

Approximate weighted model integration on DNF structures[☆]

Ralph Abboud^{*}, İsmail İlkan Ceylan, Radoslav Dimitrov

Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK



ARTICLE INFO

Article history:

Received 5 August 2021

Received in revised form 9 May 2022

Accepted 27 June 2022

Available online 30 June 2022

Keywords:

Weighted model integration

Weighted model counting

Approximations

Probabilistic inference

ABSTRACT

Weighted model counting consists of computing the weighted sum of all satisfying assignments of a propositional formula. Weighted model counting is well-known to be #P-hard for exact solving, but admits a fully polynomial randomized approximation scheme when restricted to DNF structures. In this work, we study *weighted model integration*, a generalization of weighted model counting which involves real variables in addition to propositional variables, and pose the following question: Does weighted model integration on DNF structures admit a fully polynomial randomized approximation scheme? Building on classical results from approximate weighted model counting and approximate volume computation, we show that weighted model integration on DNF structures can indeed be approximated for a class of weight functions. Our approximation algorithm is based on three subroutines, each of which can be a *weak* (i.e., approximate), or a *strong* (i.e., exact) oracle, and in all cases, comes along with accuracy guarantees. We experimentally verify our approach over randomly generated DNF instances of varying sizes, and show that our algorithm scales to large problem instances, involving up to 1K variables, which are currently out of reach for existing, general-purpose weighted model integration solvers.

© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Model counting (MC) is the task of counting the number of satisfying truth assignments (i.e., models) of a given propositional formula, and *weighted model counting* (WMC) generalizes this task by incorporating a *weight distribution* over the set of truth assignments. Specifically, given a propositional formula, and a weight function that assigns every truth assignment a weight, WMC is the problem of computing the total weight of the models of the input formula [1]. Typically, WMC is studied with weight functions, where the weight of each assignment factorizes into the weights of the individual variable assignments. WMC plays a very fundamental role in artificial intelligence, as it provides a unifying approach for encoding and solving different probabilistic inference problems that arise in various contexts, including *probabilistic graphical models* [2], *probabilistic planning* [3], *probabilistic logic programming* [4], *probabilistic databases* [5], and *probabilistic knowledge bases* [6].

Despite its wide applicability in artificial intelligence, WMC can only capture problems in discrete domains, since the whole formulation is based on *Boolean variables*. As a result, WMC cannot directly capture domains involving *real variables*, which puts even simple inference tasks in partly (or, fully) continuous domains, such as analyzing the probability of discrete

[☆] This paper is an invited revision of a paper which first appeared at the 2020 International Conference on Principles of Knowledge Representation and Reasoning (KR-20).

^{*} Corresponding author.

E-mail addresses: ralph.abboud@cs.ox.ac.uk (R. Abboud), ismail.ceylan@cs.ox.ac.uk (İ. Ceylan), radoslav.dimitrov@cs.ox.ac.uk (R. Dimitrov).

signals under Gaussian noise, beyond the scope of WMC. This observation has motivated the study of a generalization of WMC, known as *weighted model integration* (WMI) [7,8], which supports probabilistic inference in *hybrid domains* that consist of a mixture of discrete and real variables, and interactions among them.

The generalization of WMC to WMI is inspired by the generalization of the classical *satisfiability problem* (SAT) to *satisfiability modulo theories* (SMT) [9]. While SAT is limited to discrete domains, SMT allows to reason about the satisfiability of formulas involving, e.g., linear constraints over reals. This is realized through an SMT formula, which additionally contains real variables, and allows arithmetic statements (e.g., arithmetic (in)equalities) over these variables. Building on the foundations of SMT [9], WMI can be defined in a natural way: given an SMT formula, and a weight function that defines a *density* for every truth assignment of the formula, WMI is the problem of computing the sum of integrals over the densities of all the satisfying assignments of the SMT formula [8]. In other words, WMI replaces the propositional formula from WMC with an SMT formula, and accordingly asks to compute a sum of integrals over weight functions defined over the satisfying assignments of the SMT formula.

One way of obtaining special classes of WMI is through appropriately restricting the class of input formulas, i.e., by only allowing formulas in *conjunctive normal form* (CNF), or by only allowing formulas in *disjunctive normal form* (DNF). In fact, WMC is widely studied in the literature relative to both CNF and DNF structures. For example, marginal inference in Bayesian networks [10] can be reduced to WMC, by encoding the conditional dependencies from the network through a propositional formula in CNF [11,12], while, e.g., conjunctive query evaluation on probabilistic databases can be reduced to WMC, by computing the lineage representation of the input query, which yields a propositional formula in DNF [5]. The standard formulation of WMI assumes a formula in CNF as input, and to date, there is no study of WMI which is specifically tailored to formulas in DNF. This is surprising, because both variants relate to different inference problems which occur in various probabilistic data models.

In what follows, we write $\text{WMI}(\text{CNF})$ and $\text{WMI}(\text{DNF})$ to distinguish between special cases of WMI, and follow a similar convention for WMC. These problems are obviously $\#P$ -hard for exact solving, as is model counting over CNF and DNF structures [13]. For approximate solving, however, there is a strong contrast in computational complexity between variants of weighted model counting problems: $\text{WMC}(\text{DNF})$ has a *fully polynomial randomized approximation scheme* (FPRAS) due to Karp, Luby and Madras [14], producing polynomial-time approximations with guarantees, whereas $\text{WMC}(\text{CNF})$ is known to be NP-hard to approximate, provided that $\text{RP} \neq \text{NP}$ [15]. The latter polynomial-time inapproximability result immediately propagates to $\text{WMI}(\text{CNF})$, while the approximability status of $\text{WMI}(\text{DNF})$ remains *open*. In this paper, we are interested in answering the following question: Does $\text{WMI}(\text{DNF})$ admit an FPRAS?

We answer this question in the affirmative, and provide an FPRAS for $\text{WMI}(\text{DNF})$ with probabilistic accuracy guarantees for the case of *concave* weight functions. This result implies that WMI over DNF can be tractably approximated for a rich class of input problems. The intuition behind this result is based on two observations, corresponding to special cases of $\text{WMI}(\text{DNF})$. First, $\text{WMC}(\text{DNF})$, a special case of $\text{WMI}(\text{DNF})$ with no continuous variables, admits an FPRAS [14]. Second, the special case of $\text{WMI}(\text{DNF})$ with constant weight functions, and without any Boolean variables, corresponds to computing the *volume of unions of convex bodies*, and also admits an FPRAS [16]. Both of these approaches are based on the linear time coverage algorithm [17], and are instrumental for our result.

Our algorithm builds on these classical algorithms, and extends them, by allowing extra constructs essential for WMI, while preserving the approximation guarantees. More concretely, our algorithm jointly handles Boolean and real variables, and uses a natural encoding of SMT constraints and a concave weight function. Essentially, our algorithm views DNF clauses analogously to convex bodies in volume computation [16], and handles Boolean variables analogously to Karp, Luby and Madras [14], all while incorporating the weight function via weighted sampling techniques [18]. Overall, this algorithm preserves the approximation guarantees provided by both approaches, and enables tractable approximations for $\text{WMI}(\text{DNF})$. Our main contributions can be summarized as follows:

- We propose an efficient approximation algorithm for $\text{WMI}(\text{DNF})$, called APPROXWMI , extending the algorithm of Bringmann and Friedrich [16]. Our algorithm builds on the linear coverage algorithm [17], and incorporates Boolean variables and weight functions through dedicated sampling procedures.
- We prove that APPROXWMI is an FPRAS when the weight functions for $\text{WMI}(\text{DNF})$ are *concave*, and can be factorized into product of a real weight function and the weights of propositional literals, a common assumption in the literature. We further provide asymptotic bounds for the running time of APPROXWMI .
- We then relax the weight factorization assumption, and propose an extended FPRAS, APPROXWMI_D . APPROXWMI_D runs in polynomial time over concave weight functions, and naturally allows dependencies between Boolean and real variables in the weight function through a more refined sampling approach. As with APPROXWMI , we provide asymptotic bounds for the running time of APPROXWMI_D .
- We experimentally verify our approach, using an exact oracle for computing the volume of a body. Our experiments suggest that APPROXWMI efficiently solves large problem instances, including up to 1K variables, which are out of reach for any existing general-purpose WMI solver.

This work is an extended version of a conference paper which appeared in KR 2020 [19], containing all technical preliminaries, full proof details, and additional experiments. The rest of the paper is organized as follows. Section 2 is dedicated to preliminaries, where we formally introduce our context and the problem of weighted model integration. In Section 3, we

introduce APPROXWMI and prove the main results of this paper. We empirically verify APPROXWMI in Section 4. We review the related work in Section 5, and conclude with a summary and discussions for future work in Section 6.

2. Preliminaries

We briefly recall *propositional logic*, *satisfiability modulo theories* and *weighted model integration* and introduce the notation and the assumptions used throughout this work.

2.1. Propositional logic, satisfiability, and weighted model counting

Let \mathbf{V} be a (finite) set of Boolean variables (or, equivalently Boolean atoms). A *literal* is of the form p , or $\neg p$, where $p \in \mathbf{V}$. A *conjunctive clause* is a conjunction of literals, and a *disjunctive clause* is a disjunction of literals. A clause has *width* k if it has exactly k literals. A formula ϕ is in *conjunctive normal form* (CNF) if it is a conjunction of disjunctive clauses, and it is in *disjunctive normal form* (DNF) if it is a disjunction of conjunctive clauses. We say that a DNF (resp., CNF) has width k if it contains clauses of width at most k . For instance, the formula $(p_1 \wedge p_2 \wedge \neg p_3) \vee (\neg p_1 \wedge p_4)$ is a DNF with width 3.

An assignment $\nu : \mathbf{V} \mapsto \mathbb{B}$ maps every variable to either 0 (false), or 1 (true). An assignment ν *satisfies* a propositional formula ϕ , denoted $\nu \models \phi$, in the usual sense, where \models is the propositional entailment relation. The *satisfiability problem* (SAT) is the task of deciding whether a given propositional formula ϕ has a satisfying assignment. Given a propositional formula ϕ , its *model count*, denoted $\#\phi$, is the number of assignments ν satisfying ϕ . The *weighted model count* (WMC) of ϕ is defined as:

$$\text{WMC}(\phi) = \sum_{\nu \models \phi} w(\nu),$$

where $w : \mathcal{A} \mapsto \mathbb{R}$ is a *weight function*, and \mathcal{A} is the set of all possible assignments. We denote by $\text{WMC}(\text{CNF})$ and $\text{WMC}(\text{DNF})$ the special cases of WMC, where the class of input formulas is restricted to CNF and DNF, respectively.

2.2. Satisfiability modulo theories

The research field concerned with the satisfiability of formulas with respect to a background theory is called *satisfiability modulo theories* (SMT) [1]. Our context in this work is SMT on quantifier-free formulas in the theory of *linear real arithmetic* (LRA).

In this context, we consider a set \mathbf{X} of real variables in addition to the set \mathbf{V} of Boolean variables. An *LRA atom* is defined over the real variables and is of the form:

$$\sum_i \gamma_i x_i \bowtie \gamma,$$

where γ and γ_i are constant rational values, $x_i \in \mathbf{X}$, and $\bowtie \in \{<, \leq, >, \geq, =, \neq\}$ with their usual semantics. LRA atoms define constraints over the real domain. For example, the LRA atoms $x_1 + x_2 \leq 4$, $x_1 \geq 0$, $x_2 \geq 0$ together restrict the real variables x_1 and x_2 to a two-dimensional triangle lying in the fully positive quadrant of \mathbb{R}^2 .

We write $\text{atoms}(\mathbf{X}, \mathbf{V})$ to denote the set of LRA or Boolean atoms over their respective domains \mathbf{X} and \mathbf{V} . As in propositional logic, a literal is either an atom or its negation. We sometimes write *LRA literal*, or a *Boolean literal*, to distinguish the literals depending on the domain of their corresponding variables.

An SMT formula ϕ is a propositional formula over \mathbf{X} and \mathbf{V} , which is defined as a Boolean combination of literals via the logical connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ in the usual sense. We sometimes write $\phi(\mathbf{X}, \mathbf{V})$ to denote a propositional formula ϕ over \mathbf{X} and \mathbf{V} . Note that the CNF or DNF fragments of such formulas are defined as before. If $\mathbf{V} = \emptyset$, then the domain is fully continuous, and we say that ϕ is an *LRA formula*. Otherwise, if $\mathbf{X} = \emptyset$ then ϕ corresponds to a standard propositional formula.

Example 1. Consider the following formula, defined over $\mathbf{X} = \{x_1, x_2\}$ and $\mathbf{V} = \{p_1, p_2\}$:

$$\phi_{ex} = ((0 \leq x_1 \leq 5) \vee \neg p_1) \wedge (p_2 \vee \neg(10 \leq x_1 + x_2 \leq 15)).$$

This formula is defined using 2 Boolean and 2 LRA literals. If we drop p_1 and p_2 , we obtain a formula over only LRA constraints.

Analogously to the standard Boolean setting, a *truth assignment* $\nu : \text{atoms}(\mathbf{X}, \mathbf{V}) \mapsto \mathbb{B}$ for a propositional formula ϕ maps every atom to either 0 (false), or 1 (true), and *propositionally satisfies* ϕ in the usual sense. However, truth assignments over $\text{atoms}(\mathbf{X}, \mathbf{V})$ introduce a subtle difference relative to the standard definition. In particular, a propositional formula over \mathbf{X} and \mathbf{V} may have a propositionally satisfying truth assignment, but this assignment may yield inconsistency (i.e., infeasibility) with respect to the LRA constraints in the formula. We illustrate this with an example.

Example 2. Recall the formula ϕ_{ex} from Example 1, and consider the assignment ν , where

$$\nu(p_1) = 1, \quad \nu(p_2) = 0, \quad \nu(0 \leq x_1 \leq 5) = 1, \quad \nu(10 \leq x_1 + x_2 \leq 15) = 0.$$

It is easy to see that ϕ_{ex} is propositionally satisfiable. Observe that ν also satisfies the LRA constraints. Indeed, it implies that $0 \leq x_1 \leq 5$, and that either $x_1 + x_2 > 15$ or $x_1 + x_2 < 10$ holds. These constraints admit a solution, e.g., $x_1 = 2$, and $x_2 = 30$, so can be satisfied given ν . In contrast, consider the following formula:

$$\phi_{ex'} = (1 \leq x_1 + x_2 \leq 4) \wedge \neg p_1 \wedge (x_1 \leq -2) \wedge (x_2 \leq 2).$$

This formula is trivially propositionally satisfiable by setting all atoms to true, and this satisfying assignment is unique. However, this assignment yields inconsistent LRA constraints, since $(x_1 \leq -2)$, $(x_2 \leq 2)$ and $(1 \leq x_1 + x_2 \leq 4)$ cannot be satisfied simultaneously, i.e., they yield an empty solution set over \mathbb{R}^2 . Hence, the LRA constraints of $\phi_{ex'}$ cannot be satisfied by any propositionally satisfying truth assignment.

We now formally define the satisfiability of LRA atoms relative to a propositional truth assignment. A truth assignment ν is *LRA-satisfiable* if the solution space to the set of linear inequalities induced by ν is non-empty. That is, propositional formulas defined over \mathbf{X} and \mathbf{V} have two satisfiability criteria, namely conventional propositional satisfiability, and LRA satisfiability. In fact, the classical SMT problem over LRA is to determine, for a given propositional formula $\phi(\mathbf{X}, \mathbf{V})$, whether there exists an LRA-satisfiable assignment ν such that $\nu \models \phi$.

Observe that for any $\phi(\mathbf{X}, \mathbf{V})$, any choice of a real assignment \mathbf{x} to \mathbf{X} and a Boolean assignment \mathbf{v} to \mathbf{V} , together yields an assignment to ϕ (since this choice sets every atom in the universe to true or false). Hence, throughout the paper, we consider propositional formulas $\phi(\mathbf{X}, \mathbf{V})$ such that $\phi : (\mathbb{R}^{|\mathbf{X}|} \times \mathbb{B}^{|\mathbf{V}|}) \mapsto \mathbb{B}$, and write $\phi(\mathbf{x}, \mathbf{v}) \rightarrow 1$ to denote a satisfying assignment of ϕ .

2.3. Weighted model integration

We assume a set $\mathbf{X} \in \mathbb{R}^n$ of n real variables, and a set $\mathbf{V} \in \mathbb{R}^m$ of m Boolean variables. We consider propositional formulas $\phi(\mathbf{X}, \mathbf{V})$ such that $\phi : (\mathbb{R}^n \times \mathbb{B}^m) \mapsto \mathbb{B}$, and weight functions $w(\mathbf{X}, \mathbf{V})$ such that $w : (\mathbb{R}^n \times \mathbb{B}^m) \mapsto \mathbb{R}^+$.

Given a propositional formula ϕ and a weight function w over \mathbf{X} and \mathbf{V} , the *weighted model integral* of ϕ over w is defined as:

$$\text{WMI}(\phi, w \mid \mathbf{X}, \mathbf{V}) = \sum_{\mathbf{v} \in \mathbb{B}^m} \int_{\mathbf{x} \in \mathcal{X}_{\phi, \mathbf{v}}} w(\mathbf{x}, \mathbf{v}) d\mathbf{x}, \quad (1)$$

where $\mathcal{X}_{\phi, \mathbf{v}} = \{\mathbf{x} \in \mathbb{R}^n \mid \phi(\mathbf{x}, \mathbf{v}) \rightarrow 1\}$ denotes the set of *all* real valuations of \mathbf{X} satisfying $\phi(\mathbf{x}, \mathbf{v})$.

In this definition, the weight function w is very general, and can contain arbitrary dependencies between (and among) Boolean and real variables. Functions having such dependencies theoretically fall within the scope of WMI, but in practice are highly intractable, and thus are rarely adopted. Therefore, it is common to employ simplifying assumptions over w , so as to decompose and factorize it into more practically viable fragments. For WMC, a commonly used factorization is to assign every Boolean variable its own independent weight, and formulate $w(\mathbf{v})$ as:

$$w(\mathbf{v}) = \prod_{p \in \mathbf{V}} w_b(p),$$

where $w_b(p)$ is the weight of Boolean literal p . An analogous factorization is also used for WMI (cf. Section 2.1 of [20]), namely:

$$w(\mathbf{x}, \mathbf{v}) = w_x(\mathbf{x}) \prod_{p \in \mathbf{V}} w_b(p), \quad (2)$$

where $w_x : \mathbb{R}^n \rightarrow \mathbb{R}$. Thus, the weight function in Equation (1) is typically factorized into a product of a real variable weight function w_x and a product of individual Boolean literal weights, and we refer to this as the *factorization assumption*. In this work, we additionally assume that w_x and w_b all return *positive* values, that is $w_x : \mathbb{R}^n \rightarrow \mathbb{R}^+$ and $w_b : \mathbb{B} \rightarrow \mathbb{R}^+$. We make this restriction so as to align with volume computation (which necessarily returns a positive quantity) over real bodies, and to easily map the weight function over Boolean variables to a probability distribution to be used for sampling. Indeed, if the (positive) Boolean weights w_b assigned to each variable do not define a distribution, then a simple normalization of literal weights for each Boolean variable p is sufficient to recover a distribution. We now illustrate WMI computation under this factorization with an example.

Example 3. Consider the formula:

$$\phi_{\text{DNF}} = (p_1 \wedge (0 \leq x_1 \leq 5) \wedge \neg p_2) \vee (p_2 \wedge \neg(2 \leq x_1 \leq 4)),$$

where the range of x_1 is $0 \leq x_1 \leq 10$, and the weight function $w(\mathbf{x}, \mathbf{v}) = x_1 \cdot w_b(p_1) \cdot w_b(p_2)$ with $w_b(p_1) = 0.6$, and $w_b(p_2) = 0.1$. We can compute the weighted model integral over this formula and weight function as:

$$\begin{aligned} \text{WMI}(\phi_{\text{DNF}}, w \mid \mathbf{X}, \mathbf{V}) &= 0.6 \cdot (1 - 0.1) \cdot \int_0^5 x_1 dx_1 + \\ &0.6 \cdot 0.1 \cdot \left(\int_0^2 x_1 dx_1 + \int_4^{10} x_1 dx_1 \right) + \\ &(1 - 0.6) \cdot 0.1 \cdot \left(\int_0^2 x_1 dx_1 + \int_4^{10} x_1 dx_1 \right) = 11.15. \end{aligned}$$

Finally, weighted model integration is defined on fragments of propositional logic in the obvious way, i.e., WMI over DNF formulas (resp. CNF formulas) is the weighted model integration problem where the class of input formulas is restricted to formulas in DNF (resp., CNF).

2.4. Approximations with guarantees

Model counting problems are #P-hard [13] to solve exactly, and thus are intractable for exact computation. As a result, techniques for efficient *approximations* to model counting have been devised, a special class of which being *fully polynomial randomized approximation schemes* (FPRAS). Given a target error $0 < \epsilon < 1$ and confidence $0 < \delta < 1$, an FPRAS computes an approximation $\hat{\mu}$ of the actual solution μ , in polynomial time w.r.t. the input, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, such that

$$\Pr(\mu(1 - \epsilon) \leq \hat{\mu} \leq \mu(1 + \epsilon)) \geq 1 - \delta.$$

For WMC(DNF), the Karp, Luby, and Madras [14] algorithm (KLM), a special case of the Linear-Time Coverage (LTC) [17] algorithm, is an FPRAS. KLM is an iterative random sampling procedure, which samples assignments satisfying a random clause in the input DNF, and then verifies this assignment against another uniformly randomly selected clause, to compute an unbiased estimator of the overall model count. More specifically, for a DNF ϕ with m (Boolean) variables and k clauses, KLM runs $T = 8(1 + \epsilon)k \log(\frac{2}{\delta}) \frac{1}{\epsilon^2}$ sampling iterations, to compute a successful trial count N , from which the estimator is produced. At every trial, KLM performs the following steps:

1. **(Sampling)** If no current sample assignment τ exists, then a random clause c_i is selected with probability $\Pr(c_i) / \sum_{j=1}^k \Pr(c_j)$ (i.e., proportionally to its probability), where $\Pr(c_i) = \prod_{p \in c_i} w_b(p)$. Afterwards, τ is sampled uniformly from the set of satisfying assignments for c_i . More concretely, the variables appearing in c_i are all set to make c_i true, and all remaining variables are sampled using the weight function w . If τ already exists, then proceed to Step 2.
2. **(Verification)** Another clause c' (which could be identical to c_i) is *uniformly* randomly sampled with probability $\frac{1}{k}$, and τ is checked against c' . If $\tau \models c'$, N is incremented and τ is re-sampled. Otherwise, the sampled assignment τ is preserved, and re-used in the next trial.

KLM returns $T \sum_{j=1}^k \Pr(c_j) / kN$ as an estimate for WMC(DNF). Informally, this estimator is designed to count every distinct satisfying assignment exactly once in expectation, and proportionally to its weight. This is done by making assignments simultaneously satisfying multiple clauses, which are likelier to be sampled in Step 1, increase N more frequently. This lowers the overall estimator, and levels out the contribution of such assignments relative to that of less prominent assignments. In terms of running time, assignment checking in Step 2, as well as sampling in Step 1, run in $O(m)$. Therefore, KLM runs in time $O(mT) = O(mk\epsilon^{-2} \log(\frac{1}{\delta}))$.

In this work, we build on an FPRAS for volume computation over convex bodies [21]. For this problem, deterministic solving is highly intractable. Indeed, given an n -dimensional real domain \mathbb{R}^n , it is shown that no deterministic algorithm with a running time of $O(n^{cn})$, $c < \frac{1}{2}$, can even approximate the volume of convex bodies [22], which has led to the development of randomized alternatives. Randomization has proven key for producing polynomial-time approximations: following an initial breakthrough FPRAS [23], which has prohibitive practical computational complexity, several more efficient FPRAS algorithms have been proposed. For this paper, we refer to the FPRAS by Lovász and Vempala [21], which currently has the lowest computational complexity for the volume computation task. Given the highly involved nature of this FPRAS, we limit ourselves to a high-level overview of the algorithm, and refer readers to the original paper for a detailed presentation.

The Lovász and Vempala FPRAS is based on Multi-phase Monte Carlo. At every phase, the FPRAS approximates a ratio between the volume of two convex bodies, where one body is contained in the other. Phases start with large, simpler bodies, and in subsequent phases, volume ratios are approximately computed between ever smaller, more complex bodies, until the target convex body is reached in the final phase. For an n -dimensional space, the Lovász and Vempala FPRAS uses

$O^*(n)$ phases, where the asterisk denotes suppressed logarithmic factors, and at every phase, random walk algorithms such as *hit-and-run* [18] are called over the convex bodies to approximate the ratio between their volumes. Since *hit-and-run* runs in $O^*(n^3)$, the overall volume computation FPRAS runs in time $O^*(n^4)$.

3. Approximate weighted model integration

In this section, we propose APPROXWMI, an algorithm for WMI(DNF), and prove that it is an FPRAS given a concave weight function w formulated using the factorization assumption. APPROXWMI builds on APPROXUNION, an FPRAS for approximately computing the volume of unions of convex bodies [16] which is based on LTC [17]. We first introduce APPROXUNION.

3.1. APPROXUNION: computing the volume of unions of convex bodies

APPROXUNION is an FPRAS for computing the volume of the union of convex bodies. More formally, given k convex bodies B_1, \dots, B_k , APPROXUNION returns an approximation of the volume of $\bigcup_{i=1}^k B_i$, denoted $\text{Vol}(\bigcup_{i=1}^k B_i)$ with confidence $1 - \delta$ and error ϵ . Fundamentally, APPROXUNION is based on LTC, similarly to KLM: it uses an iterative sampling procedure, consisting of (i) sampling points and (ii) verifying them against a uniformly sampled convex body. However, APPROXUNION additionally introduces an initial volume computation step, to estimate the volumes of all individual bodies, and allows errors in its computations. That is, volume computation, sampling and verification can return results with errors, so long as these errors can be made arbitrarily small. More specifically, APPROXUNION is an FPRAS if volume computation, sampling and verification are each an FPRAS. Hence, APPROXUNION tolerates the use of “weak” oracles producing approximate results, while maintaining probabilistic guarantees on its outputs. We now present APPROXUNION in detail.

Initially, APPROXUNION approximately computes the volume of every convex body, yielding estimates with multiplicative error ϵ_V relative to the exact body volume, using an oracle VOLUMEQUERY. Unlike KLM, where clause probabilities can be trivially computed, volume computation is a costly operation, which requires a dedicated FPRAS to be approximated in polynomial time. Once all approximate convex body volumes are computed, APPROXUNION then performs T iterative sampling procedure steps to compute an estimator for the volume of the union of convex bodies. An APPROXUNION sampling iteration is as follows:

1. **(Sampling)** If no sample point τ exists, sample a body B_i with probability $\text{Vol}(B_i) / \sum_{j=1}^k \text{Vol}(B_j)$, and then approximately uniformly sample τ from B_i using an oracle SAMPLEQUERY with error ϵ_S . Otherwise, if τ already exists, proceed to Step 2.
2. **(Verification)** Check whether τ belongs to another uniformly randomly chosen body B' using another approximate oracle, POINTQUERY, with error ϵ_P . If $\tau \in B'$, a new point τ is sampled in Step 1, and N is incremented. Otherwise, τ is re-used in the next trial.

We observe two main differences relative to KLM. First, we note that sampling points in a convex body differ from assignment sampling for KLM clauses, in that (i) it is uniform across points in convex bodies, so does not assign distinct weights to different assignments and (ii) requires a dedicated query SAMPLEQUERY with its own error ϵ_S to perform sampling. Uniformity stems from the nature of volume computation, since all regions of a body contribute identically to its overall volume, whereas the need for a dedicated SAMPLEQUERY is due to the complexity of point sampling in an arbitrary convex body. Indeed, given the factorization assumption, assignment sampling in KLM can be trivially performed through Bernoulli trials, whereas uniform point sampling in a convex body requires advanced subroutines in the general case.

The second main difference in APPROXUNION relative to KLM is the complexity of membership verification. More specifically, APPROXUNION verifies membership of a point relative to a convex body, which is non-trivial for general convex bodies, whereas KLM simply matches Boolean variables to target values.

Following T sampling iterations, APPROXUNION returns $T \sum_{j=1}^k \text{Vol}(B_j) / kN$ as an estimate of $\text{Vol}(\bigcup_{i=1}^k B_i)$. APPROXUNION is an FPRAS for the volume computation of a union of convex bodies under certain conditions, as stated in Theorem 2 of that work [16], and these conditions are satisfied with closed-form bounds from Lemma 3 of the same work. More concretely, the following result holds:

Theorem 1 (Theorem 2 and Lemma 3, [16]). APPROXUNION relative to oracles VOLUMEQUERY, SAMPLEQUERY, POINTQUERY with errors ϵ_V , ϵ_S , and ϵ_P respectively, is an FPRAS for $\text{Vol}(\bigcup_{i=1}^k B_i)$ with error ϵ and confidence $\frac{1}{4}$ using $T = \frac{24 \ln(2)(1+\tilde{\epsilon})k}{\tilde{\epsilon}^2 - 8(\tilde{C}-1)k}$ iterations, with $\tilde{\epsilon} = \frac{\epsilon - \epsilon_V}{1 + \epsilon_V}$ and $\tilde{C} = \frac{(1+\epsilon_S)(1+\epsilon_V)(1+k\epsilon_P)}{(1-\epsilon_V)(1-\epsilon_P)}$, for ϵ_V , $\epsilon_S \leq \frac{\epsilon^2}{47k}$, and $\epsilon_P \leq \frac{\epsilon^2}{47k^2}$.

Furthermore, when this theorem holds, T is $O(\frac{k}{\epsilon^2})$. In summary, APPROXUNION generalizes LTC to allow errors within sampling, membership checking, and volume computation, so long as these can be made arbitrarily small, and computes unions of continuous sets. However, APPROXUNION does *not* allow for confidence parameters in the oracles, but this can be trivially extended to allow for FPRAS oracles having confidence δ , as stated earlier, using standard tools of probability, as we now show in the following corollary.

Corollary 2. APPROXUNION relative to FPRAS oracles VOLUMEQUERY, SAMPLEQUERY, and POINTQUERY with errors $\epsilon_V, \epsilon_S, \epsilon_P$ and confidence values $\delta_V, \delta_S, \delta_P$, respectively, is an FPRAS with error ϵ and confidence δ for $\text{Vol}(\bigcup_{i=1}^k B_i)$ using $T = \frac{8 \ln(\frac{8}{\delta})(1+\bar{\epsilon})k}{\bar{\epsilon}^2 - 8(\bar{\epsilon}-1)k}$ iterations, for

1. $\epsilon_V, \epsilon_S \leq \frac{\epsilon^2}{47k}, \epsilon_P \leq \frac{\epsilon^2}{47k^2}$, and
2. $\delta_V \leq \frac{\delta}{4k}, \delta_S + \delta_P \leq \frac{\delta}{2276 \ln(\frac{8}{\delta}) \frac{k}{\epsilon^2}}$.

Proof. We use the worst-case assumption that a single failure of any oracle, or a failure of the sampling procedure, implies the failure of APPROXUNION. Hence, we seek to upper-bound the union of these failure probabilities by δ to ensure that APPROXUNION is an FPRAS.

In Lemma 3 of [16], it is shown that Condition 1 is sufficient for reliable but weak oracles VOLUMEQUERY, SAMPLEQUERY, and POINTQUERY, to ensure APPROXUNION meets the ϵ error requirement with probability $\frac{3}{4}$. Hence, we now need to prove that, given unreliable oracles satisfying Condition 2, and failure probability of the LTC sampling procedure generalized from $\frac{1}{4}$ to a value $\delta_1 < \delta$, APPROXUNION also meets the confidence requirement δ , and is therefore an FPRAS for computing the union of convex bodies.

The generalization of the failure probability of sampling from $\frac{1}{4}$ to $\delta_1 < \delta$ can be achieved using standard probability amplification techniques, namely by multiplying the required number of trials T specified in Theorem 1 by $\frac{1}{3 \ln(2)} \ln(\frac{2}{\delta_1})$, yielding T as specified in Corollary 2. We can now upper-bound the failure probability of the LTC sampling procedure, denoted f_{LTC} , by setting $\delta_1 = \frac{\delta}{4}$, which yields the value of T shown in Corollary 2.

Given $\delta_1 = \frac{\delta}{4}$, a failure probability of $\frac{3\delta}{4}$ remains, and shall be allocated for all possible oracle failures. We now show that the confidence requirements stated in Condition 2 perform this allocation and indeed upper-bound any oracle failure probability by $\frac{3\delta}{4}$, as required:

Let f_V denote the failure of any call to VOLUMEQUERY, f_S the failure of any call to SAMPLEQUERY, and f_P the failure of any call to POINTQUERY. Furthermore, let f denote the overall failure probability of APPROXUNION. Given that VOLUMEQUERY is called k times within APPROXUNION, setting $\delta_V \leq \frac{\delta}{4k}$ yields, by the union bound:

$$\Pr(f_V) \leq k\delta_V = \frac{\delta}{4}.$$

Within APPROXUNION, POINTQUERY is called T times, whereas SAMPLEQUERY can be called up to T times, depending on the success of POINTQUERY at the previous iteration. We assume the worst-case and consider that SAMPLEQUERY is called T times. Applying the union bound with the bound on δ_P and δ_S as specified in Condition 2 yields:

$$\Pr(f_P \vee f_S) \leq T(\delta_S + \delta_P) \leq T \frac{\delta}{2276 \ln(\frac{8}{\delta}) \frac{k}{\epsilon^2}}.$$

We now use the bound from Lemma 3 in [16], which gives that, under Condition 1, $T < 2365 \frac{k}{\epsilon^2}$ for failure probability $\frac{1}{4}$. Generalizing this bound for an arbitrary δ_1 gives $T < \frac{2365}{3 \ln(2)} \ln(\frac{2}{\delta_1}) \frac{k}{\epsilon^2} < 1138 \ln(\frac{8}{\delta}) \frac{k}{\epsilon^2}$. Replacing T with this bound in the failure probability yields:

$$\Pr(f_P \vee f_S) \leq \frac{\delta}{2}.$$

Finally, using the union bound to upper-bound the overall APPROXUNION failure probability yields:

$$\Pr(f) \leq \Pr(f_V \vee f_P \vee f_S) + \Pr(f_{\text{LTC}}) \leq \frac{\delta}{4} + \frac{\delta}{2} + \frac{\delta}{4} = \delta,$$

as required. \square

3.2. APPROXWMI

We now introduce APPROXWMI (Algorithm 1), an FPRAS for WMI(DNF) given a *concave* and *factorized* weight function w . More precisely, a function w is concave if $\forall x, y \in \mathbb{R}^n$, and $\lambda \in [0, 1]$,

$$\lambda w(x) + (1 - \lambda)w(y) \leq w(\lambda x + (1 - \lambda)y),$$

and w is factorized when it can be formulated using the factorization assumption. Fundamentally, the factorization assumption simplifies the initial computation of clause weights, and subsequent sampling from formula clauses, whereas concavity ensures that the weight function restricts the scope of any volume computation in APPROXWMI only to convex bodies, enabling the use of the corresponding FPRAS algorithms. We now provide an overview of APPROXWMI, and show that it is an FPRAS for WMI(DNF).

Algorithm 1 APPROXWMI for WMI(DNF).

Input: \mathbf{X} : a set of n real variables; \mathbf{V} : a set of m Boolean variables; ϕ : a DNF consisting of k clauses c_i , $i \in \{1, \dots, k\}$; w : a concave factorized weight function.

Parameters: ϵ : error, δ : confidence.

Output: An (ϵ, δ) -approximation of $\text{WMI}(\phi, w \mid \mathbf{X}, \mathbf{V})$.

```

1:  $T \leftarrow \frac{8 \ln(\frac{\epsilon}{\delta})(1+\epsilon)k}{\epsilon^2 - 8(\epsilon-1)k}$  ▷  $T$  is  $O(\frac{k}{\epsilon^2} \ln(\frac{1}{\delta}))$  [16]
2:  $\delta_V \leftarrow \frac{\delta}{4k}, \delta_S \leftarrow \frac{\delta}{2276 \ln(\frac{8}{\epsilon}) \frac{k}{\epsilon^2}}$  ▷ Subroutine confidence parameters
3: for  $a \leftarrow 1$  to  $k$  do
4:    $U_a \leftarrow \text{CLAUSEWEIGHT}(c_a, w, \mathbf{X}, \mathbf{V})[\frac{\epsilon^2}{47k}, \delta_V]$  ▷ Clause weight computation
5:  $U \leftarrow \sum_{a=1}^k U_a$ 
6:  $\text{time} \leftarrow 0, N \leftarrow 0$ 
7: while  $\text{time} < T$  do ▷ Sampling trials
8:   Randomly select  $c_i, i \in \{1, \dots, k\}$  with probability  $\frac{U_i}{U}$ 
9:    $(\mathbf{x}, \mathbf{v}) \leftarrow \text{SAMPLE}(c_i, w, \mathbf{X}, \mathbf{V})[\frac{\epsilon^2}{47k}, \delta_S]$ 
10:   $c_{\text{sat}} \leftarrow \text{false}$ 
11:  while  $\neg c_{\text{sat}}$  do
12:    Uniformly select  $c_j$ , where  $j \in \{1, \dots, k\}$ 
13:     $\text{time} \leftarrow \text{time} + 1$ 
14:    if  $\text{time} \geq T$  then ▷ If  $T$  reached during trial
15:      return  $TU/kN$ 
16:    if  $\text{EVALUATE}(c_j, \mathbf{x}, \mathbf{v})$  then ▷ Membership checking
17:       $c_{\text{sat}} \leftarrow \text{true}, N \leftarrow N + 1$ 
18: return  $TU/kN$ 

```

Algorithm 2 Subroutines of APPROXWMI as functions CLAUSEWEIGHT, SAMPLE, and EVALUATE.

```

1: function CLAUSEWEIGHT( $c, w, \mathbf{X}, \mathbf{V}$ )[ $\epsilon, \delta$ ]
2:   $w_{\text{Bool}} \leftarrow \prod_{p \in \mathbf{v}_c} w_b(p)$  ▷ Boolean weight based on set  $\mathbf{v}_c$  of Boolean literals from  $c$ 
3:  Introduce a new variable  $d$  in  $\mathbf{X}$  ▷ Include weight function via auxiliary variable
4:   $\mathcal{X}'_c \leftarrow \mathcal{X}_c \wedge (\mathbf{x} \in \mathbb{R}^n, 0 \leq d \leq w_x(\mathbf{x}))$  ▷  $\mathcal{X}'_c$  is the convex polytope induced by the clause  $c$ 
5:   $w_{\text{Real}} \leftarrow \text{VOLUME}(\mathcal{X}'_c)[\epsilon, \delta]$  ▷ Resulting body volume automatically factors in  $w_x$ 
6:  return  $w_{\text{Bool}} \cdot w_{\text{Real}}$  ▷ Return product as clause weight
7:
8: function SAMPLE( $c, w, \mathbf{X}, \mathbf{V}$ )[ $\epsilon, \delta$ ]
9:   $\mathbf{v} \leftarrow \text{sample from } \mathbf{V} \text{ respecting } \mathbf{v}_c \text{ and } w_b$  ▷ Bernoulli Boolean sampling respecting  $\mathbf{v}_c$ 
10:  Introduce a new variable  $d$  in  $\mathbf{X}$  ▷ Auxiliary variable to reduce to uniform sampling
11:   $\mathcal{X}'_c \leftarrow \mathcal{X}_c \wedge (\mathbf{x} \in \mathbb{R}^n, 0 \leq d \leq w_x(\mathbf{x}))$ 
12:   $\mathbf{x} \leftarrow \text{CONVEXBODYSAMPLER}(\mathcal{X}'_c)[\epsilon, \delta]_{1:n}$  ▷ Sample from  $\mathcal{X}'_c$  and drop last dimension
13:  return  $(\mathbf{x}, \mathbf{v})$  ▷ Return the samples as output
14:
15: function EVALUATE( $c, \mathbf{x}, \mathbf{v}$ )
16:  Compute convex polytope  $\mathcal{X}_c$  from  $c$ 
17:  if  $\mathbf{x} \notin \mathcal{X}_c$  then return false ▷ Polytope membership
18:  if  $\mathbf{v} \models c$  then return false ▷ Boolean satisfiability
19:  return true

```

Overview of APPROXWMI. Given a DNF ϕ over \mathbf{X} and \mathbf{V} defined using propositional and LRA atoms, and a concave, factorized weight function w , APPROXWMI computes an (ϵ, δ) -approximation of $\text{WMI}(\phi, w \mid \mathbf{X}, \mathbf{V})$. APPROXWMI is based on APPROXUNION, and extends its oracle functions to allow unreliable oracles, hybrid domains, and factorized concave weight functions over convex bodies.

First, APPROXWMI sets essential parameters, namely the number of sampling iterations T (line 1) and confidence targets for subsequent clause weight computation (i.e., the WMI analog of volume computation in APPROXUNION) and point sampling operations (line 2) based on overall target error ϵ , confidence δ and the number of clauses k . Then, APPROXWMI computes the weighted model integral for every clause in ϕ given w , using the function CLAUSEWEIGHT (lines 3–4). CLAUSEWEIGHT is analogous to VOLUMEQUERY in APPROXUNION, but additionally considers the effect of w on the integral and supports discrete variables in \mathbf{V} . CLAUSEWEIGHT yields estimates of individual clause WMI with error $\frac{\epsilon^2}{47}$ and confidence δ_V .

Following calls to CLAUSEWEIGHT for all k clauses in ϕ , T sampling iterations are conducted to compute the LTC estimator (lines 7–18). To this end, the disjoint universe weight is initially computed (line 5), and the corresponding iteration counter time and success counter N are initialized (line 6). In a sampling trial, a random clause c_i is selected with probability proportional to its weight U_i , and then a point (\mathbf{x}, \mathbf{v}) is sampled from c_i according to w using the function SAMPLE (lines 8–9). Using the factorization assumption, SAMPLE independently samples Boolean (\mathbf{v}) and real (\mathbf{x}) variables satisfying c_i . Afterwards, another clause c_j (possibly c_i), is uniformly chosen from ϕ (line 12), and the point (\mathbf{x}, \mathbf{v}) is checked for membership to c_j via the function EVALUATE (line 16). EVALUATE separately validates Boolean and real components of the sampled point, and, in case of membership in both domains, the variable N is incremented (line 17).

We now explain the subroutines CLAUSEWEIGHT, SAMPLE, and EVALUATE in detail. The pseudo-codes for these subroutines are shown in Algorithm 2.

CLAUSEWEIGHT. This function returns an (ϵ, δ) -approximation of $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ for a given clause c and weight function w over the domains \mathbf{X}, \mathbf{V} . Owing to the factorization assumption, **CLAUSEWEIGHT** separately processes computation over \mathbf{X} and \mathbf{V} . For the Boolean sub-domain, **CLAUSEWEIGHT** computes w_{Bool} as the product of probabilities for all Boolean literals p appearing in c (line 2), analogously to the right-hand-side product described in Equation (2).

For the real sub-domain, **CLAUSEWEIGHT** computes the integral of w_x over the convex polytope defined by the LRA constraints of c . At a high level, this is achieved by introducing the concave weight function over the polytope as an additional convex dimension, which creates a new $n + 1$ -dimensional convex body (lines 3-4), and subsequently computing the volume of this new body using existing volume computation tools to yield the weighted integral (line 5).

More specifically, let \mathcal{X}_c denote the n -dimensional convex polytope induced by the LRA constraints of c . This body is fully determined by the LRA constraints and literals of c , but additionally covers the full domain of definition for all real variables not appearing in c . To compute the integral of w over \mathcal{X}_c , we introduce an additional $n + 1^{\text{th}}$ dimension, represented by dummy variable d , and constrain this dimension by limiting its value to the range $0 \leq d \leq w_x(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^n$. We denote the resulting body as \mathcal{X}'_c (line 3). Observe that the introduction of d reduces the current weighted integral formulation into a standard volume computation. Indeed, w is incorporated into \mathcal{X}'_c via d and its corresponding constraint, and it is clear that computing the integral of \mathcal{X}'_c corresponds to computing the weighted integral of \mathcal{X}_c given w . Hence, the weighted integral over \mathcal{X}_c can be computed as a standard integral over \mathcal{X}'_c .

We also observe that the additional constraint on d is convex, since w_x is concave. Thus, \mathcal{X}'_c is a convex body, since \mathcal{X}_c is a convex polytope. Note however, that \mathcal{X}'_c is not necessarily a polytope, as w_x can be an arbitrary concave function. In fact, \mathcal{X}'_c is a polytope if and only if w_x is a linear function. Nonetheless, the general convexity of \mathcal{X}'_c is sufficient, as it enables the computation of its volume using known FPRAS algorithms designed for convex bodies [21,24], represented by the function **VOLUME**. Therefore, the output of **VOLUME**, w_{Real} , corresponds to the weighted integral of \mathcal{X}_c given w . Finally, **CLAUSEWEIGHT** returns the product of lines 2 and 5 as its estimate for $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ (line 6).

SAMPLE. This function samples a point (\mathbf{x}, \mathbf{v}) over the domain $\mathbf{X} \cup \mathbf{V}$ from a given clause c and a weight function w . Based on the factorization assumption, it separately performs sampling over Boolean and real sub-domains. For the Boolean sub-domain, a Boolean assignment \mathbf{v} satisfying c is sampled by (i) setting all Boolean literals appearing in c to their required values and (ii) randomly sampling all remaining variables according to w_b (line 9), which corresponds to independent Bernoulli sampling for each variable based on its defined truth probability.

For the real sub-domain, an analogous transformation from \mathcal{X}_c to a convex body \mathcal{X}'_c as in **CLAUSEWEIGHT** (lines 10-11) is used to incorporate w and reduce the sampling problem to a uniform setting. Afterwards, **SAMPLE** samples a point approximately uniformly from \mathcal{X}'_c , using standard sampling approaches for convex bodies such as hit-and-run [18], denoted by **CONVEXBODYSAMPLER** (line 12). It then discards the $n + 1^{\text{th}}$ dimension of this point to yield an approximate sample \mathbf{x} from \mathcal{X}_c weighted according to w . Finally, **SAMPLE** returns the outputs of lines 9 and 12 as a sample from c (line 13).

EVALUATE. This function determines the membership of a point $(\mathbf{x}, \mathbf{v}) \in \mathbb{R}^n \times \mathbb{B}^m$ to a clause c . Specifically, it checks the membership of (\mathbf{x}, \mathbf{v}) to the polytope \mathcal{X}_c defined by c in two steps: **EVALUATE** first verifies the real component \mathbf{x} , i.e., that all the LRA constraints of c are satisfied (line 17), and then verifies the Boolean component \mathbf{v} (line 18). If both conditions are met, then (\mathbf{x}, \mathbf{v}) satisfies c . Unlike the earlier two functions, **EVALUATE** is deterministic, and its outputs have no attached uncertainty.

3.3. APPROXWMI is an FPRAS

We show the correctness of **APPROXWMI**, and prove that, under the error and confidence settings presented in Algorithm 1, it is an FPRAS for $\text{WMI}(\text{DNF})$ with concave weight functions w respecting the factorization assumption. As **APPROXWMI** builds on **APPROXUNION**, and replaces its oracles with specific **WMI** analogs for volume computation within a DNF conjunctive clause, point sampling and membership checking respectively, the proof shows that all three **APPROXWMI** oracles correctly estimate their target values and can all produce approximations within the error and confidence bounds derived in Corollary 2.

More concretely, the proof shows that **CLAUSEWEIGHT** correctly computes an (ϵ_V, δ_V) -approximation of $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$, and meets the conditions of Corollary 2. Then, the proof shows that **SAMPLE** is correct, by showing that uniform sampling from \mathcal{X}'_c and discarding the final dimension is equivalent to sampling from \mathcal{X}_c according to w , and establishes the sufficiency of running **CONVEXBODYSAMPLER** with parameters ϵ_S and δ_S to satisfy Corollary 2. Finally, the correctness and satisfaction of Corollary 2 for **EVALUATE** are trivially shown. We now formally state our result, and provide its complete proof.

Theorem 3. *APPROXWMI relative to FPRAS oracles **CLAUSEWEIGHT**, **SAMPLE**, and **EVALUATE** having error $\epsilon_V, \epsilon_S, \epsilon_P$ and confidence $\delta_V, \delta_S, \delta_P$, respectively, is an FPRAS for $\text{WMI}(\text{DNF})$ over a concave and factorized weight function w , with error ϵ and confidence δ and using $T = \frac{8 \ln(\frac{8}{\delta})(1+\epsilon)k}{\epsilon^2 - 8(\epsilon - 1)k}$ iterations, for*

1. $\epsilon_V, \epsilon_S \leq \frac{\epsilon^2}{47k}, \epsilon_P \leq \frac{\epsilon^2}{47k^2}$, and
2. $\delta_V \leq \frac{\delta}{4k}, \delta_S + \delta_P \leq \frac{\delta}{2276 \ln(\frac{8}{\delta}) \frac{k}{\epsilon^2}}$.

Proof. The difference between APPROXUNION and APPROXWMI lies in the implementation of task-specific oracles for WMI. Therefore, it is sufficient to show that all oracles in APPROXWMI satisfy the requirements of Corollary 2 to reach the desired result.

CLAUSEWEIGHT. We first show the correctness of CLAUSEWEIGHT for computing $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ given a concave and factorized w . Let $\mathbf{v} \models c$ be a Boolean assignment satisfying the Boolean literals of c , \mathcal{X}_c be the convex polytope defined by the LRA constraints of c , and \mathbf{v}_c be the set of Boolean literals appearing in c . Given a factorized w , WMI over c reduces to:

$$\begin{aligned} \text{WMI}(c, w \mid \mathbf{X}, \mathbf{V}) &= \sum_{\mathbf{v} \models c} \int_{\mathcal{X}_c} w(\mathbf{x}, \mathbf{v}) d\mathbf{x} \\ &= \sum_{\mathbf{v} \models c} \int_{\mathcal{X}_c} w_{\mathbf{x}}(\mathbf{x}) \prod_{p \in \mathbf{v}} w_b(p) d\mathbf{x} \\ &= \sum_{\mathbf{v} \models c} \prod_{p \in \mathbf{v}} w_b(p) \int_{\mathcal{X}_c} w_{\mathbf{x}}(\mathbf{x}) d\mathbf{x} \\ &= \left(\prod_{p \in \mathbf{v}_c} w_b(p) \right) \int_{\mathcal{X}_c} w_{\mathbf{x}}(\mathbf{x}) d\mathbf{x}. \end{aligned}$$

Note that the separation of the sum from the integral in the last step is possible because the body \mathcal{X}_c is constant across all Boolean assignments, since c is a conjunction of atoms enforcing a unique set of real constraints. Hence, the WMI of c given a factorized, concave w amounts to a product computation of Boolean literal probabilities plus a weighted integral computation over \mathcal{X}_c in the real domain \mathbb{R}^n . In CLAUSEWEIGHT, the product computation corresponds to line 2, whereas weighted integral computation corresponds to lines 4 and 5. Weighted integral computation is performed using a transformation of \mathcal{X}_c to \mathcal{X}'_c , as described in Section 3.2, and it is trivial to prove that the volume of \mathcal{X}'_c is equivalent to the weighted integral of \mathcal{X}_c . Therefore, CLAUSEWEIGHT indeed returns an approximation for $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$.

We now show that CLAUSEWEIGHT meets the error and confidence criteria of Corollary 2. Multiplication of Boolean weights is error-free, hence it is necessary and sufficient that the call to VOLUME have error and confidence upper-bounded by ϵ_V and δ_V respectively, as is the case in Algorithm 1, for CLAUSEWEIGHT to meet the requirements of Corollary 2.

SAMPLE. We first show that sampling from the transformation result \mathcal{X}'_c is equivalent to sampling from \mathcal{X}_c according to w . Let \mathbf{x}' be a real assignment sampled uniformly from $\mathcal{X}'_c \in \mathbb{R}^{n+1}$, and let \mathbf{x} be \mathbf{x}' with the last dimension omitted i.e., $\mathbf{x}'_{1:n}$. The weight of \mathbf{x} is therefore given by:

$$\int_0^{w_{\mathbf{x}}(\mathbf{x})} 1 d\mathbf{x}_{n+1} \sim w_{\mathbf{x}}(\mathbf{x}),$$

as required. This implies that, when \mathcal{X}'_c is successfully sampled with multiplicative error ϵ_S , \mathbf{x} is sampled over \mathcal{X}_c given weight function w with the same error. We now show that SAMPLE meets the requirements of Corollary 2. Boolean variable sampling, corresponding to line 9, occurs with zero error. Hence, it is necessary and sufficient to run CONVEXBODYSAMPLER with parameters ϵ_S and δ_S satisfying Condition 2 of Corollary 2, to ensure SAMPLE meets the requirements. This is indeed the case with SAMPLE, since CONVEXBODYSAMPLER is called with parameters ϵ_S and δ_S , as specified in Algorithm 1, and these parameters meet Condition 2. In particular, $\delta_P = 0$, since EVALUATE is deterministic, hence $\delta_S = \frac{\delta}{2276 \ln(\frac{8}{\delta}) \frac{k}{\epsilon^2}}$ is the largest δ_S satisfying Condition 2.

EVALUATE. This function is deterministic, so trivially meets error and confidence requirements. The function is also clearly correct.

Hence, APPROXWMI is an FPRAS for WMI given a concave and factorized w and given FPRAS functions CLAUSEWEIGHT and SAMPLE, and the deterministic EVALUATE. \square

As with APPROXUNION, APPROXWMI does not rely on specific configurations of its oracles for correctness, but only on their approximation properties. Indeed, APPROXWMI can be run with exact oracles, or with approximate weak oracles where error and confidence can be made arbitrarily small, and will provide probabilistic guarantees in both scenarios. The approximation ability of APPROXUNION is only affected once weak oracles cannot be made arbitrarily accurate (e.g., heuristics), which would bound the algorithm to minimum error values, or eliminate guarantees entirely.

In terms of running time, APPROXWMI strongly depends on the running time of its chosen oracles, as well as the required error and confidence for its guarantees. More specifically, for CLAUSEWEIGHT runtime r_C , SAMPLE runtime r_S , and EVALUATE runtime r_E , APPROXWMI runs in $O(k \cdot r_C + T(r_S + r_E))$. Though desired error ϵ and confidence δ directly affect this running time via the number of sampling iterations T , they can also indirectly affect approximate oracle runtimes, leading to even

higher computational requirements. Nonetheless, when every oracle is an FPRAS, APPROXWMI runs in polynomial time, yields the required guarantees, and thus is itself an FPRAS.

We now concretely illustrate the running time of APPROXWMI using the most general setting of its oracles, allowing for arbitrary convex bodies, and for concave and factorized w . In this setting, we can use the algorithm of Lovász and Vempala [21] as the VOLUME oracle in CLAUSEWEIGHT, and hit-and-run for CONVEXBODYSAMPLER [18]. These choices make CLAUSEWEIGHT run in time $O^*(m + n^4(\frac{1}{\epsilon_V})^2)$, SAMPLE run in time $O^*(m + n^3(\frac{1}{\epsilon_S})^2)$, where m is the number of Boolean variables, n is the number of real variables, and $*$ denotes suppressed logarithmic factors (including, in particular, confidence terms). Since EVALUATE runs in $O(m + Wn)$, where W is the width of a conjunction c , APPROXWMI therefore runs in time:

$$O^*\left(km + kn^4\left(\frac{1}{\epsilon_V}\right)^2 + Tm + Tn^3\left(\frac{1}{\epsilon_S}\right)^2 + T(m + Wn)\right).$$

Using the bounds $T = O(\frac{k}{\epsilon^2} \ln(\frac{1}{\delta}))$, $\epsilon_V = O(\frac{\epsilon^2}{k})$, and $\epsilon_S = O(\frac{\epsilon^2}{k})$ from Theorem 3, we obtain the complexity:

$$O^*\left(km + \frac{k^3}{\epsilon^4}n^4 + \frac{k}{\epsilon^2}m + \frac{k^3}{\epsilon^6}n^3 + \frac{k(Wn + m)}{\epsilon^2}\right) = O^*\left(\frac{k^3}{\epsilon^4}n^4 + \frac{k^3}{\epsilon^6}n^3 + \frac{k}{\epsilon^2}(m + Wn)\right).$$

Note that the time complexity of SAMPLE can be reduced by restricting the class of bodies and weight functions used. Indeed, if w is restricted to be linear, e.g., $3x_1 + 2x_2 - x_4$, then all bodies can be sampled approximately using optimized polytope sampling methods such as geodesic walks [25], which can run significantly faster for bodies defined with a small number of LRA constraints. Further to this, box-shaped bodies, defined with LRA constraints having only one variable, and a constant, uniform w , are an instance of unweighted model integration, and can be trivially sampled. Nonetheless, we assume the general case of convex bodies in this work, and build our algorithm accordingly.

In addition to naturally running over standard WMI formulations with LRA constraints, APPROXWMI can also apply to WMI instances defined with other theories, so long as these theories define convex bodies in individual DNF clauses. For instance, APPROXWMI also applies to WMI with *concave* polynomial constraints over real variables, a theory strictly subsumed by the theory of non-linear real arithmetic (NRA), but which strictly contains LRA. Hence, APPROXWMI can capture a richer class of WMI formulations, at no additional computational cost.

Nonetheless, APPROXWMI is limited to concave, factorized weight functions by construction: concavity is needed to ensure the convexity of the intermediate bodies, and the factorization assumption is essential to computations in CLAUSEWEIGHT and SAMPLE. Unfortunately, relaxing the concavity requirement does not currently look promising, as we run into intractable computational tasks, such as arbitrary body volume computation, which typically do not have computationally feasible approximation schemes. Therefore, in our subsequent study of WMI, we focus on generalizing APPROXWMI in a tractable fashion, for instance through relaxing the common factorization assumption, so as to tackle more general weight functions allowing dependencies between Boolean and real variables.

3.4. Extending APPROXWMI: APPROXWMI_D

In Section 3.2, we presented APPROXWMI, an FPRAS for WMI over DNF formulas given a factorized and concave weight function w . The factorization of the weight function simplifies WMI, in that (i) it makes Boolean variable weights independent from real variables, and (ii) simplifies the joint distribution over Booleans to a product distribution. However, this factorization prevents any interaction between Boolean and real variables, which limits the representation power of the weight function. Even simple weight functions cannot be captured using the factorization assumption, as we illustrate next.

Example 4. Consider $\mathbf{X} = \{x_1\}$, $\mathbf{V} = \{p_1\}$ and the following weight function:

$$w(\mathbf{x}, \mathbf{v}) = \begin{cases} 2x_1 & \text{if } p_1, \\ 1 & \text{otherwise.} \end{cases}$$

It is easy to see that this weight function invalidates the factorization assumption, as the real component weight changes based on \mathbf{v} .

To alleviate the limitations of this factorization, it has been proposed to define separate weight functions over *mutually exclusive* Boolean partitions of the problem domain, and subsequently solve the WMI problem separately over each of these partitions [20]. For instance, in our example, one can define two weight functions $w_0(\mathbf{x}) = 2x_1$ and $w_1(\mathbf{x}) = 1$, corresponding to the different truth values of each possible Boolean assignment, solve the corresponding problems and subsequently aggregate their results. Through this partitioning scheme, the hybrid domain is separated into subspaces where dependencies do not exist, and therefore where the factorization assumption applies. However, this partitioning can produce up to $2^{|\mathbf{V}|}$ sub-problems, and that makes this separation approach intractable in practice.

In this section, we extend the applicability of APPROXWMI to more general weight functions allowing for such dependencies, while maintaining tractability. More concretely, we study weight functions:

$$w(\mathbf{x}, \mathbf{v}) = w_x(\mathbf{x}, \mathbf{v}) \prod_{p \in \mathbf{v}} w_b(p), \quad (3)$$

where w_x also depends on Boolean variables (unlike Equation (2)), allowing the aforementioned dependencies.

First, we note that this factorization is very general, as the w_x term alone is sufficient to capture any arbitrary weight function, and therefore it can clearly capture $w(\mathbf{x}, \mathbf{v})$ in Example 4. In fact, it is only a factorization in that it maintains independence between Boolean variables, and thus reduces the joint distribution over these variables to a product distribution.

In this more general setting, we build on APPROXWMI, and propose an FPRAS APPROXWMI_D that naturally incorporates the additional dependencies supported by Equation (3). APPROXWMI_D modifies clause weight computation to consider multiple different randomly sampled Boolean assignments per clause, and aggregates results for these assignments to yield an approximate overall clause weight. It also extends point sampling to respect the different distributions which distinct Boolean variable assignments may impose.

Prior to introducing APPROXWMI_D, we first introduce some key notation and concepts. As earlier, let $\mathcal{X}_c \subseteq \mathbb{R}^n$ be the convex polytope induced by the LRA constraints of a clause c , $\mathbf{v} \in \mathbb{B}^m$ be a Boolean assignment, and w_x be the hybrid component of weight function w , defined following Equation (3). We now define two functions a and b that map all subsets of \mathbb{R}^n to positive real values. Intuitively, these functions will act as lower and upper bounds for the integral of w_x such that, for any subset of \mathbb{R}^n , the integral of w_x over this subset will be bounded by a and b , for all possible Boolean assignments.

In defining a and b , we aim to capture the *variability* of w_x relative to changes in the Boolean domain. This variability, introduced by our generalization of w , directly impacts the sample complexity needed to approximate the weight of a given clause, and thus we define a and b to threshold the range of w_x , and maintain a tractable number of random samples needed for clause weight computation in APPROXWMI_D.

Formally, we define $a, b : P(\mathbb{R}^n) \mapsto \mathbb{R}^+$, where P denotes the power set operation, such that

$$a(\mathcal{X}_c) = \min_{\mathbf{v}} \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) d\mathbf{x}, \text{ and}$$

$$b(\mathcal{X}_c) = \max_{\mathbf{v}} \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) d\mathbf{x}.$$

We first note that the minimum and maximum functions a and b exist, as the Boolean domain \mathbb{B}^m only defines a *finite* set of 2^m distinct possible values for the integral of $w_x(\mathbf{x}, \mathbf{v})$. We also note that a and b are finite-valued since all our WMI integral computations are finite-valued, as continuous variables are bounded. As a result, the bound functions a and b exist for the integral of any function w_x (with finite density over \mathbb{R}^n), over any subset of \mathbb{R}^n , in particular a convex polytope \mathcal{X}_c . This translates to the following statement:

$$\forall \mathcal{X}_c \in P(\mathbb{R}^n), \exists a, b, \forall \mathbf{v} \in \mathbb{B}^m : P(\mathbb{R}^n) \mapsto \mathbb{R}^+, a(\mathcal{X}_c) \leq \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) d\mathbf{x} \leq b(\mathcal{X}_c). \quad (4)$$

Using a and b , we now define the maximum ratio ρ , to better study the variability of w_x :

$$\rho = \max_c \frac{b(\mathcal{X}_c)}{a(\mathcal{X}_c)}.$$

Intuitively, ρ implicitly produces a range of integral values for w_x given a convex polytope \mathcal{X}_c . More concretely, if w_x yields an integral value I for \mathcal{X}_c for a given Boolean assignment \mathbf{v} , then w_x cannot produce integrals over \mathcal{X}_c with a different assignment \mathbf{v}' that are larger than ρI , or smaller than $\frac{I}{\rho}$. Hence, ρ correlates directly with Boolean-real dependency. In fact, the special case $\rho = 1$, its minimum possible value, corresponds to independence between Boolean and real variables, and thus to w_x defined using the factorization assumption.

Using ρ , we estimate the sample complexity for approximating clause weights under the factorization of Equation (3). In fact, we show that this sample complexity depends quadratically on ρ , and therefore that, for any w , ρ must be upper-bounded by a polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, n , m , and k , to maintain polynomial sample complexity. In what follows, we refer to weight functions w satisfying this criterion as ρ -restricted.

In spirit, our definition and restriction of ρ resembles the restriction of *tilt* θ in approximation methods for weighted model counting [26]. For those approximate WMC approaches, tilt imposes a maximum ratio between the minimum and maximum *values* of the weight function over the Boolean domain. However, such restrictions on θ are *tighter* than restrictions on ρ . Indeed, ρ is based on the ratio between the maximum integral and minimum integral of a function over a *fixed* real body, and so does not compare between integral values over different bodies as with tilt. As a result, when $\theta = H$, it naturally follows that $\rho \leq H$, whereas the opposite direction does not hold, yielding the inequality $\rho \leq \theta$. We now illustrate the difference between ρ and θ with two examples.

Example 5. Let $\mathbf{V} = \{p_1\}$, $\mathbf{X} = \{x_1\}$, $0 \leq x_1 \leq 10$, and let:

$$w_x(\mathbf{x}, \mathbf{v}) = \begin{cases} 3 & \text{if } p_1, \\ 2 & \text{otherwise.} \end{cases}$$

Then, $\rho = 1.5$, and $\theta = 1.5$, and thus the bound $\rho \leq \theta$ is tight. More generally, $\rho = \theta$ holds for all w_x defined as piece-wise constants over \mathbf{V} .

In Example 5, we consider a piece-wise constant function over \mathbf{V} , and observe that $\rho = \theta = 1.5$: for tilt, the maximum and minimum values of w_x are 3 and 2 respectively, whereas $\rho = 1.5$ since the maximum integral ratio over any body necessarily involves alternating p_1 . Hence, the inequality $\rho \leq \theta$ is tight. However, piece-wise constant functions are very limited in their representation capacity. By contrast, for functions with variations for a fixed Boolean assignment, the scenario changes completely, as Example 6 shows.

Example 6. Let $\mathbf{V} = \{p_1\}$, $\mathbf{X} = \{x_1\}$, $0 \leq x_1 \leq 10$, and let:

$$w_x(\mathbf{x}, \mathbf{v}) = \begin{cases} \frac{e^{10}}{1+e^{-x_1}} - 0.5e^{10} + 1 & \text{if } p_1, \\ \frac{2e^{10}}{1+e^{-x_1}} - e^{10} + 2 & \text{otherwise.} \end{cases}$$

Then, $\rho = 2$, but $\theta \approx 2 + e^{10}$. Though w_x for $p_1 = 1$ is simply half the other case, the variability within every case is significant, and alone produces a tilt of $1 + 0.5e^{10}$. By contrast, ρ is not affected by this variability.

In this example, ρ is clearly upper-bounded by 2. However, the tilt θ is almost equal to $2 + e^{10}$, since the maximum of w_x is $w_x(10, \text{false}) = \frac{2e^{10}}{1+e^{-10}} - e^{10} + 2 \approx e^{10} + 2$, and its minimum is $w_x(0, \text{true}) = \frac{e^{10}}{1+e^{-0}} - 0.5e^{10} + 1 = 1$. In this example, w_x has large variability in its cases, and this raises tilt, but not does affect the ratio ρ .

Theoretically, ρ helps assess the viability of clause weight approximation using sampling techniques. However, it may not be feasibly computable. In this case, a simpler quantity ρ' , evaluated over points in \mathbb{R}^n , rather than subsets, can be used instead. Fundamentally, ρ' is defined as the ratio between the maximum $w(\mathbf{x}, \mathbf{v})$ and the minimum $w(\mathbf{x}, \mathbf{v}')$ over all Boolean assignments. Note that these extrema are defined relative to an *identical* real assignment \mathbf{x} , but potentially distinct \mathbf{v} and \mathbf{v}' . More formally,

$$\rho' = \max_{\mathbf{x}} \frac{w_{\max}(\mathbf{x})}{w_{\min}(\mathbf{x})}, \quad (5)$$

where $w_{\min}(\mathbf{x}) = \min_{\mathbf{v}} w_x(\mathbf{x}, \mathbf{v})$ and $w_{\max}(\mathbf{x}) = \max_{\mathbf{v}} w_x(\mathbf{x}, \mathbf{v})$.

Observe that tilt θ is also more restrictive than ρ' , as its objective, though similar to that of ρ' , uses *distinct* assignments \mathbf{x} and \mathbf{x}' in Equation (5). Hence, $\rho' \leq \theta$. Furthermore, since ρ' also restricts its ratio to identical real assignments, similarly to ρ , it can also be significantly smaller than θ . For instance, in Example 6, $\rho' = 2$. Finally, we note that $\rho \leq \rho'$, as no integral over any subset of \mathbb{R}^n can exceed the ratio ρ' given positive w_x . Therefore, $\rho \leq \rho' \leq \theta$.

We now introduce APPROXWMI_D, an extension to APPROXWMI that supports weight functions w factorized according to Equation (3). At a high level, APPROXWMI_D operates identically to APPROXWMI, but replaces the oracles CLAUSEWEIGHT and SAMPLE with CLAUSEWEIGHT_D and SAMPLE_D, respectively, such that these oracles now incorporate Boolean-real dependencies. Otherwise, all algorithmic details of APPROXWMI, namely the sampling iteration structure based on LTC, and clause membership checking via EVALUATE, remain the same.

Intuitively, CLAUSEWEIGHT_D considers Boolean-real dependencies via *Monte-Carlo sampling* over the Boolean sub-domain, and subsequently averaging volume computations across different samples to yield an overall estimate of the clause weight. At this stage, ρ -restriction is key, as it makes (ϵ_V, δ_V) -approximation for the oracle to be computable using a polynomial number of samples. On the other hand, SAMPLE_D only marginally changes relative to SAMPLE, in that Boolean assignments are initially sampled, and real assignment sampling is now conditioned on these Boolean assignments. The pseudo-code for these two oracles is provided in Algorithm 3. We now discuss these oracles in detail.

CLAUSEWEIGHT_D. To exactly compute the weight of a clause c under the factorization of Equation (3), one must iterate over an *exponential* number of satisfying Boolean assignments of c , compute scores for each assignment using a volume computation tool, and finally return a weighted average of these scores as the WMI of c . Therefore, CLAUSEWEIGHT_D performs *Monte-Carlo sampling* over the set of Boolean assignments to approximate the WMI of c .

In this setup, CLAUSEWEIGHT_D has two sources of error and failure, namely, (i) the Monte-Carlo sampling procedure itself, which has error ϵ_{Samp} and confidence δ_{Samp} , and (ii) the error and confidence of the volume computation tool, denoted ϵ_{Comp} and δ_{Comp} respectively. Hence, CLAUSEWEIGHT sets these parameters such that their combination yields error and confidence bounds within the overall (ϵ_V, δ_V) requirement.

In CLAUSEWEIGHT_D, sampling is conducted for s iterations, where s is a function of ϵ_{Samp} and δ_{Samp} , and ρ (see comment “# of sampling trials” in Algorithm 3 for the closed-form equation). Within every trial, VOLUME is called with error ϵ_{Comp}

Algorithm 3 Subroutines of APPROXWMI_D as functions CLAUSEWEIGHT_D and SAMPLE_D.

```

1: function CLAUSEWEIGHTD( $c, w, \mathbf{X}, \mathbf{V}$ )[ $\epsilon, \delta$ ]
2:    $\epsilon_{\text{Samp}}, \epsilon_{\text{Comp}} \leftarrow \frac{\epsilon}{1+\sqrt{2}}, \delta_{\text{Samp}} \leftarrow \frac{\delta}{2}$ 
3:    $s \leftarrow \ln\left(\frac{2}{\delta_{\text{Samp}}}\right) \frac{1}{2\epsilon_{\text{Samp}}} \rho^2$  ▷ # of sampling trials
4:    $\delta_{\text{Comp}} \leftarrow \frac{\delta}{2s}$ 
5:   for  $i \leftarrow 1$  to  $s$  do ▷ Perform Monte-Carlo sampling over clause weights
6:      $\mathbf{v}_i \leftarrow \text{sample from } \mathbf{V} \text{ respecting } \mathbf{v}_c \text{ and } w_b$ 
7:     Introduce a new variable  $d$  in  $\mathbf{X}$ 
8:      $\mathcal{X}'_c \leftarrow \mathcal{X}_c \wedge (\mathbf{x} \in \mathbb{R}^n, 0 \leq d \leq w_x(\mathbf{x}, \mathbf{v}_i))$  ▷ Create convex body using the induced  $w_x$ 
9:      $X_i \leftarrow \text{VOLUME}(\mathcal{X}'_c)[\epsilon_{\text{Comp}}, \delta_{\text{Comp}}]$  ▷ Return its volume as the Monte-Carlo sample value
10:   BoolWeight  $\leftarrow \prod_{p \in \mathbf{v}_c} w_b(p)$ 
11:   return BoolWeight  $\cdot \frac{1}{s} \sum_{i=1}^s X_i$  ▷ Return sample mean as clause weight estimate
12: function SAMPLED( $c, w, \mathbf{X}, \mathbf{V}$ )[ $\epsilon, \delta$ ]
13:    $\mathbf{v} \leftarrow \text{sample from } \mathbf{V} \text{ respecting } \mathbf{v}_c \text{ and } w_b$  ▷ Boolean sampling
14:   Introduce a new variable  $d$  in  $\mathbf{X}$ 
15:    $\mathcal{X}'_c \leftarrow \mathcal{X}_c \wedge (\mathbf{x} \in \mathbb{R}^n, 0 \leq d \leq w_x(\mathbf{x}, \mathbf{v}))$ 
16:    $\mathbf{x} \leftarrow \text{CONVEXBODYSAMPLER}(\mathcal{X}'_c)[\epsilon, \delta]_{1:n}$  ▷ Perform sampling over conditional distribution
17:   return  $(\mathbf{x}, \mathbf{v})$ 
18:
```

and confidence δ_{Comp} . More specifically, each sampling iteration consists of randomly sampling a Boolean assignment \mathbf{v}_i satisfying c according to w_b (line 6), computing the induced weight function $w_x(\mathbf{x}, \mathbf{v}_i)$, and using this induced function to obtain the transformed convex body \mathcal{X}'_c (line 7), analogously to SAMPLE and CLAUSEWEIGHT. At this point, CLAUSEWEIGHT_D behaves identically to CLAUSEWEIGHT: it computes the weighted integral over c using a subroutine VOLUME (line 8).

Following all s sampling steps, the average of all samples is computed, multiplied by the product of all \mathbf{v}_c literal weights, and returned (lines 9 and 11). The left-hand side of the returned value is identical to that of CLAUSEWEIGHT, and reflects the continued independence of individual Boolean variables in this new factorization, whereas the right-hand side completes the approximation of $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ and incorporates the w_x term via the sampling average weight.

SAMPLE_D. This function is defined almost identically to SAMPLE in APPROXWMI, with the difference being that $w_x(\mathbf{x}, \mathbf{v})$ is now induced from \mathbf{v} (line 16) prior to applying the transformation. Unlike SAMPLE, it is necessary for Boolean sampling to run first in SAMPLE_D, given the factorization, so as to condition w_x on the Boolean sample output and subsequently sample from \mathcal{X}'_c .

3.5. APPROXWMI_D is an FPRAS

We now show that APPROXWMI_D is an FPRAS for WMI(DNF), by lifting the result given in Theorem 3. First, we show the correctness of CLAUSEWEIGHT_D, and prove that it is an FPRAS for $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ over concave, ρ -restricted w factorized according to Equation (3), given sampling and volume computation operations whose error and confidence can be made arbitrarily small.

The correctness proof for CLAUSEWEIGHT_D relies on the correctness of Monte-Carlo sampling for producing approximations with probabilistic guarantees, and its polynomial-time tractability is shown with a probabilistic argument, proving that the CLAUSEWEIGHT_D sampling procedure only requires a polynomial number of samples given the ρ -restriction of w_x . In particular, given a clause c and its convex polytope \mathcal{X}_c , this proof establishes that all random samples are necessarily bound by the functions $a(\mathcal{X}_c)$ and $b(\mathcal{X}_c)$, as per Equation (4). Then, since these samples are independent and identically distributed, their average approximates the expected value of the sampled distribution with probabilistic guarantees from the Hoeffding bound. These guarantees, complemented by the ρ -restriction of w_x , yield a polynomial lower bound for the number of samples s . Hence, the sampling procedure can have arbitrarily small error, and introduces a worst-case polynomial overhead to CLAUSEWEIGHT_D.

Finally, the proof shows that CLAUSEWEIGHT_D produces an (ϵ, δ) -approximation of $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ given the error and confidence values for sampling and volume computation shown in Algorithm 3, as these values, once aggregated across the operation of CLAUSEWEIGHT_D, remain below ϵ and δ respectively, and therefore, CLAUSEWEIGHT_D is an FPRAS as required. We now formally state this result, along with its necessary conditions, as a Lemma, and provide its complete proof.

Lemma 4. CLAUSEWEIGHT_D relative to Monte-Carlo sampling error ϵ_{Samp} and confidence δ_{Samp} , and an FPRAS VOLUME with error ϵ_{Comp} and confidence δ_{Comp} , is an FPRAS for $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$, where c is a conjunctive clause and w is concave, ρ -restricted, and factorized according to Equation (3), with error ϵ , confidence δ , and using $s = \ln\left(\frac{2}{\delta_{\text{Samp}}}\right) \frac{1}{2\epsilon_{\text{Samp}}} \rho^2$ iterations, for

1. $\epsilon_{\text{Samp}}, \epsilon_{\text{Comp}} \leq \frac{\epsilon}{1+\sqrt{2}},$
2. $\delta_{\text{Samp}} \leq \frac{\delta}{2},$ and
3. $\delta_{\text{Comp}} \leq \frac{\delta}{2s}.$

Proof. We first show the correctness of CLAUSEWEIGHT_D for computing $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$. Under the factorization of Equation (3), w_x now depends on both real and Boolean variables. Hence, the decomposition of WMI computation into two separate Boolean and real parts used in the proof of Theorem 3 no longer holds. Instead, WMI computation over a clause c given w can only be written as:

$$\begin{aligned} \text{WMI}(c, w \mid \mathbf{X}, \mathbf{V}) &= \sum_{\mathbf{v} \models c} \prod_{p \in \mathbf{v}} w_b(p) \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) d\mathbf{x} \\ &= \left(\prod_{p \in \mathbf{v}_c} w_b(p) \right) \sum_{\mathbf{v} \models c} \left(\prod_{p \in \mathbf{v} \setminus \mathbf{v}_c} w_b(p) \int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) d\mathbf{x} \right), \end{aligned} \quad (6)$$

where \mathbf{v}_c denotes the set of all Boolean literals appearing in c .

Note that the outer summation no longer simplifies as in Theorem 3, as the inner integral now depends on the Boolean assignment \mathbf{v} . Therefore, WMI computation over c in this setting requires $2^{m-|\mathbf{v}_c|}$ calls to a weighted volume computation tool, which is simulated by calling the VOLUME FPRAS on \mathcal{X}'_c , the convex body transformation of \mathcal{X}_c having an additional weight-constrained dimension.

Given the intractability of exactly performing the computation in Equation (6), CLAUSEWEIGHT_D uses *Monte-Carlo sampling* to approximate the WMI of c . We show in this proof that this sampling is (i) correct, (ii) returns probabilistic guarantees, and (iii) can be performed in polynomial time, since w is ρ -restricted.

We start by proving the correctness of Monte-Carlo sampling for approximating $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$. In Equation (6), observe that the final form for $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$ can also be written as

$$\prod_{p \in \mathbf{v}_c} w_b(p) \mathbb{E}_{\mathbf{v} \models c} \left[\int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) d\mathbf{x} \right], \quad (7)$$

since the summation, weighted by the probability product for literals not appearing in c , corresponds to the expected value of the inner integral random Boolean assignment from the distribution w_b . Hence, Equation (6) can be approximated with probabilistic guarantees by taking Monte-Carlo samples from the aforementioned distribution and computing the mean of all integral samples. This sampling is done as follows:

1. Sample a random Boolean assignment \mathbf{v} using w_b (CLAUSEWEIGHT_D line 6).
2. Compute the resulting weighted model integral over \mathcal{X}_c given $w_x(\mathbf{x}, \mathbf{v})$ using $\text{VOLUME}[\epsilon_{\text{Comp}}, \delta_{\text{Comp}}]$. (CLAUSEWEIGHT_D lines 7-8.)

As the mean of this sampling procedure approximates the summation in Equation (6), it only remains to multiply this mean by the product of Boolean weights, which corresponds to lines 9 and 11 of CLAUSEWEIGHT_D , to correctly yield an approximation of $\text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$, as required.

We now study probabilistic guarantees for CLAUSEWEIGHT_D . The sampling procedure produces s independent and identically distributed (i.i.d.) random variables from the w_b -weighted distribution of possible integral values. Let \bar{X} denote the mean of these s random variables, and let $\mu = \mathbb{E}_{\mathbf{v} \models c} \left[\int_{\mathcal{X}_c} w_x(\mathbf{x}, \mathbf{v}) d\mathbf{x} \right]$ for ease of notation. From Equation (4), we infer that $a(\mathcal{X}_c) \leq \mu \leq b(\mathcal{X}_c)$. Therefore, we can use $a(\mathcal{X}_c)$ and $b(\mathcal{X}_c)$ within the Hoeffding bound for sampling s i.i.d. variables. This yields, for a target additive difference t ,

$$\Pr(|\bar{X} - \mu| \geq t) \leq 2 \exp \left(\frac{-2st^2}{(b(\mathcal{X}_c) - a(\mathcal{X}_c))^2} \right)$$

We now replace t with $\epsilon_{\text{Samp}}\mu$ to obtain a multiplicative error bound, and upper-bound the failure probability by δ_{Samp} to compute a lower bound for sample complexity:

$$s \geq \ln \left(\frac{2}{\delta_{\text{Samp}}} \right) \frac{1}{2\epsilon_{\text{Samp}}^2} \frac{(b(\mathcal{X}_c) - a(\mathcal{X}_c))^2}{\mu^2} \quad (8)$$

$$\geq \ln \left(\frac{2}{\delta_{\text{Samp}}} \right) \frac{1}{2\epsilon_{\text{Samp}}^2} \frac{(b(\mathcal{X}_c) - a(\mathcal{X}_c))^2}{a(\mathcal{X}_c)^2} \quad (9)$$

$$\geq \ln \left(\frac{2}{\delta_{\text{Samp}}} \right) \frac{1}{2\epsilon_{\text{Samp}}^2} \left(\frac{b(\mathcal{X}_c)}{a(\mathcal{X}_c)} \right)^2 \quad (10)$$

$$\geq \ln \left(\frac{2}{\delta_{\text{Samp}}} \right) \frac{1}{2\epsilon_{\text{Samp}}^2} \rho^2 \quad (11)$$

To obtain Equation (9), we replace μ by $a(\mathcal{X}_c)$ to yield the most pessimistic lower bound possible for s : Since $a(\mathcal{X}_c) \leq \mu \leq b(\mathcal{X}_c)$, then replacing μ by its minimum value maximizes the fraction, and results in a worst-case sample bound that is sufficient to produce the required error and confidence guarantees for any value of μ . To obtain Equation (10), we eliminate the $a(\mathcal{X}_c)$ term in the numerator, which is possible, since $a(\mathcal{X}_c) \geq 0$, to obtain an explicit ratio between a and b , which can be replaced with ρ in Equation (11), as ρ , by definition, is greater or equal to the maximum value of this ratio for any convex polytope.¹

Given s Monte-Carlo samples, CLAUSEWEIGHT_D introduces an error of at most ϵ_{Samp} with probability $1 - \delta_{\text{Samp}}$. This sample complexity s is polynomial in $\frac{1}{\epsilon_{\text{Samp}}}, \frac{1}{\delta_{\text{Samp}}}, n, m$, and k as w is ρ -restricted. Furthermore, a sampling step runs in polynomial time, as both Boolean sampling and the VOLUME FPRAS call run in polynomial time. Hence, CLAUSEWEIGHT_D runs in polynomial time, more precisely in $O^*(sn^4\epsilon_{\text{Comp}}^{-2}) = O^*(n^4\epsilon_{\text{Samp}}^{-2}\epsilon_{\text{Comp}}^{-2}) = O^*(n^4\epsilon_V^{-4}) = O^*(n^4\epsilon^{-8})$, where ϵ is the overall target error for APPROXWMI_D.

Finally, we show that, under the conditions of Lemma 4, CLAUSEWEIGHT_D produces an (ϵ, δ) -approximation of WMI($c, w \mid \mathbf{X}, \mathbf{V}$). In what follows, we assume that CLAUSEWEIGHT_D fails if Monte-Carlo sampling fails or if *any* of the VOLUME calls fail. Therefore, we first prove that, when no failure occurs, the error bounds in Lemma 4 produce an estimate within the ϵ error requirement. Then, we prove that the asserted confidence bounds upper-bound the probability of any CLAUSEWEIGHT_D failure, denoted f_V , by δ .

Error bounds. We first assume a successful run where Monte-Carlo sampling and VOLUME produce values within their error bounds. Setting ϵ_{Samp} and ϵ_{Comp} within the bounds of Lemma 4 yields the following bound on \bar{X} :

$$\left(1 - \frac{\epsilon}{1 + \sqrt{2}}\right)^2 \mu \leq \bar{X} \leq \left(1 + \frac{\epsilon}{1 + \sqrt{2}}\right)^2 \mu.$$

For $\forall 0 < \epsilon \leq 1$, we have that:

$$(1 + \epsilon) - \left(1 + \frac{\epsilon}{1 + \sqrt{2}}\right)^2 = \epsilon - \frac{2\epsilon}{1 + \sqrt{2}} - \frac{\epsilon^2}{(1 + \sqrt{2})^2} = \epsilon \left(\frac{(1 + \sqrt{2})^2 - 2(1 + \sqrt{2}) - \epsilon}{(1 + \sqrt{2})^2} \right) = \frac{\epsilon(1 - \epsilon)}{(1 + \sqrt{2})^2} \geq 0.$$

Analogously,

$$\left(1 - \frac{\epsilon}{1 + \sqrt{2}}\right)^2 - (1 - \epsilon) = \epsilon - \frac{2\epsilon}{1 + \sqrt{2}} + \frac{\epsilon^2}{(1 + \sqrt{2})^2} = \frac{\epsilon(1 + \epsilon)}{(1 + \sqrt{2})^2} \geq 0.$$

Therefore, we can replace the bounds in the original inequality by $(1 + \epsilon)$ and $(1 - \epsilon)$ respectively, as follows:

$$(1 - \epsilon)\mu \leq \bar{X} \leq (1 + \epsilon)\mu,$$

and, following multiplication by $\left(\prod_{p \in \mathbf{V}_c} w_b(p)\right)$ on all sides, we obtain:

$$(1 - \epsilon)Q \leq \bar{X} \left(\prod_{p \in \mathbf{V}_c} w_b(p) \right) \leq (1 + \epsilon)Q,$$

where $Q = \text{WMI}(c, w \mid \mathbf{X}, \mathbf{V})$, as required.

Confidence bounds. VOLUME is called s times within CLAUSEWEIGHT_D. Hence, we apply the union bound, to upper-bound f_V , and obtain:

$$\Pr(f_V) \leq s\delta_{\text{Comp}} + \delta_{\text{Samp}} = s\frac{\delta}{2s} + \frac{\delta}{2} = \frac{\delta}{2} + \frac{\delta}{2} = \delta. \quad \square$$

We now combine Theorem 3 and Lemma 4 to show that APPROXWMI_D is an FPRAS for WMI(DNF) given a concave and ρ -restricted w under the factorization of Equation (3).

Theorem 5. APPROXWMI_D relative to FPRAS oracles CLAUSEWEIGHT_D, SAMPLE_D, and EVALUATE having error $\epsilon_V, \epsilon_S, \epsilon_P$ and confidence $\delta_V, \delta_S, \delta_P$, respectively, is an FPRAS for WMI(DNF) over a w that is concave, ρ -restricted, and factorized according to Equation (3), with error ϵ and confidence δ and using $T = \frac{8 \ln(\frac{8}{\delta})(1+\epsilon)k}{\epsilon^2 - 8(\bar{c}-1)k}$ iterations, for

1. $\epsilon_V, \epsilon_S \leq \frac{\epsilon^2}{47k}, \epsilon_P \leq \frac{\epsilon^2}{47k^2}$, and
2. $\delta_V \leq \frac{\delta}{4k}, \delta_S + \delta_P \leq \frac{\delta}{2276 \ln(\frac{8}{\delta}) \frac{k}{\epsilon^2}}$.

¹ Note that, in practice, one can use ρ' instead of ρ for a higher, but tractably computable sampling bound, as $\rho \leq \rho'$.

Finally, the running time of APPROXWMI_D, as a function of CLAUSEWEIGHT_D runtime r'_C , SAMPLE_D runtime r'_S , and EVALUATE runtime r_E , is $O(k \cdot r'_C + T(r'_S + r_E))$, analogously to APPROXWMI. Furthermore, for the same choice of CONVEX-BODYSAMPLER, it is easy to see that $r_S = r'_S$. However, the main difference in running time between APPROXWMI and APPROXWMI_D comes from the difference between r'_C and r_C . Indeed, given the same choice of VOLUME oracle, with running time r_V , $r_C = O(m + r_V)$, whereas $r'_C = O(s(m + r_V)) = O(\frac{1}{\epsilon^4} \ln(\frac{1}{\delta})(m + r_V))$. Hence, the wider applicability of APPROXWMI_D comes at the expense of a larger runtime complexity, owing to the larger power of $\frac{1}{\epsilon}$ required for CLAUSEWEIGHT_D.

4. Experimental evaluation

In this section, we empirically evaluate the performance of APPROXWMI on a set of randomly generated DNF formulas. In particular, we measure the running time APPROXWMI required to solve different DNF instances, and analyze the impact of different DNF parameters, such as clause width, number of variables, and number of clauses, on this running time.

First, we present our DNF generation procedure. Then, we present and justify our experimental setup, namely our choices for weight functions and oracles for sampling and volume computation. Finally, we report our experimental results, and show the runtime efficiency, efficacy and scalability of APPROXWMI.

4.1. Generating evaluation data

To evaluate APPROXWMI, we randomly generate an evaluation set of DNF formulas. This set is generated with different configurations relative to the number of Boolean and real variables, the width of clauses, and the number of these clauses, such that all possible configurations are evenly represented during evaluation. More specifically, we set our configuration parameters as follows:

- **Number of Boolean and real variables (m, n):** We generate DNF formulas such that they have an equal number of real and Boolean variables, i.e., $m = n$, $|\mathbf{X}| = |\mathbf{V}|$. We opt for setting $m = n$ to produce DNF instances that fairly represent both real and Boolean domains in the hybrid domain, and avoid edge case instances that too closely resemble either volume computation (for $m \approx 0$) or standard weighted model counting (for $n \approx 0$).
- **Clause width (W):** For simplicity, we opt for a uniform clause width W in our generation, such that all the clauses of a single DNF formula have width W .
- **Number of clauses (k):** The number of clauses is set as $k = \lfloor \frac{m+n+20}{W} \rfloor$, as this choice for k ensures a balanced computational requirement for different similarly-sized DNF instances. More precisely, this value of k ensures that the total number of atoms present across all DNF clauses (including redundancies) in a given formula remains highly similar to that of distinct formulas with identical values of m and n , but different widths W .

For every configuration (m, n, k, W) , a DNF formula is randomly generated. Initially, the configuration imposes a high-level structure of the DNF, namely that it has a fixed number of clauses k , and each of these clauses has W atoms. What remains to obtain a fully specified DNF is therefore to *assign* values to the atoms in every clause, either to Boolean literals or LRA constraints defined over \mathbf{X} . We perform this allocation of atoms analogously to the common “balls in boxes” combinatorial problem.

More specifically, we define the kW atom locations in the DNF as “balls”, and define “boxes” corresponding to all Boolean variables and to a number of LRA constraints. Then, we randomly allocate the balls to these boxes, such that no box remains empty. Thus, this allocation ensures that all Boolean variables, and a predetermined number of LRA clauses, appear in at least one clause of the DNF. Concretely, we define m Boolean boxes and n LRA boxes, which are placeholders that each independently yield LRA constraints in the final DNF. This ensures that an almost even split between LRA and Boolean atoms arises in our generated formulas. Furthermore, since n LRA boxes are defined over \mathbf{X} , and $|\mathbf{X}| = n$, and since LRA atoms involve at least one real variable, then every real variable is expected to appear in at least one LRA atom. We illustrate this construction with a running example in Example 7.

Example 7. Consider a configuration (m, n, k, W) , where $m = n = 3$, $k = 3$, and $W = 3$. Then, we obtain $kW = 9$ “balls” corresponding to all 9 atom positions in the DNF, and 6 “boxes”, namely 3 Boolean boxes corresponding to Boolean variables p_0, p_1, p_2 respectively, and 3 LRA placeholder boxes l_0, l_1, l_2 . These placeholders can be represented with a single symbol l^* , as each of the appearances of l^* will later be independently instantiated with LRA constraints over \mathbf{X} .

Remark. Though we use a single placeholder l^* to generate LRA constraints, we nonetheless represent this placeholder using n boxes (rather than 1) in our generation procedure, as this produces more uniform allocation distributions for l^* with respect to Boolean variables. More concretely, by using n boxes, we have equality of Boolean literal and LRA constraint appearances in expectation.

To allocate atom balls to boxes, we define two distinct allocation schemes, such that one of these is uniformly randomly selected for every DNF to be generated:

- **Uniform allocation:** All balls are allocated uniformly from the set of all $\binom{kW-1}{m+n-1}$ configurations² yielding non-empty boxes as in the standard balls in boxes problem. However, an additional check is imposed to prevent a box from receiving more balls than the number of clauses k .
- **Privileged allocation:** A small subset of “privileged” boxes is uniformly randomly selected, and randomly allocated another random “surplus” subset of balls, such that the removal of this surplus subset from the overall ball set keeps at least as many balls as overall boxes. In other words, for a set of kW balls and $m+n$ boxes, where $kW \geq m+n$, the size of this surplus subset cannot exceed $kW - m - n$. This privileged mechanism ensures that a small number of Boolean variables can appear across many different clauses, and thus creates strong dependencies within the DNF. Following privileged allocation, all remaining balls are allocated in a uniform fashion, as described earlier.

Following box allocation, a heuristic is used to identify specific DNF atom locations for every variable and LRA constraint, such that no variable or constraint appears twice in the same clause. Intuitively, the heuristic sets the allocations of boxes in decreasing order of their ball allocations, while keeping the number of available positions in all clauses as evenly spread as possible. Finally, once all clause positions are filled, signs for Boolean literals are randomly selected (the placeholder l^* is not affected by this step) as follows:

1. All positions allocated uniformly, either through uniform or after privileged allocation, are uniformly randomly set to the Boolean literal or its negation.
2. All privileged Boolean atoms allocated are *jointly* set a literal sign. That is, *all* atom allocations share one same literal sign, which is negated with probability 0.5. This ensures that privileged variables have substantial impact on the overall DNF model integral, and do not create mutually exclusive clauses.

This allocation scheme is based on earlier work [27], which generates formulas for neural weighted DNF counting. An allocation of balls to boxes, mapped to DNF clause positions, is shown in Example 8.

Example 8. Consider the earlier configuration where $m = n = 3, k = 3$, and $W = 3$. Then, an example atom allocation, generated uniformly, is:

$$(p_0 \wedge p_2 \wedge \neg p_1) \vee (l^* \wedge p_2 \wedge l^*) \vee (p_1 \wedge l^* \wedge l^*),$$

where, as before, l^* is a placeholder such that, at every location it appears, an LRA constraint is subsequently generated *independently*. An example privileged allocation, with privileged Boolean variable p_0 , is:

$$(p_0 \wedge p_2 \wedge \neg p_1) \vee (l^* \wedge p_0 \wedge l^*) \vee (p_1 \wedge l^* \wedge p_0).$$

Observe that p_0 is a positive literal in all its appearances, by construction.

It now remains to replace all LRA placeholder l^* appearances with concrete constraints so as to yield a non-empty convex body at the *clause* level. For a given DNF clause c , this is done as follows:

1. First, a random point z in \mathbb{R}^n is uniformly sampled, such that all generated constraints in c must be satisfied by z . This ensures that the convex polytope defined by c is non-empty.
2. For every placeholder: (i) randomly select a subset S of $Geom(0.5)$ real variables, where $Geom$ denotes the geometric distribution, (ii) sample $w_S \in \mathbb{R}^{|S|}$ weights for these variables from Gaussian distributions and (iii) sample a random Gaussian value v and set the linear constraint $w_S \cdot S \leq v$.
3. If z satisfies $w_S \cdot S \leq v$, then $w_S \cdot S \leq v$ is added to c , irrespective of the original placeholder sign. Otherwise, $w_S \cdot S \geq v$, which z must then satisfy, is added.

Example 9. We again consider the earlier example:

$$(p_0 \wedge p_2 \wedge \neg p_1) \vee (l^* \wedge p_2 \wedge l^*) \vee (p_1 \wedge l^* \wedge l^*).$$

Then, an example LRA constraint generation yields points $(0, 0, 0)$ and $(1, 1, 1)$ for the second and third clauses respectively, and the following constraints are randomly generated:

$$(p_0 \wedge p_2 \wedge \neg p_1) \vee (x_2 - x_0 \geq -1 \wedge p_2 \wedge x_1 + x_2 \leq 1) \vee (p_1 \wedge x_0 + 3x_2 \leq 6 \wedge x_0 - 3.5x_1 \leq 1),$$

where $x_2 - x_0 \geq -1$ is negated to be satisfied by $(0, 0, 0)$.

² In the balls and boxes problem, putting a balls in b boxes such that no box is empty can be done in $\binom{a-1}{b-1}$ ways. Our number of configurations is therefore a direct application of this formula.

4.2. Experimental setup

In our experiments, we bound all real variables in \mathbf{X} such that $\forall x \in \mathbb{R}^n, 0 \leq x \leq 10$. This ensures a bounded domain of integration, and that integrals for a finite-valued weight function w are finite-valued. In terms of w , we opt for *polynomial* functions w_x , given their simplicity and popularity for evaluation in the WMI literature, and given the ease of generating concave polynomial functions, in keeping with our theoretical requirement.

Concretely, we generate a polynomial function w_x consisting of a sum of up to 4 terms. Within this polynomial, every non-constant term consists of a coefficient, which is a uniformly randomly sampled integer in $\{1, \dots, 10\}$, as well as a degree randomly chosen from the distribution $Geom(0.6)$ and upper-bounded by 5. This degree is then uniformly randomly allocated to one or more distinct real variables. For instance, a degree of 5 can be allocated as $x_1^3 x_2^2$. Next, the coefficients of terms with degree of 2 or higher are made negative to maintain concavity, and finally a constant is added to the polynomial to ensure the positivity of the weight function over the defined real domain.

For the volume oracle VOLUME within CLAUSEWEIGHT, we use LattE [28], an exact volume computation tool. Though this theoretically is not scalable given the complexity of the algorithm, it is viable in practice for multiple reasons. First of all, LattE, despite being exact, natively supports polynomial weight functions, and performs reliably in practice for smaller-scale formulas compared with approximate techniques [29]. Second, we use LattE in a more optimized fashion, which eliminates all variables in \mathbf{X} not appearing in the LRA constraints of a given clause, or in w_x , which minimizes computation time. More specifically, we optimize our use of LattE by separately (trivially) integrating over variables not appearing in a clause or a term of w , and only running LattE over the appearing variables.

This optimization is effective in practice for two reasons. First, the number of real variables appearing in a clause is small in expectation, at around $2W$ in the worst case of an all-LRA clause, and W for an even distribution of Boolean and LRA atoms, and thus the dimensionality of the polytope space is small for computation purposes. Second, the modularity of the DNF representation breaks down an n -dimensional problem into k smaller sub-problems, which can be solved more efficiently. Observe that this is not the case with CNF representations, as the outer conjunctions prevent an efficient computation of smaller sub-problems. By contrast, the clauses in a DNF formula are not entangled, as each clause defines an individual lower-dimensional polytope, and these are combined with the outer disjunction. Hence, DNF representations allow for significantly more efficient division of the input problem relative to CNF.

For the oracle SAMPLE used within CONVEXBODYSAMPLER, we use hit-and-run [18], with a constant factor 10^3 used to compute the number of walk iterations, as is standard in practice. It is *unknown* whether this constant preserves theoretical guarantees, but it is widely used given that the best-known theoretical constant factors for hit-and-run are loose and prohibitively large (i.e., 10^{30}) [30], and because this value allows for a reasonable trade-off between accuracy and efficiency. We investigate the accuracy of this setup in detail with a dedicated experiment in Section 4.4, where we compare the outputs of our APPROXWMI setup against those produced by an exact solver on small DNF instances. Finally, we run all experiments with a timeout of 5000 seconds, on a server with a Haswell 5-2640v3, 2.60 GHz CPU and 12 GB of RAM.

4.3. Runtime experiments

To evaluate the runtime of APPROXUNION, we produce a dataset consisting of 160 DNF formulas, resulting from 40 distinct configurations, each represented by 4 DNF formulas. More specifically, we jointly set the value for m and n to values from 50 to 500 inclusive, in increments of 50, and thus the total number of variables is between 100 and 1K, and set width W to values from the set $\{3, 5, 8, 13\}$, to ensure a comprehensive coverage of all width ranges. Then, we run APPROXWMI on each formula using 4 distinct (ϵ, δ) settings: (0.15, 0.05), (0.20, 0.10), (0.25, 0.15), and (0.35, 0.25), and compute the mean runtime across 5 runs per setting. These error and confidence values are chosen in keeping with realistic practical targets, with 0.35 representing a reasonable loose error requirement, and 0.15 a common tighter requirement, particularly for larger instances.

All APPROXWMI running times with respect to the total number of variables $m + n$, clause width W , and the target error ϵ and confidence δ are presented in Fig. 1. We see that APPROXWMI performs strongly, and solves DNF instances with up to 1K variables within 5000 seconds over all W for $\epsilon = 0.35$ and $\delta = 0.25$. In fact, we also observe that, for instances with 1K variables having $W = 5, 8, 13$, all run within 2000 seconds, for $(\epsilon, \delta) = (0.25, 0.15)$, and that the increase in running time relative to the (0.35, 0.25) setting for all widths is minor, at around 40%, which is far smaller than the theoretically expected increase of $\frac{0.35^6}{0.25^6} \approx 7.5$. This is due to the efficiency of the practical setup of SAMPLE relative to tighter ϵ_S and δ_S , which alleviates the impact of theoretical higher negative powers of ϵ (implicitly ϵ_S) derived in Section 3.3. More concretely, increasing the sampling iterations (due to tighter bounds) within the current hit-and-run implementation incurs a minimal additional overhead given the commonly applied constant factor and scale of our experimental instances, as samples can easily be parallelized.

For tighter ϵ and δ , the system maintains high performance, despite the increase in T and the theoretical complexity of APPROXWMI, which involves high negative powers of ϵ . Indeed, with $\epsilon = 0.15$ and $\delta = 0.05$, all instances with widths 8 and 13 finish within ~ 2000 seconds, whereas large instances of width 3 and $m + n \geq 500$, and width 5, $m + n \geq 800$, time out. Furthermore, all experiments at widths 5, 8, and 13 terminate within the time limit for $\epsilon = 0.2$, $\delta = 0.1$, with only width 3 experiments timing out for $m + n \geq 700$. These results highlight the scalability of APPROXWMI, deployed with

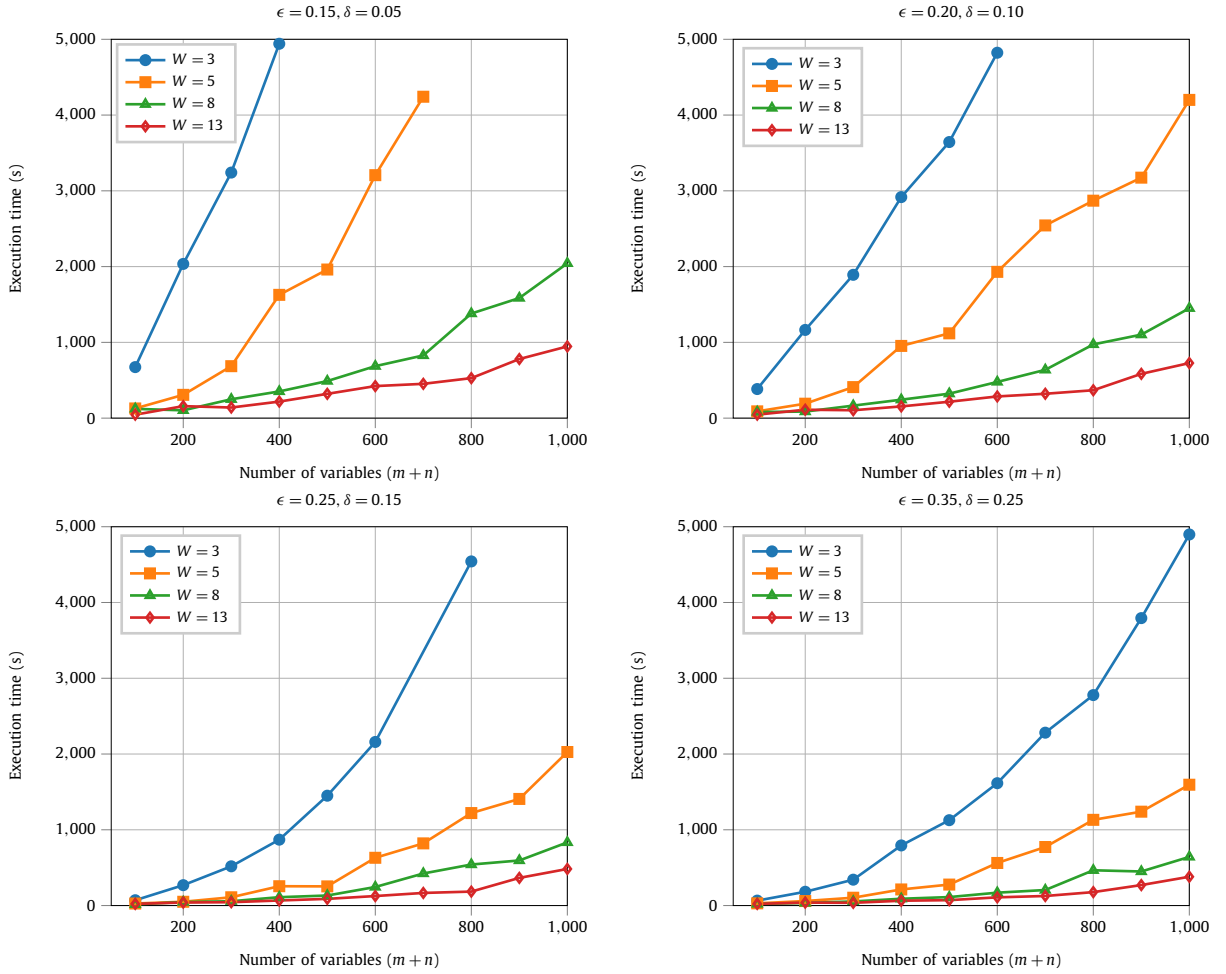


Fig. 1. Runtime results for APPROXWMI relative to different ϵ, δ . For $\epsilon = 0.35, \delta = 0.25$, all instances, including those with 1K variables, terminate within 5000 seconds, and higher-width instances ($W = 8, 13$) all terminate within ~ 2000 seconds for all ϵ, δ . (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

standard sampling and volume computation tools, for computing approximations to WMI, even relative to tighter error and confidence targets, and on large instances involving 1K variables.

Somewhat unintuitively, our experiments show that system performance worsens as W decreases, irrespective of error and confidence targets. In fact, we can see that, for DNF instances with $W = 3$, APPROXWMI requires almost triple the time compared to instances with higher W across all (ϵ, δ) configurations. Though this behavior is surprising, it can be attributed to an increased number of sampling replacements and calls within APPROXWMI. Indeed, for smaller widths, it is likelier that a call to EVALUATE will yield “True” since there are less constraints and Boolean literals to satisfy. As a result, further calls to SAMPLE, which runs in time $O^*(m + n^3)$, will be required. Though the implementation for SAMPLE is efficient relative to ϵ_S , as mentioned earlier, it still remains an expensive component in the APPROXWMI sampling iteration, and has an overhead when called. For comparison, a membership check runs in $O(m + Wn)$, which is very small in our setting as W is small. Thus, the increased number of calls to sample due to likelier replacements at small W configurations, combined with the typically large number of sampling iterations T , impose a significant computational overhead on APPROXWMI. This behavior also justifies the improved performance with increased width observed in Fig. 1, as higher-width clauses are less likely to cause replacements, which decreases the expected number of calls to SAMPLE and reduces running time.

Overall, our results confirm our intuitions about APPROXWMI: APPROXWMI indeed scales to large instances having up to 1K total variables, and can provide approximations in a reasonable time using common oracle choices, even for tight ϵ and δ . Our experiments also highlight that calls to CLAUSEWEIGHT, though invoking an exact volume computation tool, are not a bottleneck to system performance. In fact, for any instance in our evaluation, CLAUSEWEIGHT calls require at most 450 seconds in total to run. This also aligns with our experimental setup choices and intuitions, as the bounded width W used in our experiments and the modularity of the DNF structure allow for lower-dimensional volume computations that LattE can perform very efficiently. Finally, our results show that the main performance bottleneck for APPROXWMI is the number

Table 1

Frequency of APPROXWMI runs ($\epsilon = 0.15$, $\delta = 0.05$) satisfying error bounds relative to different hit-and-run constant factors. Even with a constant value of 100, APPROXWMI comfortably satisfies the approximation confidence bound (0.95).

Hit-and-run constant	10	100	1000
Within ϵ frequency	0.635	0.986	0.998

of calls to SAMPLE. This is particularly evident from the high dependence of running time on width W , and shows that, even with a reduced constant factor, convex body sampling is a highly costly operation when called repeatedly.

We conclude our discussion with an added theoretical note that further supports our choice of an exact VOLUME oracle: in addition to competitive practical performance at smaller scale, exact VOLUME oracles set $\epsilon_V, \epsilon_P, \delta_V, \delta_P = 0$, which enables the derivation of looser error and confidence bounds for SAMPLE. More concretely, replacing these values in the derivation of Lemma 3 of APPROXUNION yields the improved bound $\epsilon_S < \frac{\epsilon^2}{8k}$, plugging in $\delta_P = 0$ into the proof of Corollary 2 improves the bound by a factor of 1.5, yielding $\delta_S \leq \frac{\delta}{1518 \ln(\frac{8}{\delta}) \frac{k}{\epsilon^2}}$. Though these new bounds are only constant factor improvements over

the case with approximate VOLUME, these factors are rather large, at 5.875 and 1.5 respectively, and amplify when exponentiated, particularly for ϵ . Thus, they allow for significant practical gains, which were very beneficial in our experiments, all while preserving the theoretical guarantees of standard APPROXWMI. Moreover, the use of an exact volume computation tool also preserves the polynomial running time of APPROXWMI given bounded width and a bounded number of variables in all LRA constraints. Though these assumptions are somewhat restrictive, they are typically encountered in practice: clause widths typically do not exceed 100, and LRA constraints do not include large sets of variables. Hence, the use of an exact VOLUME oracle is justified theoretically, both in terms of reducing the overhead of sampling, and for its effectiveness in practice given the boundedness of width, LRA constraints, and the modularity of DNF structures.

4.4. Accuracy experiments

APPROXWMI is an FPRAS for WMI(DNF), and provably provides approximation guarantees. However, in practice, hidden constant factors in the subroutines of APPROXWMI, such as the prohibitively large constant factor in hit-and-run (10^{30}), make a theoretically faithful implementation intractable. Given this limitation, and in line with standard practice, we therefore used a lower constant factor of 10^3 for hit-and-run in our setup. However, this constant is not known to preserve the theoretical guarantees of hit-and-run. Therefore, a natural question is whether this scaling down of the hit-and-run constant nonetheless achieves performance that still matches the FPRAS guarantees, and thus whether this setup is reliable in practice.

To address this question, we generate an additional dataset consisting of smaller formulas, which can tractably be solved exactly. More specifically, this dataset consists of 20 formulas, each having $m = n = 20$, and with widths evenly distributed among the set $\{2, 3, 5, 8\}$. Furthermore, this dataset only uses weight functions where the real component w_x is restricted to be linear, and to involve at most three variables, so as to best match the performance profile of exact solvers.

To solve dataset instances exactly, we use a compilation tool based on extended algebraic decision diagrams (XADD) [31], offered as part of the PyWMI library [32]. Then, we set up APPROXWMI with $\epsilon = 0.15$ and $\delta = 0.05$, and run it 1000 times with 3 different constant factors for hit-and-run: 10, 100, and 1000 (as in the earlier experiment). Finally, we check whether, for every hit-and-run constant factor, the WMI returned by APPROXWMI is within ϵ relative to the exact solution with a frequency exceeding $1 - \delta = 0.95$.

The frequency results for this experiment are shown in Table 1, with values exceeding 0.95 shown in bold. First, we observe that, at the default constant value of 1000, APPROXWMI significantly oversatisfies the theoretical confidence guarantees, and near-perfectly makes predictions within the approximation error bounds. Surprisingly, APPROXWMI continues to oversatisfy the confidence bound even with a hit-and-run constant of 100. In fact, APPROXWMI only underperforms when the constant is set to 10, a very low and unrealistic setting. This suggests that hit-and-run can indeed make good predictions with a reasonable choice of constant, and that the current setup of APPROXWMI, with hit-and-run constant factor 1000, comfortably achieves high-quality performance that aligns well with the theoretical guarantees.

All in all, these are encouraging findings, which further motivate the practical use of APPROXWMI, and corroborate the empirical evidence for the looseness of the theoretical hit-and-run constant. In particular, these results, combined with the earlier runtime performance of APPROXWMI, highlight that this APPROXWMI setup performs reliably, efficiently, and can scale to large instances involving up to 1K variables.

5. Related work

WMC is a unifying framework for probabilistic inference across prominent probabilistic models: inference in probabilistic graphical models [2] reduces to WMC(CNF) [11,12], and similar reductions of inference tasks to WMC exist for Markov Logic Networks (MLNs) [33], probabilistic logic programming [34], and more generally, for relational models [35]. However, WMC

cannot capture hybrid probabilistic models involving both continuous and discrete variables, which occur in various contexts, including hybrid MLNs [36], and hybrid Bayesian networks [37,38]. WMI is proposed as a unifying inference framework for these hybrid models [7], and has been addressed in recent years both with exact and approximate solving techniques [39].

In terms of computational complexity, both WMI and WMC are #P-hard [13], so are highly intractable for exact solving, as the class $P^{\#P}$ contains the entire polynomial hierarchy [40]. Unfortunately, this intractability even extends to restricted settings of both these problems. For example, DNF counting remains #P-hard, even on positive, partitioned DNF formulas with clause width at most 2 [41]. Furthermore, WMI can be tractably solved if and only if formulas have a primal graph that is a tree with diameter logarithmic in the number of nodes [42]. In a primal graph, formula variables are represented as nodes, and edges between these nodes appear when the variables co-appear in a same clause. Thus, this result shows that only branching tree-shaped dependencies between variables can be tractably solved exactly for WMI, and that even simple linear chain dependencies make formulas hard to solve.

Despite the intractability of exact solving, many general-purpose exact solvers, based on several optimizations, have been developed. These solvers exploit formula structure and perform substantial pre-processing to reduce the computation load of WMI. For instance, a tool based on SMT *predicate abstraction* techniques [8] has been proposed to reduce the number of models. Predicate abstraction introduces a set of predicates ψ and then abstracts away solutions of the original formula ϕ by iterating over truth assignments to ψ and considering solutions that are shared with ϕ . Hence, ψ , typically designed to be relevant to ϕ , that is, having predicates whose truth or falsehood affects ϕ , allows to significantly reduce the search space for the SMT solver. Indeed, predicate abstraction reduces the original exponential search over the atoms of ϕ to a more manageable (but still exponential) search over ψ , where $|\psi| \ll \text{atoms}(\mathbf{X}, \mathbf{V})$. This approach also supports a rich class of weight functions, allowing conditional dependencies and case splits.

In addition to predicate abstraction, Symbo [20] uses knowledge compilation to more efficiently perform WMI. More specifically, Symbo compiles ϕ into a d-DNNF representation, which can subsequently be run in polynomial time. Though the compilation is itself computationally expensive, it leads to substantial computational speedup when integrals over ϕ need to be computed multiple times. Symbo is inspired by knowledge compilation approaches for WMC, and performs compilation by casting WMI as an algebraic model counting (AMC) task. With the AMC formulation, Symbo then defines the corresponding literal labels and symbolic weights, which a symbolic integration tool then uses, along with the compiled d-DNNF ϕ , to compute WMI. Symbo uses the common factorization assumption of Equation (2), but additionally supports continuous variables using the theory of non-linear real arithmetic (NRA), which supports sums of *rational powers* of real variables, as opposed to simple linear combinations of variables. Furthermore, Kolb et al. [31] use compilation based on extended algebraic decision diagrams (XADDs) to tackle WMI with LRA atoms, and where w is a piece-wise polynomial function. XADD enables parametrized (i.e., partial) solving of WMI, and is combined with symbolic dynamic programming (SDP) to integrate over the Boolean-real domain and perform partial computation caching, enabling more efficient exact WMI solving.

Finally, exact lower and upper solution bounds, as opposed to the exact solution, are computed for WMI, based on hyper-rectangular decomposition and orthogonal transformations [43]. Intuitively, hyper-rectangular decomposition incompletely covers the integration domain using sets of hyper-rectangles, such that the total integral over these hyper-rectangles yields a lower bound for WMI. Analogously, upper bounds are computed by covering the negation of the domain of integration and subtracting the integral from the total domain integral. However, the domain of integration can be hard to cover efficiently using hyper-rectangles alone, if, for instance, this domain is not axis-aligned. Therefore, the approach proposes orthogonal transformations, which transform the input formula using rotations and Pythagorean triples to another representation that is more efficiently decomposable. This transformation is shown to preserve weights and the correctness of the problem setting for variables following a Gaussian density function. This method is therefore more efficient than standard WMI solving, as integration over hyper-rectangles is simple, yields exact bounds, and can be made arbitrarily accurate by increasing the number of used hyper-rectangles.

Parallel to exact solvers, *approximate* methods have been developed for WMI to circumvent its intractability in the exact setting. A common such approximation method is *hashing*. At a high level, hashing methods partition the solution space into disjoint, smaller, partitions where counting (resp., integration) is tractable, solve the problem over these easier partitions, then leverage their results to compute an approximation, with guarantees, for the overall model count (resp., integral). Hashing-based methods are popular for WMC [44], and were lifted to WMI [45] following its formulation [7]. More concretely, WMI is approached using *propositional abstraction*, such that WMI statements are directly cast into WMC. With this representation, it is then shown that universal hashing techniques used for WMC also apply to WMI, and therefore, that an NP-oracle can also effectively partition the solution space for hybrid domains. Hence, this abstraction-based hashing tool for WMI can compute probabilistic approximations, with guarantees, using a polynomial number of calls to an NP oracle.

In addition to hashing, knowledge compilation is also used to support WMI approximation. In particular, Sampo [20] extends Symbo to overcome the limitations of symbolic integration. More concretely, it uses Monte Carlo sampling to compute approximate integrals, and thus can apply to general WMI theories such as real arithmetic (RA), where certain integrals cannot be symbolically computed. Sampo performs a number of sampling iterations, in which random variables are sampled and fed into the compiled representation of ϕ , analogously to Symbo, to yield a weight. Then, the average of these weights yields an approximate density. Sampo therefore approximates WMI with probabilistic guarantees.

Beyond Monte Carlo sampling, Markov Chain Monte Carlo (MCMC) methods have been applied to WMI, but such approaches do not provide any guarantees. The only exception for this is a tool for #SMT [46], the unweighted case of WMI. This tool approximates the solution for #SMT by calling a SAT solver following a randomized algorithm, analogously to MCMC approaches for WMC(CNF), and produces probabilistic guarantees with polynomially many SAT calls [47].

Finally, WMI has recently been approximated with the help of *relaxation* schemes. More specifically, ReCoIn [42], which is short for “Relax, compensate and then integrate”, *relaxes* a given WMI instance by removing some of its constraints to yield a “simpler” formula which can be tractably solved exactly. Then, it *compensates* for this relaxation by introducing new auxiliary constraints to the relaxed formula. These new constraints are added to maintain semantic similarity of the relaxed formula relative to the original formula, but are also formulated to keep the relaxed formula tractable. Following relaxation and compensation, the resulting formula is then solved exactly (the *integration* operation), and its solution returned as the approximate WMI. However, the modification of input instances can lead to arbitrarily dissimilar instances, and thus ReCoIn does not provide guarantees with its approximations.

To the best of our knowledge, there is no dedicated study for WMI(DNF), despite WMC(DNF) being extensively studied, partly motivated by the rich literature and research on probabilistic data management [5]. Indeed, query answering in probabilistic databases reduces to WMC(DNF) as every conjunctive query is equivalent to a DNF via its lineage representation and this correspondence extends to different probabilistic data models [48] and to special classes of ontology-mediated probabilistic query answering [6,49]. This has led to a number of tools and algorithms from KLM [14] to hashing-based techniques [50], and recently to neural model counting approaches [27]. However, these tools, by construction, cannot handle extended data models which also include continuous distributions. Such data models include Monte Carlo Databases (MCDBs) [51], and the system PIP [52], which extends MCDBs to efficiently query probabilistic data defined over hybrid distributions. Abstracting away subtle technical differences, MCDBs and PIP both introduce approximate inference algorithms. However, unlike our approach, these approaches do not provide guarantees. Therefore, our theoretical study of WMI(DNF), and our proposed FPRAS algorithms for this setting, additionally serve as a unifying framework and perspective for a large class of data models, spanning both standard probabilistic databases and hybrid data models.

6. Summary and outlook

In this work, we studied weighted model integration on DNF structures. First, we presented APPROXWMI, an FPRAS for WMI(DNF) given a concave and factorized weight function w , which is based on FPRAS oracles for volume computation and point sampling over convex bodies. Then, we relaxed the factorization assumption over w , and extended APPROXWMI to a new FPRAS APPROXWMI_D, which uses Monte-Carlo sampling to capture dependencies between the Boolean and real sub-domains. Our FPRAS results for WMI(DNF) complement the result of WMC(DNF), and help draw a more complete picture of approximability for these problems. Furthermore, our experimental analysis further shows the potential of APPROXWMI as a scalable and reliable WMI solver over DNF formulas.

Looking forward, an interesting direction for future work is to investigate alternative approaches for approximate WMI with guarantees, such as hashing-based approaches, to reduce the need for expensive sampling operations, and thus further improve the scalability of approximate WMI(DNF). Beyond this, it is important to explore further generalizations of this work, allowing it to apply to new families of weight functions, as well as other SMT theories. We hope this work leads to a better understanding of the complexity of WMI, and stimulates further investigation and development of WMI approaches, leading to more robust WMI systems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the Alan Turing Institute under the UK EPSRC grants EP/N510129/1, EP/R013667/1, EP/L012138/1, and EP/M025268/1. Ralph Abboud is funded by the Oxford-DeepMind Graduate Scholarship and the Alun Hughes Graduate Scholarship. Experiments for this work were conducted on servers provided by the Advanced Research Computing (ARC) cluster administered by the University of Oxford (<https://doi.org/10.5281/zenodo.22558>).

References

- [1] C.P. Gomes, A. Sabharwal, B. Selman, Model counting, in: Handbook of Satisfiability, IOS Press, 2009, pp. 633–654.
- [2] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
- [3] C. Domshlak, J. Hoffmann, Probabilistic planning via heuristic forward search and weighted model counting, J. Artif. Intell. Res. 30 (1) (2007) 565–620.
- [4] L. De Raedt, A. Kimmig, H. Toivonen, ProbLog: a probabilistic prolog and its application in link discovery, in: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI), 2007, pp. 2462–2467.
- [5] D. Suciu, D. Olteanu, C. Ré, C. Koch, Probabilistic Databases, Morgan & Claypool, 2011.
- [6] S. Borgwardt, I.I. Ceylan, T. Lukasiewicz, Ontology-mediated queries for probabilistic databases, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI), 2017, pp. 1063–1069.

- [7] V. Belle, A. Passerini, G. Van Den Broeck, Probabilistic inference in hybrid domains by weighted model integration, in: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 2015, pp. 2770–2776.
- [8] P. Morettin, A. Passerini, R. Sebastiani, Advanced SMT techniques for weighted model integration, *Artif. Intell. J.* 275 (2019) 1–27.
- [9] C. Barrett, R. Sebastiani, S. Seshia, C. Tinelli, Satisfiability modulo theories, in: *Frontiers in Artificial Intelligence and Applications*, vol. 185, IOS Press, 2009, pp. 825–885, Ch. 20.
- [10] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
- [11] T. Sang, P. Bearne, H. Kautz, Performing bayesian inference by weighted model counting, in: *Proceedings of the Twentieth AAAI Conference on Artificial Intelligence (AAAI)*, 2005, pp. 475–482.
- [12] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, *Artif. Intell.* 172 (6) (2008) 772–799.
- [13] L.G. Valiant, The complexity of computing the permanent, *Theor. Comput. Sci.* 8 (2) (1979) 189–201.
- [14] R.M. Karp, M. Luby, N. Madras, Monte-Carlo approximation algorithms for enumeration problems, *J. Algorithms* 10 (3) (1989) 429–448.
- [15] D. Roth, On the hardness of approximate reasoning, *Artif. Intell. J.* 82 (1–2) (1996) 273–302.
- [16] K. Bringmann, T. Friedrich, Approximating the volume of unions and intersections of high-dimensional geometric objects, *Comput. Geom.* 43 (6–7) (2010) 601–610.
- [17] M.G. Luby, Monte-Carlo methods for estimating system reliability, Ph.D. thesis, UC Berkeley, 1983.
- [18] M. Chen, B.W. Schmeiser, General hit-and-run Monte Carlo sampling for evaluating multidimensional integrals, *Oper. Res. Lett.* 19 (4) (1996) 161–169.
- [19] R. Abboud, İ.İ. Ceylan, R. Dimitrov, On the approximability of weighted model integration on DNF structures, in: *Proceedings of 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, AAAI Press, 2020*, pp. 828–837.
- [20] P. Martires, A. Dries, L. De Raedt, Exact and approximate weighted model integration with probability density functions using knowledge compilation, in: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)*, 2019, pp. 7825–7833.
- [21] L. Lovász, S.S. Vempala, Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm, *J. Comput. Syst. Sci.* 72 (2) (2006) 392–417.
- [22] I. Bárány, Z. Füredi, Computing the volume is difficult, *Discrete Comput. Geom.* 2 (1987) 319–326.
- [23] D.L. Applegate, R. Kannan, Sampling and integration of near log-concave functions, in: *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing (STOC)*, 1991, pp. 156–163.
- [24] R. Kannan, L. Lovász, M. Simonovits, Random walks and an $O^*(n^5)$ volume algorithm for convex bodies, *Random Struct. Algorithms* 11 (1) (1997) 1–50.
- [25] Y.T. Lee, S.S. Vempala, Geodesic walks in polytopes, in: *Proceedings of the Forty-Ninth Annual ACM Symposium on Theory of Computing (STOC)*, 2017, pp. 927–940.
- [26] S. Chakraborty, D.J. Fremont, K.S. Meel, S.A. Seshia, M.Y. Vardi, Distribution-aware sampling and weighted model counting for SAT, in: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*, 2014, pp. 1722–1730.
- [27] R. Abboud, İ.İ. Ceylan, T. Lukasiewicz, Learning to reason: leveraging neural networks for approximate DNF counting, in: *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, 2020, pp. 3097–3104.
- [28] V. Baldoni, N. Berline, J.A. De Loera, B. Dutra, M. Köppe, S. Moreinis, G. Pinto, M. Vergne, J. Wu, A user's guide for LattE integrale v1.7.2, *Optimization* 22 (2).
- [29] I.Z. Emiris, V. Fisikopoulos, Practical polytope volume approximation, *ACM Trans. Math. Softw.* 44 (4) (2018) 38:1–38:21.
- [30] L. Lovász, S. Vempala, Where to start a geometric random walk, Tech. rep., Microsoft Research, 2003.
- [31] S. Kolb, M. Mladenov, S. Sanner, V. Belle, K. Kersting, Efficient symbolic integration for probabilistic inference, in: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 5031–5037.
- [32] S. Kolb, P. Morettin, P. Zuidberg Dos Martires, F. Somavilla, A. Passerini, R. Sebastiani, L. De Raedt, The pywmi framework and toolbox for probabilistic inference using weighted model integration, in: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 6530–6532.
- [33] M. Richardson, P. Domingos, Markov logic networks, *Mach. Learn.* 62 (1) (2006) 107–136.
- [34] D. Fierens, G. Van Den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, L. De Raedt, Inference and learning in probabilistic logic programs using weighted boolean formulas, *Theory Pract. Log. Program.* 15 (3) (2015) 358–401.
- [35] V. Gogate, P. Domingos, Probabilistic theorem proving, *Commun. ACM* 59 (7) (2016) 107–115.
- [36] J. Wang, P. Domingos, Hybrid Markov logic networks, in: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI)*, 2008, pp. 1106–1111.
- [37] V. Gogate, R. Dechter, Approximate inference algorithms for hybrid bayesian networks with discrete constraints, in: *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005, pp. 209–216.
- [38] S. Sanner, E. Abbasnejad, Symbolic variable elimination for discrete and continuous graphical models, in: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, 2012, pp. 1954–1960.
- [39] P. Morettin, P. Zuidberg Dos Martires, S. Kolb, A. Passerini, Hybrid probabilistic inference with logical and algebraic constraints: a survey, in: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.
- [40] S. Toda, On the computational power of PP and +P, in: *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science (FOCS)*, 1989, pp. 514–519.
- [41] J.S. Provan, M.O. Ball, The complexity of counting cuts and of computing the probability that a graph is connected, *SIAM J. Comput.* 12 (4) (1983) 777–788.
- [42] Z. Zeng, P. Morettin, F. Yan, A. Vergari, G.V. den Broeck, Probabilistic inference with algebraic constraints: theoretical limits and practical approximations, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), *Proceedings of the Thirty-Third Annual Conference on Advances in Neural Information Processing Systems (NeurIPS)*, 2020, pp. 11564–11575.
- [43] D. Merrell, A. Albarghouthi, L. D'Antoni, Weighted model integration with orthogonal transformations, in: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, 2017, pp. 4610–4616.
- [44] S. Chakraborty, K.S. Meel, M.Y. Vardi, A scalable approximate model counter, in: *Proceedings of the Nineteenth International Conference on the Principles and Practice of Constraint Programming (CP)*, 2013, pp. 200–216.
- [45] V. Belle, G.V. den Broeck, A. Passerini, Hashing-based approximate probabilistic inference in hybrid domains, in: *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015, pp. 141–150.
- [46] D. Chistikov, R. Dimitrova, R. Majumdar, Approximate counting in SMT and value estimation for probabilistic programs, *Acta Inform.* 54 (8) (2017) 729–764.
- [47] M.R. Jerrum, L.G. Valiant, V.V. Vazirani, Random generation of combinatorial structures from a uniform distribution, *Theor. Comput. Sci.* 43 (2–3) (1986) 169–188.
- [48] İ.İ. Ceylan, A. Darwiche, G. Van den Broeck, Open-world probabilistic databases: semantics, algorithms, complexity, *Artif. Intell.* 295 (2021) 103474.
- [49] S. Borgwardt, İ.İ. Ceylan, T. Lukasiewicz, Ontology-mediated query answering over log-linear probabilistic data, in: *Proceedings of the 33rd National Conference on Artificial Intelligence, AAAI Press, 2019*, pp. 2711–2718.
- [50] K.S. Meel, A.A. Shrotri, M.Y. Vardi, On hashing-based approaches to approximate DNF-counting, in: *Proceedings of the Thirty-Seventh IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2017, pp. 41:1–41:14.

- [51] R. Jampani, F. Xu, M. Wu, L.L. Perez, C. Jermaine, P.J. Haas, MCDB: a Monte Carlo approach to managing uncertain data, in: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2008, pp. 687–700.
- [52] O. Kennedy, C. Koch, PIP: a database system for great and small expectations, in: *Proceedings of the Twenty-Sixth International Conference on Data Engineering (ICDE)*, 2010, pp. 157–168.