Y_1, Z_1), \omega)$
   $\times\ \text{WMax\#SAT}(\Phi_2(X_2, Y_2, Z_2), \omega)$

Let us give the intuition of why these observations hold, with the help of the following example:

▶ **Example 11.** Let $\exists Z.\Phi(X, Y, Z) = (a \vee b) \wedge (a \vee c \vee d) \wedge (\neg a \vee c \vee d) \wedge (\neg a \vee \neg c \vee \neg d) \wedge (\neg a \vee \neg c \vee d) \wedge (e \vee f)$ be a weighted projected CNF formula, with $X = \{a, b, e\}$, $Y = \{c, f\}$ and $Z = \{d\}$, and ω the weight function such that $\omega(a) = 0.4$, $\omega(b) = 0.3$, $\omega(c) = 0.7$, $\omega(e) = 0.3$, $\omega(f) = 0.5$ (wih $\omega(\ell) = 1 - \omega(\neg\ell)$). The tree depicted in Figure 1 represents a possible trace of `d4Max` regarding $\exists Z.\Phi(X, Y, Z)$.

By definition, if x belongs to X, then the variable x must be assigned in the solution. Let $\langle \tau_x, c_x \rangle$ and $\langle \tau_{\neg x}, c_{\neg x} \rangle$ evaluate the two possible situations for x, with $\langle \tau_x, c_x \rangle = \text{WMax\#SAT}((\Phi \wedge x)(X, Y, Z),\ \omega)$ and $\langle \tau_{\neg x}, c_{\neg x} \rangle = \text{WMax\#SAT}((\Phi \wedge \neg x)(X, Y, Z),\ \omega)$.

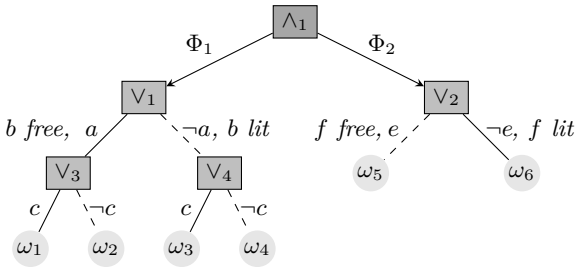


Figure 1 Possible trace of `d4Max` regarding the formula given in Example 11.

It is clear that an optimal solution for WMax#SAT($\Phi(X,Y,Z)$, ω) should maximize the number of the weighted model count, that is $\langle \tau_x, c_x \rangle$ if $c_x > c_{\neg x}$ and $\langle \tau_{\neg x}, c_{\neg x} \rangle$ otherwise (see the operator `max` in Section 3). Concerning Observation 2, since the weight function only assigns strictly positive values to literals, if Φ is unsatisfiable, then whatever the assignment to X is, the weighted model count will be zero (i.e., $\langle \tau \in 2^X, 0 \rangle$ is a solution). Observation 3 is a direct consequence of the WeightedMax#SAT definition. Because weights are strictly positive, Observation 4 can be obtained from Observations 1 and 2. If $x \in X$ is free in $\Phi(X,Y,Z)$, then $\Phi \wedge x \equiv \Phi \wedge \neg x$. Let $\langle \tau, c \rangle = \text{WMax\#SAT}(\Phi(X \setminus \{x\}, Y, Z), \omega)$. Since x is free in Φ, then WMax#SAT$((\Phi \wedge x)(X,Y,Z), \omega) = \langle \tau \wedge x, c \times w(x) \rangle$. Consequently, choosing the literal of x that is maximal regarding ω constructs an optimal solution (Observation 5). Since unit literals and free variables from Y and Z can be accumulated and considered only when the set of optimization variables is empty, we directly get that Observations 6 and 7 are correct.

▶ **Example 12** (Example 11 cont'd). As in $\vee_3$ there is no more X variables, we know that ω_1 is equal to 0 (Observation 2, no extension to Z satisfies $\Phi_1|_{\{a,c\}}$), and that ω_2 is equal to $\omega(c)$ (Observation 3). Then, as b is free, $\omega(\Psi_1|_a)$ is equal to $(\omega_1 + \omega_2) \times \omega(\neg b) = 0.49$, as per Observation 5 and as $\omega(\neg b) > \omega(b)$. Likewise, there is no more X variables in $\vee_4$ and whatever the valuation of c, there exists an extension to d that satisfies $\Psi_1|_{\neg a}$. Thus, as b is propagated to true when a is fixed to false to ensure the satisfiability of Φ_1, $\omega(\Phi_{\neg a}) = (\omega_3 + \omega_4) \times \omega(b) = 0.3$ (Observation 4). As in $\vee_1$, a belongs to X, $\omega(\vee_1) = max(\omega(\Phi_a) \times \omega(a), \omega(\Phi_{\neg a}) \times \omega(\neg a)) = max(0.196, 0.18)$ (Observation 1). Thus, the solution returned by $\vee_1$ is $\{a \wedge \neg b, 0.196\}$. Concerning Φ_2, ω_5 is equal to $\omega(f) + \omega(\neg f) = 1$ (Observation 7, f being free) and ω_6 to $\omega(f)$ (Observation 6, f being fixed). Then, as in $\vee_1$, $\omega(\Phi_2) = max((\Phi_2|_e \times \omega(e), \Phi_2|_{\neg e} \times \omega(\neg e)) = max(0.3, 0.35)$ (Observation 1, $e \in X$). The solution returned by $\vee_2$ is $\{\neg e, 0.35\}$.

Regarding Observation 8, let us suppose that $\Phi(X,Y,Z)$ can be split into two connected components $\Phi_1(X_1,Y_1,Z_1)$ and $\Phi_2(X_2,Y_2,Z_2)$, i.e., $\Phi \equiv \Phi_1 \wedge \Phi_2$ and $Var(\Phi_1) \cap Var(\Phi_2) = \emptyset$. Let us consider two optimal solutions for the corresponding subproblems: $\langle \tau_1, c_1 \rangle = \text{WMax\#SAT}(\Phi_1(X_1,Y_1,Z_1), \omega)$ and $\langle \tau_2, c_2 \rangle = \text{WMax\#SAT}(\Phi_2(X_2,Y_2,Z_2), \omega)$. We already know that connected components can be leveraged for weighted projected model counting. Then, $\| \exists Z.\Phi_{|\tau_1 \wedge \tau_2} \|_\omega = c_1 \times c_2$. Let $\langle \tau_1' \wedge \tau_2', c \rangle = \text{WMax\#SAT}(\Phi(X,Y,Z), \omega)$ with $Var(\tau_1') \cap X_2 = Var(\tau_2') \cap X_1 = \emptyset$ and suppose, for sake of contradiction, that $c > c_1 \times c_2$. Since $\Phi_1(X_1,Y_1,Z_1)$ and $\Phi_2(X_2,Y_2,Z_2)$ do not share any variables, $c = c_1' \times c_2' = \| \exists Z_1.\Phi(X_1,Y_1,Z_1)_{|\tau_1'} \|_\omega \times \| \exists Z_2.\Phi(X_2,Y_2,Z_2)_{|\tau_2'} \|_\omega$. Then, $c > c_1 \times c_2$ means either $c_1' > c_1$ or $c_2' > c_2$. Without loss of generality, let us suppose that $c_1' > c_1$. Thus, $\langle \tau_1', c_1' \rangle$ is a solution for WMax#SAT$(\Phi_1(X_1,Y_1,Z_1), \omega)$ and would be strictly better than $\langle \tau_1, c_1 \rangle$, which contradicts the fact that $\langle \tau_1, c_1 \rangle$ is an optimal solution for WMax#SAT$(\Phi_1(X_1,Y_1,Z_1), \omega)$.

Algorithm 1 `d4Max`.

Input: $\Phi(X, Y, Z)$: a CNF formula, and ω a weight function over $Lit(\Phi)$.
Output: $ret = \langle \tau, c \rangle$ s.t. τ is an optimal WeightedMax#SAT solution for the given input, with $c = \| \exists Z.\Phi_{|\tau} \|_\omega$ the weighted model count obtained.
Global variables: $best = \langle \beta, b \rangle$ is the best solution found so far (it is initialized to $\langle \emptyset, -1 \rangle$), and $scale = \langle \sigma, s \rangle$ is a partial solution (it is initialized to $\langle \emptyset, 1 \rangle$).

1 **if** Φ *is unsat* **then return** $\langle \tau \in 2^X, 0 \rangle$
2 $(\Phi', \mu) \leftarrow$ `BCP` (Φ)
3 $m \leftarrow \omega(\mu \cap Lit(X \cup Y)) \times \prod_{x \in X \setminus Var(\Phi')} \texttt{max}(w(x), w(\neg x)) \times \prod_{y \in Y \setminus Var(\Phi')} w(y) + w(\neg y)$
4 $\alpha \leftarrow \mu \cap Lit(X)$
5 $\alpha \leftarrow \alpha \cup \{\ell \mid x \in X \setminus Var(\Phi')$ with $\ell = x$ if $w(\ell) \geq w(\neg \ell)$ and $\ell = \neg x$ otherwise$\}$
6 $ret \leftarrow \langle \emptyset, 1 \rangle$
7 **if** $cache[\Phi'] \neq \emptyset$ **then** $ret \leftarrow$`cache`$[\Phi']$
8 **else if** $Var(\Phi') \cap X = \emptyset$ **then** $ret \leftarrow \langle \emptyset, \| \exists Z.\Phi' \|_\omega \rangle$
else
9 $saveScale \leftarrow scale$; $scale \leftarrow scale \times \langle \alpha, m \rangle$
10 $\{\Phi_1, \ldots, \Phi_j\} \leftarrow$ `connectedComponent`(Φ')
11 **if** $j > 1$ **then**
12 $map \leftarrow$ `dig(`$\{\Phi_1, \ldots, \Phi_j\}$`,`$(X, Y, Z)$`, `$\omega$`) // with` $map[\Phi_i] = sol_i$
13 $scale \leftarrow scale \times map[\Phi_1] \times \ldots \times map[\Phi_j]$
14 **for** $\Phi_i \in \{\Phi_1, \ldots, \Phi_j\}$ **do**
15 $scale \leftarrow scale \div map[\Phi_j]$
16 $current \leftarrow$ `d4Max`$(\Phi_i(X \cap Var(\Phi_i),\ Y \cap Var(\Phi_i),\ Z \cap Var(\Phi_i)), \omega)$
17 $scale \leftarrow scale \times current$
18 $ret \leftarrow ret \times current$
else
19 $v \leftarrow$ `selectVar`$(Var(\Phi') \cap X)$
20 $\langle \tau_1, c_1 \rangle \leftarrow$ `d4Max`$((\Phi' \wedge v)(X \cap Var(\Phi'), Y \cap Var(\Phi'), Z \cap Var(\Phi'), \omega)$
21 $\langle \tau_2, c_2 \rangle \leftarrow$ `d4Max`$((\Phi' \wedge \neg v)(X \cap Var(\Phi'),\ Y \cap Var(\Phi'),\ Z \cap Var(\Phi'), \omega)$
22 $ret \leftarrow$ `max`$(\langle \tau_1, c_1 \rangle,\ \langle \tau_2, c_2 \rangle)$
23 `cache`$[\Phi'] \leftarrow ret$
24 $scale \leftarrow saveScale$
25 **if** $ret \times \langle \alpha, m \rangle \times scale > best$ **then** $best \leftarrow ret \times \langle \alpha, m \rangle \times scale$
26 **return** $ret \times \langle \alpha, m \rangle$

▶ **Example 13** (Examples 11 and 12 cont'd). Previously, we compute that the solutions for Φ_1 and Φ_2 are respectively equal to $\{a \wedge \neg b, 0.196\}$ and $\{\neg e, 0.35\}$. As $\Phi \equiv \Phi_1 \wedge \Phi_2$ and $Var(\Phi_1) \cup Var(\Phi_2) = \emptyset$, we have, thanks to Observation 8, WMax#SAT$(\Phi(X,Y,Z), \omega) = \{a \wedge \neg b \wedge \neg e, 0.196 \times 0.35\}$.

Based on the model counter `d4` [33], `d4Max` is a top-down tree-search algorithm which is decomposed into two parts. As long as the current formula contains variables from X, we branch on such variables leveraging Observation 1. Once X is empty (line 8), we fall into the observation 3 and compute the assignment that maximizes $\| \exists Z.\Phi_\tau \|_\omega$.

Algorithm 1 provides the pseudo-code of `d4Max` that solves exactly WeightedMax#SAT (we leave for the next subsection the blue part of the algorithm which concerns the anytime version). It takes as an input a CNF $\Phi(X, Y, Z)$ and a weight function ω over the literals

of Φ. (X, Y, Z) forms a partition of $Var(\Phi)$, with X, Y and Z respectively the optimization, counting, and existentially quantified variables. It returns a term τ, corresponding to an assignment to X, and the projected number of weighted models of $\exists Z.\Phi_{|\tau}$ over Y.

First, at line 1, one tests whether the formula Φ is satisfiable (this case meets Observation 2). If not, whatever the interpretation considered on X is, the corresponding weighted model count would be equal to zero and `d4Max` would return $\langle \tau \in 2^X,\ 0 \rangle$. `BCP` simplifies Φ at line 2 and returns Φ', the formula obtained after applying unit propagation over Φ, alongside the unit literals μ which were propagated. Since all variables that became free during propagation are not returned by `BCP`, we need to retrieve them to compute the correct solution (for both the weight model count and the term). Propagated literals that belong in X or Y also need to be considered in the returned solution. Consequently, we compute a multiplicative factor m (line 3) that is obtained from the observations 4–7 and that considers both unit literals ($\omega(\mu \cap Lit(X \cup Y))$) and free variables (the remaining part). This multiplicative factor is then used to get the correct count when the solution is returned by `d4Max` (line 26). Similarly, thanks to the observations 4 and 5, we compute a term α that will be concatenated with the solution returned by `d4Max` to get the correct solution (lines 4–5).

Afterwards, we initialize $ret = \langle \emptyset, 1 \rangle$ (line 6), a temporary variable which serves to store the result returned at line 26. We use a cache to avoid computing once more an already encountered subformula (line 7). More precisely, each time a new value of $\langle \tau, c \rangle$ is computed for a given input Φ, it is memoized in a hash table (Φ being the key and $\langle \tau, c \rangle$ the associated value). Thus, if Φ' has already been cached, `cache[`Φ'`]` can be reused and then ret is set accordingly (line 7). As we ensure that the formula is satisfiable (line 1), we can not have cache inconsistency [52, 10], that makes the formula memoized line 23 usable.

Next, if the set of optimization variables is empty, we take advantage of Observation 3 to directly construct a solution by calling a weighted projected model counter on Φ' (line 8).

Finally, we consider the case where $Var(\Phi') \cap X \neq \emptyset$ (lines 9–23). We first take advantage of the dynamic decomposition of `d4`. If Φ can be partitioned into disjoint subformulas $\{\Phi_1, \ldots, \Phi_d\}$, then regarding the observation 8, each subformula Φ_i is treated separately and their solutions aggregated afterwards (lines 11–18). More precisely, `connectedComponent` partitions Φ' into a set of disjoint connected components at line 10. If there are more than one component, then at lines 11–18, ret is used to aggregate solutions of all subcomponents. If Φ' can not be partitioned into more than one component ($j = 1$), a variable v from $Var(\Phi') \cap X$ is selected (line 19). In our experiments the heuristic VSADS is used [52]. We compute via two recursive calls to `d4Max` the solutions for either conditioning Φ' by v or $\neg v$ (lines 20 and 21). As pointed out by Observation 1, the optimal solution is equal to the one having the higher weighted model count (line 22). At line 26, we return the current result stored in ret by extending its term with α and by multiplying its number of models by m.

4.2 An Anytime Weighted Max#SAT Solver

The previous subsection left the blue part of Algorithm 1 undescribed. However, it is primordial whether we want to turn `d4Max` into an anytime solver. Without it, we can not update from anywhere in the search space the current best solution found as we would need to wait for the completion of the recursion, preventing the anytime aspect.

Even if an optimal solution of the input formula is returned at line 26 of Algorithm 1, it can not be considered as an optimal solution of the original problem as it misses information given at the end of the recursion. First, information about which literals are assigned or which variables are free is lost when `d4Max` is recursively called. To recover this information

Algorithm 2 `dig`.

Input: $\{\Phi_1, \ldots, \Phi_j\}$: a set of satisfiable CNF formulas, (X, Y, Z) a partition of propositional variables, and ω a weight function over $Lit(\Phi_i)$.
Output: $map = \{\Phi_1 : \langle \tau_1, c_1 \rangle, \ldots, \Phi_j : \langle \tau_j, c_j \rangle\}$ s.t. $\forall \Phi_i$, $\tau_i \in 2^{X \cap Var(\Phi_i)}$ and $c_i = \| \exists Z.\Phi_i(X \cap Var(\Phi_i),\ Y \cap Var(\Phi_i),\ Z \cap Var(\Phi_i))_{|\tau_i} \|_\omega$.

1 $map \leftarrow \{\}$
2 **for** $\Phi_i \in \{\Phi_1, \ldots, \Phi_j\}$ **do**
3 $\quad \tau_i \leftarrow$ `solve`$(\Phi_i) \cap Lit(Var(\Phi) \cap X)$
4 $\quad map[\Phi_i] \leftarrow \langle \tau_i, \| \exists Z.\Phi_i(X \cap Var(\Phi_i), Y \cap Var(\Phi_i), Z \cap Var(\Phi_i))_{|\tau_i} \|_\omega \rangle$
5 **return** map

we need to aggregate to a global variable, called *scale* and initially set to $\langle \emptyset, 1 \rangle$, the partial solutions $\langle \alpha, m \rangle$ that are extracted from the free variables and the unit literals of the input formula (lines 3–5 of Algorithm 1). Nevertheless, even if *scale* now gives enough information to retrieve the context for the considered formula, it is still possible that *ret*, updated with the information contained in *scale*, is greater than the optimal solution. Indeed, if the formula has been previously partitioned into disjointed components, the current solution only concerns the current component and not the global formula. Let us illustrate this situation with the formula of Example 1 and the weight function given in Example 6.

In Algorithm 1, if we first branch on x_1 and call `d4Max` on $(\Phi \wedge x_1)$, then we get the following two connected components $\{\{(x_2 \vee \neg y_3)\},\ \{(\neg y_1 \vee y_2),\ (y_1 \vee \neg y_2)\}\}$ and $scale = \langle x_1, 1 \rangle$. Let us consider first the component $\{(x_2 \vee \neg y_3)\}$, then the optimal solution for this formula would be $\langle x_2, 3 \rangle$, which becomes $\langle x_1 \wedge x_2, 3 \rangle$ when updated with the context. Clearly this solution is greater than the optimal solution which is equal to 1.5 for this problem (Example 10). This situation occurs because the weight function can assign values in $[0, 1]$ to literals. Thus, we need to consider all the connected components to obtain a complete solution.

To overcome this issue, that arises when considering connected components, we consider suboptimal solutions on the connected components that have not yet been checked. By updating *scale* accordingly, it lets us consider solutions that are complete and thus which can be used to update the best solution found so far. The function `dig`, described in Algorithm 2, constructs a map that respects this requirement. Given a set of satisfiable CNF formulas, a partition of variables and a weight function, this function computes a map that associates for each given CNF formula a suboptimal solution. It begins by initializing an empty map (line 1), that will be filled in within the *for* loop (lines 2–4) and finally returned at line 5. For each CNF formula Φ_i, a suboptimal solution is computed and saved in the returned map. First, a model of Φ_i is obtained by calling a SAT solver via the function call `solve`(Φ_i) (line 3) and narrowed afterwards to the literals of $Var(\Phi_i) \cap X$ to produce the assignment τ_i. Finally, a weighted projected model counter is called to get the weight model count c_i of Φ_i associated with τ_i in order to map with Φ_i the (potentially suboptimal) solution $\langle \tau_i, c_i \rangle$.

Now, let us continue the description of Algorithm 1. Besides the input/output variables, we consider two global variables: *best*, used to store the best solution found so far and initialized to $\langle \emptyset, -1 \rangle$, and *scale* the suboptimal solution described before, which is used to test if the current solution can be completed with a better solution than *best* (line 25). At line 9, *scale* is saved in *saveScale* and updated with the information about propagated literals and free variables from `BCP`. In order to keep up to date the variable for the caller, *scale* is reset to its initial value (line 24). The remaining blue part revolves around the connected component

management. We first compute a map that associates for each formula a suboptimal solution, by calling `dig` (line 12), and we update *scale* by aggregating all the computed potentially suboptimal solutions (line 13). Then, when a formula Φ_i is considered, we remove from *scale* the contribution of the suboptimal solution $map[\Phi_i]$ (line 15). Consequently, when `d4Max` is recursively called on the i^{th} component (line 16), *scale* only contributes for the other connected components. Because the solution computed line 16 is optimal, we can use it to update *scale* (line 17) and to improve the suboptimal solution used for the remaining connected components.

Let us remark that, in Algorithm 2, it would be possible to promote literals with high weights by modifying the polarity heuristic of the SAT solver in such a way that the solver, when deciding the phase of the selected variable, branches first on the literal with the highest weight. However, we have decided to keep the solver unchanged and to leave this lead for future works. Nonetheless, we chose to exploit this idea in Algorithm 1 by considering first the best literal (regarding the weight function) when calling recursively `d4Max` (lines 20–21). This is the version we ran for the experiments presented in the next section.

5 Experimental Evaluation

In order to evaluate `d4Max`, we consider benchmarks that model the problem of searching for the *marginal maximum a posteriori hypothesis* (MMAP) in probabilistic inference, as well as Exist-Random SSAT formulas. The experiments have been conducted on Intel Xeon E5-2643 (3.3 GHz) bi-processors with 64 GiB RAM. A time-out of thirty minutes and a memory-out of 32 GiB have been considered on each instance. The source code of `d4Max`, the benchmarks used, and logs of empirical results are available in supplementary materials.

On MMAP benchmarks, we compare `d4Max` with the inference tool `merlin` (`https://github.com/radum2275/merlin`), that implements state-of-the-art exact and approximate algorithms for probabilistic inference over Bayesian networks and Markov random fields. `merlin` takes upon entry instances respecting the UAI competition format, as well as two input files: one taking the evidence (in our case an empty set) and the other the selected MAP variables. We selected MMAP among the different probabilistic inference tasks it supports and the exact algorithm `bte` for *Bucket Tree Elimination* [28]. We also consider the SSAT solvers `erSSAT` (`https://github.com/NTU-ALComLab/ssatABC`) and `ClauSSat` (`https://github.com/NTU-ALComLab/ClauSSat`) for both MMAP and Exist-Random SSAT formulas. `ClauSSat` is executed using the option *sguwc* that enables all optimization techniques. `erSSat` is run sequentially using its default parameters. However, an underflow problem occuring whenever the probabilities are too small (which was our case) lead us to prefer the model counter `Cachet` [51], rather than using BDDs, to compute weighted model counts. This issue has also been pointed out by the *github* community. For `d4Max`, the branching heuristic used depends on the problem. Concerning MMAP benchmarks, we chose the *VSADS* heuristic [52], a well-established heuristic in model counters. For SSAT instances, the *DLCS* heuristic, which prioritizes variables that appear in more clauses, is used. The heuristic *best* selects the polarity to explore first of the maximizing variable chosen in decision nodes, taking into account the best solution found so far. Otherwise, default parameters are used.

MMAP Formulas

We considered 1190 weighted constraint networks from the `Compile!Project` (`http://www.cril.univ-artois.fr/kc/benchmarks.html`), corresponding to Bayesian networks or random Markov fields in the UAI competition format. They are gathered into seven data sets (see

https://www.ics.uci.edu/~dechter/softwares/benchmarks/UAI08 for more details about the benchmarks), as follows: *bn2o* (BN, 18), *diagnose* (BN, 100), *grids2* (BN, 320), *linkage* (RN, 22), *promedas* (RN, 238), *BN, relational* (395), *UAI06* (RN, 97).

In the following we present the experimental protocol we use to generate MMAP benchmarks from Bayesian networks (BN). Since Markov networks are, in essence, quite similar to Bayesian networks, the protocol would be the same. Given a BN and an assignment (called an *evidence*) on some variables, the MAP query is to find the configuration of a selected subset of MAP variables with the highest probability. The MMAP estimation, in addition, marginalizes a set of (hidden) variables, which means it averages their probability.

In our experiments, we searched for the most probable configuration among all possible combinations (i.e., without any evidence), of a subset of variables taken randomly from the original set of MAP variables. Obviously, this problem is computationally harder than one single #SAT query, as per each computation the associated probability is computed. Therefore, there is no need to consider all instances for which a single weighted model counting fails within 10 minutes. Thus, we retrieved all instances whose weighted model count was successfully computed by `d4` given this time limit. We also gather instances the inference tool `merlin` solved[1]. Finally, 861 instances were considered, `d4Max` and `merlin` having solved respectively 763 and 359 instances.

For each network, we first randomly selected four different sets of random variables, containing respectively 20%, 40%, 60% of the initial set, and a fixed-size set of 10 variables. These campaigns will be denoted p_{20}, p_{40}, p_{60} and f_{10}. Then, a total of 3444 instances were considered for this first experimentation. Afterwards, as `d4Max` takes as input weighted CNF instances, we translated the graphical models into CNF formulas and weight maps with the translator `bn2cnf` [8]. The random variables are encoded using a logarithmic boolean representation of the domains [8] and the conditional probability tables (CPTs) are translated into SAT using the classic encoding presented in [13].

Considering the WeightedMax#SAT instances, for a given BN problem encoded into SAT using `bn2cnf`, we put into X the propositional variables that represent the selected MAP variables. All remaining propositional variables (meaning those used to encode the remaining random variables and those added to represent CPTs) are put into Y. For this problem the set Z is always empty. Concerning the weight function ω, we simply use the one generated by `bn2cnf`. That is, for each variable x used to encode random variables $\omega(x) = \omega(\neg x) = 1$, and for each propositional variable encoding CTP probabilities $\omega(x) + \omega(\neg x) = 1$.

In order to take advantage of the two SSAT solvers[2], we convert MMAP problems into functional E-MajSAT problems by splitting the variables into MAP variables, that will be considered as existentially quantified, and the others variables, that will be considered as randomized quantified. About the randomized quantifiers, we consider for each variable x used to encode random variables $Pr(x) = Pr(\neg x) = 0.5$, and for each propositional variable encoding CTP probabilities $Pr(x) + Pr(\neg x) = 1$.

For each instance, we measured the time in seconds required by all solvers to find the best configuration. We also retrieve the best solutions found to ensure the exactness of all solvers, but we will not include them in the following figures. Table 1 summarizes the number of instances considered per each family, as well as the number of instances solved per each solver and per each campaign.

[1] For this selection phase, `erSSAT` and `ClauSSat` were not considered as they have the same behavior as `d4` on weighted counting problems.

[2] In [48], the authors proposed a branch-and-bound solver for functional E-MajSAT problems, that exploits upper bounds computed from d-DNNF [18] representation of the given formula. Unfortunately, this software is no more maintained and the authors were not able to provide us a version we could use for our experiments.

Table 1 Summary of the number of solved inputs per each family and per each campaign.

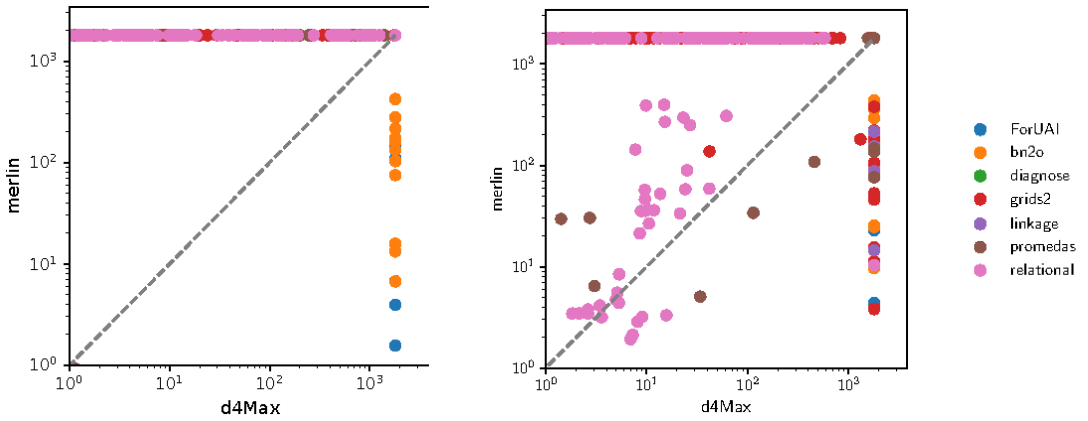
Benchmark		d4Max				merlin				erSSAT				ClauSSat			
Family	#I	p_{20}	p_{40}	p_{60}	f_{10}	p_{20}	p_{40}	p_{60}	f_{10}	p_{20}	p_{40}	p_{60}	f_{10}	p_{20}	p_{40}	p_{60}	f_{10}
bn2o	18	0	0	0	0	**17**	**6**	0	**18**	0	0	0	0	0	0	0	0
diagnose	100	0	0	0	0	0	0	**5**	**100**	0	0	0	0	0	0	0	0
grids2	244	**73**	**74**	**76**	**77**	0	0	21	56	0	0	0	0	0	0	0	7
linkage	5	0	0	0	0	0	0	0	**4**	0	0	0	0	0	0	0	0
promedas	91	**44**	**26**	21	**25**	8	17	**23**	**25**	6	6	3	16	1	1	1	1
relational	384	**256**	**257**	**269**	**383**	1	1	1	103	105	123	65	306	1	1	1	1
UAI06	19	5	6	**6**	5	**15**	**11**	5	**15**	3	1	0	1	0	0	0	0

As can be seen, the results show that `d4Max` outperforms the SSAT solvers `erSSAT` and `ClauSSat` whatever the campaign or the family considered. Specifically, `d4Max` is able to solve all the benchmarks solved by `ClauSSat`. `erSSAT` seems better than `ClauSSat`, which is quite surprising since `ClauSSat` is a more recent version of `erSSAT` with additional features. However, as it will be pointed out in the next paragraph, `erSSAT` appears to be incorrect since it can return solutions that are not optimal. For 6 ouf of 8 instances solved by `ClauSSAt`, `erSSAT` returns a different satisfying probability than `d4Max` and `ClauSSat`. Nevertheless, regardless the correctess of `erSSAT`, `d4Max` solves more instances than `erSSAT` and in quicker times. We can note that both SSAT solvers seem to behave similarly to `d4Max`, solving instances from the same families with a better rate when few variables are existentially quantified (f_{10}).

Concerning the comparison with the dedicated tool `merlin`, we show that `d4Max` and `merlin` solve different families of instances, as stated in Figures 2. Only campaigns p_{20} (Fig. 2a) and f_{10} (Fig. 2b) are figured, as there is no significant distinction between p_{20} and campaigns p_{40} and p_{60}. More precisely, `d4Max` and `merlin` only solve the same 184 instances: 14 for p_{20}, 21 for p_{40}, 19 for p_{60} and 130 for f_{10}. Both solvers completed more instances in f_{10} than in other campaigns. However, we can note that in general, `merlin` either succeeds within 10 seconds or fails. Thus, the solving times of `merlin` are significantly lower than `d4Max`'s, but `merlin`, no matter the campaign, also solves less instances that `d4Max`.

We remark that while `merlin` solves all *bn2o* benchmarks when there are less than 25% of selected MAP variables and all *diagnose* instances in campaign f_{10}, `d4Max` solves none (as both SSAT solvers). It may be due to an encoding problem. *bn2o* networks consist of noisy-OR relations between nodes distributed in two layers [32] and *diagnose* instances are hand-built and relatively large with approximately 200-300 nodes that assume causal independence, i.e., noisy-MAX relations. In [36], authors state that instances containing noisy-OR and noisy-MAX relations are intractable for weighted model counters if the encoding used does not exploit their semantics, which was not the case here. Thus, it may explain why `d4Max` could not solve these two families. We could also note that `merlin` scores among its longer solving times for *bn2o* instances, with in average 105 seconds against an handful for others families. In general, `merlin` has difficulties to solve instances whenever the number of MAP variables is large, and could only solve *diagnose* instances when the number of selected variables was fixed at ten (against between 41-66, 82-132 and 122-198 variables for respectively p_{20}, p_{40} and p_{60} campaigns).

Both `d4Max` and `merlin` perform relatively well on small instances of the family *UAI06*, excepted for graph coloring instances that none solves. Beside in campaign p_{60}, `merlin` completed more instances than `d4Max` in this category. Concerning the *grids* benchmarks,



(a) Campaign p_{20}. **(b)** Campaign f_{10}.

Figure 2 Comparative of solved inputs for `d4Max` and `merlin`.

they are separated into three groups depending on the fraction of deterministic variable, i.e. variables that are deterministic functions of their parents. `merlin` solves some instances for the first two groups (50% and 75%), mainly on f_{10}, but none for the last one (90%). `d4Max` solves less instances for 50% and 75% instances but completes the major part of the 90% grids, the remaining unsolved ones being the larger grids that were encoded with more than 2000 propositional variables. As other instances in the first two categories with less variables were unsolved, we can suppose that determinism was the major key to be solved by `d4Max`. As for *linkage* instances, there are relatively large in terms of MAP and propositional variables, and only four out of the 88 were solved by `merlin` for the last campaign (f_{10}). Behaviors for *promedas* benchmarks were similar for both tools, with the completed instances being the ones with few MAP and CNF variables.

The main discrepancy in the results remains with *relational* instances, in particular with *blockmap* and *students* ones. `d4Max` nearly solves all in mere seconds, while `merlin` completes none for p_{20}, p_{40} and p_{60}, only succeeding *students* family in f_{10}. In the same way, only an handful of *mastermind* benchmarks were solved by both solvers on the first three campaigns while all of them were solved by `d4Max` in f_{10} and 36 for `merlin` (out of 128). All *relational* instances have high levels of determinism, the main difference between *blockmap* and *students* with *masterminds* benchmarks being the ratio of the level of CPT parameters to the number of variables, higher for the two firsts [14].

Then, experiments show that although `d4Max` is generic and not dedicated for MMAP problem, it stays rather competitive and solves more instances than `merlin`, albeit less rapidly. Furthermore, `d4Max` seems to scale better than `merlin`, which solves for the most part when the set of maximizing variables was small. But, determinism rather seems to be important for `d4Max`, and a customized encoding for noisy-OR and noisy-MAX relations may be required to permit `d4Max` to solve more families.

Exist-Random Formulas

We considered all Exist-Random and Exist-Random-Exist formulas from the collection of SSAT instance (`https://github.com/NTU-ALComLab/ssat-benchmarks`), excepted for random k-CNF formulas. In Table 2, the first family (MaxCount) encodes maximum satisfiabilty,

quantitative information flow and problem synthesis into Exist-Random SSAT; the second family encodes the maximum probabilistic equivalence (MPEC); and the four last families encode planning problems (Conformant, Sand-Castle, Tiger and ToiletA).

Table 2 Summary of the number of solved Exist-Random SSAT inputs per each family.

Benchmark		d4Max	erSSAT	ClauSSat
Family	#I			
MaxCount	26	8	***9***	1
MPEC	60	11	***40***	4
Conformant	24	**2**	*1*	**2**
Sand-Castle	25	**22**	15	11
Tiger	5	**4**	3	2
ToiletA	77	**61**	*43*	49

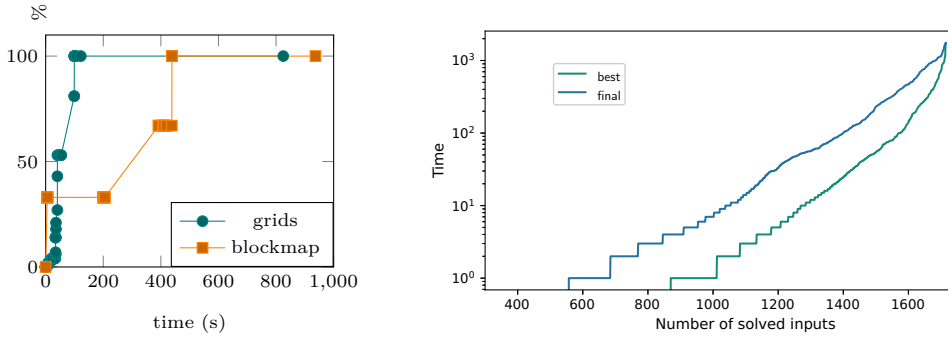
In general, whenever the numbers of variables and clauses are large (more than a thousand of each, mainly in MaxCount, MPEC and Conformant families), no solver found a solution. On smaller benchmarks (Sand-Castle, Tiger and ToiletA from the planning family), with at most a few hundred of variables, solvers are relatively efficient. Let us observe that d4Max clearly outperforms ClauSSat since it solves all the instances solved by ClauSSat, and in quicker times. Concerning erSSat, the evaluation is more complicated. Indeed, on several instances (indicated in italics in Figure 2), the results found by erSSAT either differ from the solutions given by d4Max and ClauSSAT or are erroneous. In the first case, we only compare the results when all three solvers terminated and consider significant discrepancies that do not revolve around approximation (for exemple $0.9772288[\dots]$ against $9.772290e^{-01}$). This issue, erSSAT disagreeing with d4Max and ClauSSat about the model counts, arises for two MaxCount instances, one Conformant benchmark and 34 ToiletA instances. The second case concerns benchmarks whose model counts are equal to 1, i.e., benchmarks where there exists an assignment over the variables of X such that any assignment over the variables of Y can be extended into a model. In such a case, we can check for the correctness of the solvers on these instances by replacing all randomized quantifiers with universal quantifiers and by using a QBF solver (in our case, DepQBF (https://lonsing.github.io/depqbf/)). If the QBF solver returns unsatisfiable, then the solvers that return a model count equal to 1 are incorrect. Following this protocol, we observe than erSSAT is incorrect on all benchmarks from the MPEC family it solves. It is important to point out that on all the benchmarks d4Max and ClauSSAT return a model count equal to 1, then the QBF version of the formula is well satisfiable. Thus, the relative effectiveness of erSSAT regarding the Exist-Random instances should be balanced with its discrepancies. When we compare d4Max with erSSAT when all solvers agree on the solutions (mainly on the Sand-Castle family), d4Max is quicker than erSSAT.

In the next section, we will evaluate d4Max anytime approach, to check if its longer resolution times can be compensated by its quickness to find a solution close to the optimal.

Evaluating d4Max Anytime Approach

In this part we want to evaluate how fast d4Max picks a solution whose weight model count is close to the optimal. To do so, we can not use any instance whose solution is unknown as we need to compare the solutions found with the optimal one. Therefore, we only consider benchmarks d4Max was able to solve and for each instance, we record whenever a better solution is found and its quality, i.e., how close it is to the optimal in terms of percentage.

At first, in addition to retrieve the time (in seconds) required to compute the optimal weight model count $final$, we also wanted to check when intermediate weight models counts reached at least 90% and 95% of $final$. However, experiments show that weight model counts rarely went through these levels and directly reached 100% of $final$ instead. We denote by *best* the first solution that reaches 100%. We can note that precision settings prevent from differentiating weighted model counts past the fiftieth digit. Therefore, a weight model count stamped 100% could, in fact, be slightly lower than the optimal one.



(a) Evolution of two instances. **(b)** Evaluation of `d4Max`.

Figure 3 Quality evolution of solutions generated by `d4Max`.

Figure 3a shows for two instances (from the families *grids2* and *blockmap*) the times when new better solutions have been computed, as well as their quality. For both instances, `d4Max` spotted a solution really close to the optimal well before the timeout. It found a good solution at 438.45 seconds for *blockmap* and at 99.46 seconds for *grids2*, while their complete times were respectively 937 and 824.98 seconds, meaning that at least half the time was dedicated into scanning the remaining of the search space. Prior levels were 67% and 81%, thus missing 90% and 95% stages. Figure 3b resumes for all instances solved by `d4Max` the times in seconds it took to find the first solution at 100% (*best*) and to complete the search (*final*). Experiments show that for 1469 out of 1613 instances, a good solution was found under 100 seconds. For all instances whose final time is relatively long (>100 seconds): 50% of them can have their final time cut at least by half, and 25% have a good enough solution in at most 10% of the final time. Thus, `d4Max` benefits from becoming an anytime solver, as for the major part, there is no need to wait for the completion of the algorithm. When a good estimation of the optimal solution is known before the execution, a threshold can also be set up to stop the search as soon as `d4Max` finds a solution that exceeds it.

We also looked at MMAP instances `d4Max` failed to solve but `merlin` completed. As `merlin` outputs are less precise that `d4Max`'s, we could not generate percentages as in the previous experiment, but we still can easily observe that `d4Max` and `merlin` have different fields of expertise: `d4Max` was not able to draw near the solution, reaching, at most, 10% of the optimal weight model counting.

6 Conclusion and Future Work

In this paper, we defined the WeightedMax#SAT problem, which generalizes both Max#SAT and functional E-MajSAT problems. We presented an exact algorithm `d4Max`, a top-down tree-search procedure based on the sequential counter `d4` that computes the assignment, to a given set of variables, which maximizes the weighted projected model count. Additionally, we

proposed an alternative algorithm in order to make `d4Max` an anytime WeightedMax#SAT solver. In our experiments, we first compared `d4Max` with the state-of-the-art inference tool `merlin` on the MMAP query on Bayesian and Markov networks. Results shown that `d4Max`, although not dedicated to this problem, is quite competitive and may be more scalable than `merlin`. Additionally to the MMAP instances, we also considered exist-random SSAT benchmarks to evaluate the effectiveness of `d4Max` against state-of-the-art SSAT solvers. Our results clearly demonstrated the superiority of `d4Max` on the considered benchmarks. We also shown how `d4Max` benefits from having an anytime approach.

As a next step, we plan to evaluate `d4Max` on other applications in areas such as probabilistic programming, explainable AI and probabilistic model checking. For the anytime version of `d4Max`, we plan to investigate dedicated branching heuristics that could help converge more rapidly towards an optimal solution. An option would be to leverage Multi-Armed bandit techniques exploiting policy, such that EXP3 [5], UBC1 [4] or Thompson Sampling [49], to estimate the best branch to explore. It would be also interesting to take advantage of the best of the works presented in [48, 23, 60, 62], i.e., to exploit upper bounds that could be computed approximately, in order to prune some parts of the search space.

References

1 *Handbook of Satisfiability*, volume 185, 2009.

2 Udi Apsel and Ronen I. Brafman. Lifted MEU by weighted model counting. In *Proceedings of AAAI'12*, 2012.

3 Gilles Audemard, Jean-Marie Lagniez, Marie Miceli, and Olivier Roussel. Identifying soft core in propositional formulae. In *Proceedings of ICAART'22*, 2022.

4 Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, 2002.

5 Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, 2002.

6 Rehan Abdul Aziz, Geoffrey Chu, Christian J. Muise, and Peter J. Stuckey. #∃sat: Projected model counting. In *Proceedings of SAT'15*, volume 9340, pages 121–137, 2015.

7 Anicet Bart, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. Symmetry-driven decision diagrams for knowledge compilation. In *Proceedings of ECAI'14*, volume 263, pages 51–56, 2014.

8 Anicet Bart, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. An improved CNF encoding scheme for probabilistic inference. In *Proceedings of ECAI'16*, volume 285, pages 613–621, 2016.

9 Elazar Birnbaum and Eliezer L. Lozinskii. The good old davis-putnam procedure helps counting models. *J. Artif. Intell. Res.*, 10:457–477, 1999.

10 Florent Capelli, Jean-Marie Lagniez, and Pierre Marquis. Certifying top-down decision-dnnf compilers. In *Proceedings of AAAI'21*, pages 6244–6253, 2021.

11 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In *Proceedings of CP'13*, volume 8124, pages 200–216, 2013.

12 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proceedings of IJCAI'16*, pages 3569–3576, 2016.

13 Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008.

14 Mark Chavira, Adnan Darwiche, and Manfred Jaeger. Compiling relational bayesian networks for exact inference. *Int. J. Approx. Reason.*, 42(1-2):4–20, 2006.

15 Pei-Wei Chen, Yu-Ching Huang, and Jie-Hong R. Jiang. A sharp leap from quantified boolean formula to stochastic boolean satisfiability solving. In *Proceedings of AAAI'21*, pages 3697–3706, 2021.

16 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001.

17 Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *Proceedings of IJCAI'11*, pages 819–826, 2011.

18 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.

19 Carmel Domshlak and Jörg Hoffmann. Fast probabilistic planning through weighted model counting. In *Proceedings of ICAPS'06*, pages 243–252, 2006.

20 Jeffrey M. Dudek, Vu H. N. Phan, and Moshe Y. Vardi. Procount: Weighted projected model counting with graded project-join trees. In *Proceedings of SAT'21*, volume 12831, pages 152–170, 2021.

21 Leonardo Dueñas-Osorio, Kuldeep S. Meel, Roger Paredes, and Moshe Y. Vardi. Counting-based reliability estimation for power-transmission grids. In *Proceedings of AAAI'17*, pages 4488–4494, 2017.

22 Linus Feiten, Matthias Sauer, Tobias Schubert, Alexander Czutro, Eberhard Böhl, Ilia Polian, and Bernd Becker. #sat-based vulnerability analysis of security components - A case study. In *Proceedings of DFT'12*, pages 49–54, 2012.

23 Daniel J. Fremont, Markus N. Rabe, and Sanjit A. Seshia. Maximum model counting. In *Proceedings of AAAI'17*, pages 3885–3892, 2017.

24 John Gill. Computational complexity of probabilistic turing machines. *SIAM J. Comput.*, 6(4):675–695, 1977.

25 Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Near-uniform sampling of combinatorial spaces using XOR constraints. In *Proceedings of NIPS'06*, pages 481–488, 2006.

26 Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In *Handbook of Satisfiability*, volume 185, pages 633–654. IOS Press, 2009.

27 Roberto J. Bayardo Jr. and Joseph Daniel Pehoushek. Counting models using connected components. In *Proceedings of IAAI'00*, pages 157–162, 2000.

28 Kalev Kask, Rina Dechter, Javier Larrosa, and Fabio Cozman. Bucket-tree elimination for automated reasoning, 2001.

29 Vladimir Klebanov, Norbert Manthey, and Christian J. Muise. Sat-based analysis and quantification of information flow in programs. In *Proceedings of QEST'13*, volume 8054 of *Lecture Notes in Computer Science*, pages 177–192, 2013.

30 Tuukka Korhonen and Matti Järvisalo. Integrating tree decompositions into decision heuristics of propositional model counters (short paper). In *Proceedings of CP'21*, volume 210, pages 8:1–8:11, 2021.

31 Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Knowledge compilation for model counting: Affine decision trees. In *Proceedings of IJCAI'03*, pages 947–953, 2013.

32 Alexander V. Kozlov and Jaswinder Pal Singh. Computational complexity reduction for BN2O networks using similarity of states. In *Proceedings of UAI'96*, pages 357–364, 1996.

33 Jean-Marie Lagniez and Pierre Marquis. An improved decision-dnnf compiler. In *Proceedings of IJCAI'17*, pages 667–673, 2017.

34 Jean-Marie Lagniez and Pierre Marquis. A recursive algorithm for projected model counting. In *Proceedings of AAAI'19*, pages 1536–1543, 2019.

35 Nian-Ze Lee, Yen-Shi Wang, and Jie-Hong R. Jiang. Solving exist-random quantified stochastic boolean satisfiability via clause selection. In Jérôme Lang, editor, *Proceedings IJCAI'18*, pages 1339–1345, 2018.

36 Wei Li, Pascal Poupart, and Peter van Beek. Exploiting structure in weighted model counting approaches to probabilistic inference. *J. Artif. Intell. Res.*, 40:729–765, 2011.

37 Fangzhen Lin and Pierre Marquis. Causal theories of action: A computational core. In *Proceedings of IJCAI'03*, pages 1073–1078, 2003.

38 Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *CoRR*, 9:1–36, 1998.

39 Michael L. Littman, Stephen M. Majercik, and Toniann Pitassi. Stochastic boolean satisfiability. *J. Autom. Reason.*, 27(3):251–296, 2001.

40 Sibylle Möhle and Armin Biere. Dualizing projected model counting. In *Proceedings of ICTAI'18*, pages 702–709, 2018.

41 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC'01*, pages 530–535, 2001.

42 Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-dnnf compilation with sharpsat. In *Proceedings of AI'12*, volume 7310, pages 356–361, 2012.

43 Nina Narodytska, Aditya A. Shrotri, Kuldeep S. Meel, Alexey Ignatiev, and João Marques-Silva. Assessing heuristic machine learning explanations with model counting. In *Proceedings of SAT'19*, volume 11628 of *Lecture Notes in Computer Science*, pages 267–278, 2019.

44 Héctor Palacios, Blai Bonet, Adnan Darwiche, and Hector Geffner. Pruning conformant plans by counting models on compiled d-dnnf representations. In *Proceedings of ICAPS'05*, pages 141–150, 2005.

45 James D. Park. MAP complexity results and approximation methods. In *Proceedings of UAI'02*, pages 388–396, 2002.

46 James D. Park and Adnan Darwiche. Solving MAP exactly using systematic search. In *Proceedings of UAI'03*, pages 459–468, 2003.

47 James D. Park and Adnan Darwiche. Complexity results and approximation strategies for MAP explanations. *J. Artif. Intell. Res.*, 21:101–133, 2004.

48 Knot Pipatsrisawat and Adnan Darwiche. A new d-dnnf-based bound computation algorithm for functional E-MAJSAT. In *Proceedings of IJCAI'09*, pages 590–595, 2009.

49 Daniel Russo and Benjamin Van Roy. An information-theoretic analysis of thompson sampling. *J. Mach. Learn. Res.*, 17:68:1–68:30, 2016.

50 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

51 Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *Proceedings of SAT'04*, 2004.

52 Tian Sang, Paul Beame, and Henry A. Kautz. Performing bayesian inference by weighted model counting. In *Proceedings of AAAI'05*, pages 475–482, 2005.

53 Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In *Proceedings of IJCAI'19*, pages 1169–1176, 2019.

54 Mate Soos and Kuldeep S. Meel. BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *Proceedings of AAAI'19*, pages 1592–1599, 2019.

55 Larry J. Stockmeyer. The complexity of approximate counting (preliminary version). In *Proceedings of STOC'83*, pages 118–126, 1983.

56 Marc Thurley. Sharpsat - counting models with advanced component caching and implicit BCP. In *Proceedings of SAT'06*, volume 4121, pages 424–429, 2006.

57 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

58 Jacobo Torán. Complexity classes defined by counting quantifiers. *J. ACM*, 38(3):753–774, 1991.

59 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

60 Clément Viricel, David Simoncini, Sophie Barbe, and Thomas Schiex. Guaranteed weighted counting for affinity computation: Beyond determinism and structure. In Michel Rueher, editor, *Proceedings ofCP'16*, volume 9892 of *Lecture Notes in Computer Science*, pages 733–750. Springer, 2016.

61 Stephan Wäldchen, Jan MacDonald, Sascha Hauch, and Gitta Kutyniok. The computational complexity of understanding binary classifier decisions. *J. Artif. Intell. Res.*, 70:351–387, 2021.

62 Jiong Yang, Supratik Chakraborty, and Kuldeep S. Meel. Projected model counting: Beyond independent support. *CoRR*, abs/2110.09171, 2021.

63 Erik Peter Zawadzki, André Platzer, and Geoffrey J. Gordon. A generalization of SAT and #sat for robust policy evaluation. In *Proceedings of IJCAI'13*, pages 2583–2590, 2013.

64 Hantao Zhang and Mark E. Stickel. An efficient algorithm for unit propagation. In *Proceedings of AI-MATH'96*, pages 166–169, 1996.