The *pywmi* Framework and Toolbox for Probabilistic Inference using Weighted Model Integration

Samuel Kolb[1 *], **Paolo Morettin**[2], **Pedro Zuidberg**[1], **Francesco Sommavilla**[2], **Andrea Passerini**[2], **Roberto Sebastiani**[2] and **Luc De Raedt**[1]

[1]KU Leuven

[2]University of Trento

{samuel.kolb, pedro.zuidbergdosmartires, luc.deraedt}@cs.kuleuven.be,
{paolo.morettin, andrea.passerini, roberto.sebastiani}@unitn.it
francesco.sommavilla@studenti.unitn.it

Abstract

Weighted Model Integration (WMI) is a popular technique for probabilistic inference that extends Weighted Model Counting (WMC) – the standard inference technique for inference in discrete domains – to domains with both discrete and continuous variables. However, existing WMI solvers each have different interfaces and use different formats for representing WMI problems. Therefore, we introduce *pywmi*, an open source framework and toolbox for probabilistic inference using WMI, to address these shortcomings. Crucially, *pywmi* fixes a common internal format for WMI problems and introduces a common interface for WMI solvers. To assist users in modeling WMI problems, *pywmi* introduces modeling languages based on SMT-LIB.v2 or MiniZinc and parsers for these languages. To assist users in comparing WMI solvers, *pywmi* includes implementations of several state-of-the-art solvers, a fast approximate WMI solver, and a command-line interface to solve WMI problems. Finally, to assist developers in implementing new solvers, *pywmi* provides Python implementations of commonly used subroutines.

1 Modeling WMI problems

In a nutshell, WMI [Belle *et al.*, 2015] traces back to SAT, the problem of deciding if a Boolean formula is satisfiable. #SAT builds on SAT but answers the question of *how many* models satisfy a formula and WMC extends #SAT by allowing models to be weighted. Like SMT(LRA) extends SAT to answer satisfiability for logical formulas with linear inequalities over real variables, WMI extends WMC to integrate over (instead of summing) over the (possibily infinitely many) weighted models of an SMT(LRA) formula. A WMI problem consists of a *support* ϕ, an SMT(LRA) formula that describes all feasible worlds, a *weight function* $w : \mathbb{R}^r \times \mathbb{B}^b \mapsto \mathbb{R}$ that assigns a weight to every possible world and a set of queries, every query being an SMT(LRA) formula whose probability we want to compute. Worlds are value assignments to the problem variables. Internally, *pywmi* represents WMI problems as tuples $\langle \mathrm{dom}, \phi, w, Q \rangle$. The *domain* dom contains problem variables, their types and, optionally, the valid ranges for numeric variables (e.g., $x \in [0, 1]$). Both SMT(LRA) formulas ($\phi, q \in Q$) and weight functions (w) are represented as Abstract Syntax Trees (ASTs) using the implementation provided by the PySMT library [Gario and Micheli, 2015].

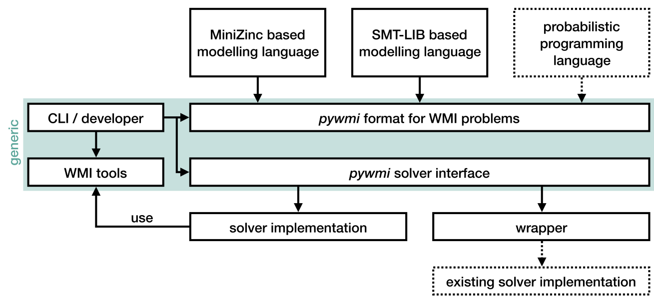

Figure 1: An overview of the *pywmi* framework. The language decouples specific modeling languages (top) and solvers (bottom) using a generic format and solver interface, as well as providing command-line interface and tools for solvers.

The internal representation is exposed to developers and, for example, probabilistic programming languages can interface with it. For example, a language such as Problog [Dries *et al.*, 2015] produces ground programs to be solved using WMC, an extension with continuous variables can solve ground programs using WMI instead. However, for end-users, *pywmi* also provides two modeling languages for WMI problems that can be directly parsed to the *pywmi* representation (see Figure 1 for an overview). Both formats can be used to compactly encode a WMI problem, differing in the syntax used for the expressions (inspired by MiniZinc [Nethercote *et al.*, 2007] or SMT-LIB.v2 [Barrett *et al.*, 2010]).

As an example, consider modelling a factory that produces banana and chocolate flavoured ice cream. Figure 2 shows a toy model of the factory production in the MiniZinc-inspired syntax. Variables b and c represent the amount of banana and chocolate ice cream produced in a work day by the factory, while $weekday$ is a Boolean variable distinguishing between

*Contact Author

```
var float : b; % banana-flavoured ice cream
var float : c; % chocolate-flavoured ice cream
var bool : weekday; % weekday or weekend

par float : CAPACITY = 10.0;

% production is nonnegative
constraint (b >= 0) /\ (c >= 0);

% production cannot exceed the storage capacity
constraint (b + c < CAPACITY);

weight : (if weekday then 5/7 else 2/7) *
         (if weekday
             then (0.004 * b + 0.002 * c)
             else (0.002 * b + 0.004 * c))

% queries and evidence
constraint (not weekday);
query (b >= 2*c);
query (b >= 9.0) \/ (c > 9.0);
```

Figure 2: The ice cream production example problem encoded using the MiniZinc-flavored WMI modeling language.

weekdays and weekends. Intuitively, the production is determined by physical constraints, i.e., being non-negative and not exceeding the capacity of the factory, and market trends causing the preference over flavours to produce to be inverted in the weekends. In this toy example, the density functions combines the probability of weekday vs weekend and a linear combination of the amount of flavours produced, conditioned on $weekday$, that increases with the amount of ice cream produced. The WMI file can also contain evidence, e.g., conditioning on weekend days, and a list of queries to, for example, compute the probability that the banana ice cream production at least doubles the chocolate one (xxx), or the probability of having a particularly high production of any flavour (yyy).

2 Framework for WMI Solvers

Given a WMI problem in a standardized format, a WMI solver is employed to calculate weighted model integrals, most frequently, with the aim to compute query probabilities. The core interface for WMI solvers in *pywmi* consists of two methods: 1) computing a weighted model integral, given a domain, support and weight functions; and 2) computing the probability of a set of queries, given a domain, support, weight function and a set of queries. As computing the probabilities of a set of queries can be reduced to a set of weighted model integral computations, *pywmi* offers a default implementation. However, solvers that can only compute query probabilities or offer more efficient ways to compute query probabilities (e.g., using knowledge compilation [Kolb *et al.*, 2018]), can override the default implementation.

The solving interface makes it possible to use any supported solver to solve any WMI problem in the common representation format. For users, *pywmi* offers a command-line interface to call or compare different solvers for a problem. Additionally, it can assist with converting file formats, installing new solvers or visualizing WMI problems. For developers, *pywmi* offers a growing amount of common functionality that might be useful to manipulating WMI problems or implementing a new solver, such as, easy-to-use data-structures for WMI problems and domains, wrappers around numeric integration software (e.g., Latte [De Loera *et al.*, 2013]) and symbolic computation systems (e.g., PSI [Gehr *et al.*, 2016]) that make it easy to write backend-agnostic solvers, and subroutines for computing with PySMT ASTs, linear inequalities, polynomials, sampling from WMI densities, evaluating points w.r.t. to their feasibility and density, etc.

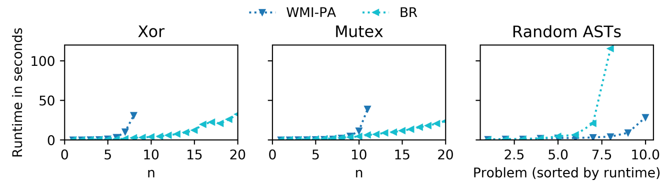

Figure 3: The performance of the solvers varies for different types of problems. For highly structured problems the BR solver performs better, while for less structured problems the PA solver is faster.

3 Comparing WMI Solvers

As mentioned above, *pywmi* allows the performance of different solvers to be compared. Aside from a common interface, however, *pywmi* also provides implementations for many state-of-the-art algorithms. These implementations either wrap around existing tools or libraries, or are (re-)implemented natively in *pywmi*. For solvers with external dependencies *pywmi* tries to make the installation as easy as possible by automating installations or providing detailed instructions. The supported state-of-the-art WMI solvers are: WMI-PA [Morettin *et al.*, 2019], XADD-based path-enumeration [Sanner *et al.*, 2011] or bound-resolution (BR) algorithms [Kolb *et al.*, 2018], symbolic SDD-based solver Symbo [Zuidberg Dos Martires *et al.*, 2019], PRAiSE [De Salvo Braz *et al.*, 2016][1]. Additionally, *pywmi* includes a native implementation of XADDs and the BR algorithm, a fast approximate solver based on rejection-sampling and new WMI solvers that are currently being developed using *pywmi*[2].

Using *pywmi*, we can now easily compare how different solvers behave on different problems. For example, we can compare how two state-of-the-art solvers (WMI-PA and BR) compare on some of the problems they introduce: highly structured *xor* and *mutex* problems with sparse inequalities (introduced alongside the BR solver), and synthetic WMI problems with random ASTs as support and weight function (using the WMI-PA problem generator). Running both solvers on these problems shows that the search-based WMI-PA solver and its numeric integration software outperform the BR solver for the WMI problems with random ASTs, while the compilation-based BR solver better exploits the structure of the *xor* and *mutex* problems (Figure 3).

[1]through the WMI-PA wrapper, without support for raw WMI computations – only query probabilities

[2]these will remain private until their publication

References

[Barrett *et al.*, 2010] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[Belle *et al.*, 2015] Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. Probabilistic Inference in Hybrid Domains by Weighted Model Integration. In *IJCAI*, pages 2770–2776, 2015.

[De Loera *et al.*, 2013] Jesús A De Loera, Brandon Dutra, Matthias Koeppe, Stanislav Moreinis, Gregory Pinto, and Jianqiu Wu. Software for exact integration of polynomials over polyhedra. *Computational Geometry*, 46(3):232–252, 2013.

[De Salvo Braz *et al.*, 2016] Rodrigo De Salvo Braz, Ciaran O'Reilly, Vibhav Gogate, and Rina Dechter. Probabilistic inference modulo theories. In *IJCAI*, pages 3591–3599, 2016.

[Dries *et al.*, 2015] Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens, Guy Van den Broeck, Jonas Vlasselaer, and Luc De Raedt. Problog2: Probabilistic logic programming. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 312–315. Springer, 2015.

[Gario and Micheli, 2015] Marco Gario and Andrea Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop 2015*, 2015.

[Gehr *et al.*, 2016] Timon Gehr, Sasa Misailovic, and Martin Vechev. PSI: Exact Symbolic Inference for Probabilistic Programs. In *CAV*, pages 62–83. Springer, 2016.

[Kolb *et al.*, 2018] Samuel Kolb, Martin Mladenov, Scott Sanner, Vaishak Belle, and Kristian Kersting. Efficient Symbolic Integration for Probabilistic Inference. In *IJCAI*, pages 5031–5037, 2018.

[Morettin *et al.*, 2019] Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. Advanced smt techniques for weighted model integration. *Artificial Intelligence*, 2019.

[Nethercote *et al.*, 2007] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[Sanner *et al.*, 2011] Scott Sanner, Karina Valdivia Delgado, and Leliane Nunes De Barros. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In *UAI*, 2011.

[Zuidberg Dos Martires *et al.*, 2019] Pedro Zuidberg Dos Martires, Anton Dries, and Luc De Raedt. Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation. In *AAAI*, 2019.