Summer Internship Report

Internship Period: May 2025 – August 2025

Jiahao Xia

August 22, 2025

Contents

Abstract

Probabilistic Logic Programs (PLPs) extend logic programs by annotating facts with probabilities. Problog, a popular PLP engine, suffers from several performance bottlenecks. These include interpreter-based execution, the lack of optimized Binary Decision Diagram (BDD) variable ordering and Sentential Decision Diagram (SDD) vtree ordering, and insufficient reuse of shared intermediate results across query tasks (e.g., marginal probability computation). We extend Soufflé, a Datalog synthesizer, to support probabilistic semantics, replacing Problog's interpreter with compiled execution and integrating a dynamic BDD reordering strategy. Experimental results show that Soufflé outperforms Problog, especially on large-scale instances.

1 Introduction

Probabilistic Logic Programs (PLPs) combine probabilistic reasoning with logic programming, enabling the modeling of uncertainty in domains such as bioinformatics, artificial intelligence, and software analysis. Among existing PLP systems, Problog is widely used.

However, Problog faces several scalability challenges. First, interpreter-based execution introduces overhead for large-scale workloads. Second, while it employs Binary Decision Diagrams (BDDs) and Sentential Decision Diagrams (SDDs) for Weighted Model Counting (WMC), it does not optimize BDD variable ordering or SDD vtree ordering. Third, when computing marginal probabilities (`MARG`), it performs limited sharing of intermediate results.

We address these limitations by extending Soufflé, a compiled Datalog synthesizer, to support the probabilistic semantics of Problog. This design enables control-flow optimizations, implements the MARG task as in Fierens et al. [1] on both BDDs and SDDs, and integrates a dynamic BDD reordering strategy to perform WMC efficiently.

The remainder of this paper is organized as follows. Section 2 introduces the background on Soufflé and the knowledge representation structures. Sections 3 and 4 describe our key contributions. Section 5 presents experimental results.

2 Background

2.1 A high-performance engine: Soufflé

Soufflé [1] is a Datalog tool that efficiently synthesizes Datalog specifications into executable C++ programs, allowing control-flow optimizations that result in a performance advantage over Problog.

To translate rules into a RAM program, Soufflé adopts the semi-naive evaluation technique. This bottom-up strategy incrementally derives new facts from existing ones. For each rule, tuples in the body relations are enumerated, and when the join conditions are satisfied, the corresponding tuples are inserted into the head relation. For recursive rules, only the newly derived tuples from the previous iteration (delta relations) are used in subsequent computations, avoiding redundant joins. The process repeats until no new facts can be derived.

The corresponding stage in ProbLog's pipeline is grounding, which transforms a first-order model into a propositional formula. Listings 1 and 2 (see `https://dtai.cs.kuleuven.be/problog/tutorial/advanced/00_inference.html`) illustrate the grounding process, which involves replacing variables with concrete constants, omitting irrelevant facts, and generating a propositional program.

Listing 1: Original ProbLog Program

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

Listing 2: Grounded ProbLog Program

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

2.2 Knowledge Representations: BDD and SDD

After grounding, Problog uses SDD (by default) to do WMC. Similar to that, we convert the resulting formula to another knowledge representation that supports efficient WMC. In this project, we focus on two well-studied representations: BDD and SDD.

2.2.1 BDD

Binary Decision Diagram (BDD) is a directed acyclic graph (DAG) that represents Boolean functions in a canonical form given a fixed variable ordering. BDDs satisfy the

mutually exclusive property, which enables efficient WMC. However, the size of a BDD can grow exponentially in the number of variables if a poor variable ordering is chosen. Therefore, finding a good variable ordering is crucial. We use the CUDD library (see `https://github.com/ivmai/cudd`) and assign weights to the edges of BDD nodes to perform WMC.

2.2.2 SDD

Sentential Decision Diagram (SDD) is another representation. Every OBDD can be represented as an SDD [2]. Therefore, SDD inherits the canonical and mutually exclusive properties of BDD. The size of an SDD depends on its vtree, which specifies how variables are partitioned.

2.3 Variable ordering for BDD

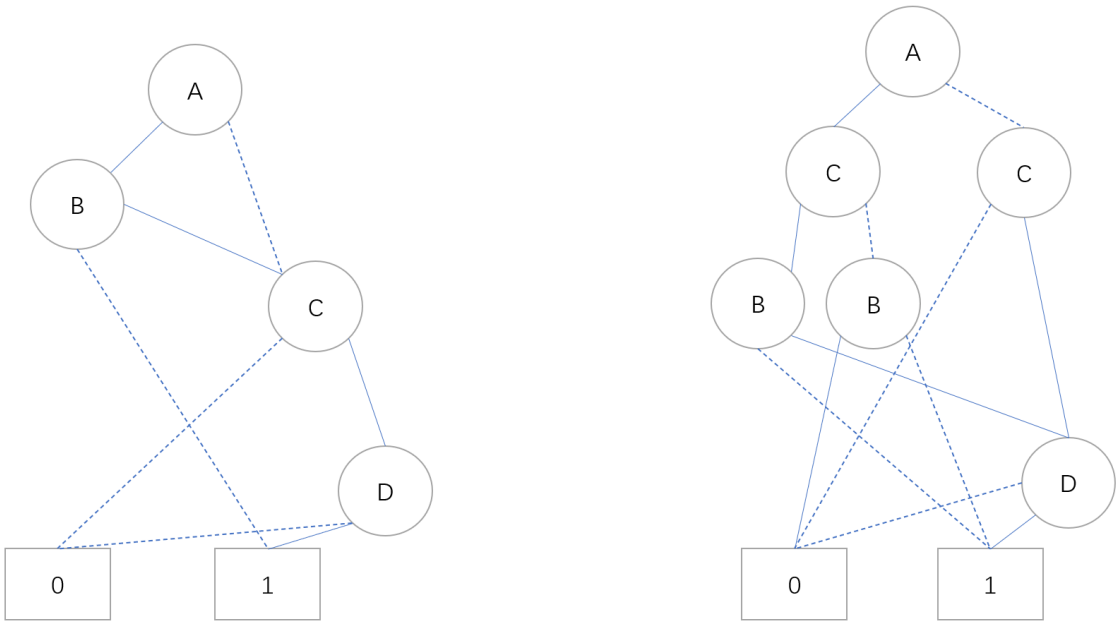

Figure a
Nodes: 4
Path: 7

Figure2.1: Instance of (A∧¬B)∨(C∧D)

Figure b
Nodes: 6
Path: 8

Figure 2.1 gives a toy example of how variable ordering affects the size of BDD. Figure a adopts the ordering of A, B, C, D while b adopts A, C, B, D, which results in different sizes. When the size becomes larger, the effect will be more significant. There are many existing works on BDD to find a good variable ordering. such as sift by Rudell [3], FORCE by Aloul [4], and their variation by Jiang [5].

The current techniques of BDD reordering in CUDD library applies include sift, window permutation [3]. By modifying the original heuristics in CUDD to an adaptive dynamic approach resulted in performance gains up to dozens of times, which indicates that ordering is crucial to the size of BDD. A good variable ordering allows us to do

WMC efficiently.

2.4 Marginal Probabilities computation: `MARG`

In `MARG`, given a set of query atoms $\mathbf{Q}$, we aim to compute the probability of each query atom conditioned on evidence, represented as $P(Q \mid \mathbf{E} = e)$ for each $Q \in \mathbf{Q}$. Naive methods using BDD or SDD form the conjunction of evidence with each query atom to compute the joint probability $P(Q \wedge \mathbf{E} = e)$. Consequently, if there are N query atoms in $\mathbf{Q}$, the WMC computation must be executed N times, once for each query atom. As a result, many intermediate results are computed redundantly.

3 Evidence Semantics and MARG Task in Soufflé

The branch on which we worked is: `github.com/Hughshine/souffle/tree/evidence-extension`, which is based on the current online branch. In this branch, we:

Extended Soufflé's probabilistic reasoning by introducing the `evidence` predicate syntax, enabling statements of the form `evidence(a(1), true)`. This required modifications to the scanner, parser, AST, RAM, derivation graph, and synthesizer.

Implemented the `MARG` computation task, allowing Soufflé's to compute marginal probabilities over query nodes when evidence nodes are provided. Important technical work included

- Extending the compiler to propagate evidence, identifying nodes as evidence when calculating the formula,
- Using BDD and SDD libraries for efficient probabilistic computation.
- Compiling into an arithmetic circuit to calculate all required probabilities in parallel, presented in Fierens et al. [6]

We are currently evaluating the boost brought by the Algorithm 34 and 35, as described in Darwiche [7]. Such algorithms allow us to evaluate $P(Q \mid \mathbf{E} = e)$ for all $Q \in \mathbf{Q}$ within two iterations rather than instantiating the indicator variables for each query node Q separately.

4 SddManager

To reduce the burden of manual memory management and prevent common memory errors such as leaks, we introduced `SddManager`, which adopts the reference counting strategy of `CuddManager.h` developed by Xuyang Li. This approach ensures that SDD nodes are automatically retained and released as needed. In developing this component, we examined the source code of the SDD library, paying particular attention to

its `minimize` function, as described by Choi [8]. This function can be viewed as a naive generalization of the sifting reordering strategy in BDD, potentially searching among up to twelve possible states of a vtree fragment to identify the state that produces the most compact SDD.

5 Experiments and Conclusion

Problem	**Problog Status / Time (s)**	**Soufflé Status / Time (s)**
P1	success / 68.6981	success / 0.270
P3	success / 72.7188	success / 0.315
P4	success / 0.8115	success / 0.030
P5	success / 1.3897	success / 0.033
P6	success / 10.7144	success / 0.045
P7	timeout / 1800	success / 114.832
P8	timeout / 1800	timeout / 1800
P9	timeout / 1800	success / 0.535

Table 1: Problog vs Soufflé benchmark results for P1-P9

We run Problog and Soufflé on the same examples (see `github.com/Hughshine/problog-benchmark`). As expected from our experiments, Table 1 shows that Soufflé significantly outperforms Problog(both on BDD) in terms of execution time in most benchmark problems. Although Problog shows comparative performance for small problems, it experiences timeouts on larger instances (P7, P9), whereas Soufflé completes these successfully and much faster. This highlights the efficiency advantage of Soufflé over Problog.

6 Future Work

In future work, we plan to continue exploring static BDD reordering strategies and SDD vtree optimization. Besides, we may utilize the potential of derivation graph in Problog to find a good ordering. Additionally, we aim to study on compiling BDD into arithmetic circuits to compute marginal probabilities more efficiently.

Acknowledgements

We thank Xuyang Li for his detailed guidance and Dr. Jingbo Wang for the given opportunity.

References

[1] Herbert Jordan, Bernhard Scholz, and Pavle Subotić. Soufflé: On synthesis of program analyzers. In *Computer Aided Verification (CAV 2016)*, volume 9780 of *Lecture Notes in Computer Science*, pages 422–430. Springer, 2016.

[2] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 819–826. AAAI Press, 2011.

[3] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '93, pages 42–47, Santa Clara, California, USA, 1993. IEEE Computer Society Press.

[4] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. Force: a fast and easy-to-implement variable-ordering heuristic. In *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, GLSVLSI '03, pages 116–119, Washington, D. C., USA, 2003. Association for Computing Machinery.

[5] Chuan Jiang, Junaid Babar, Gianfranco Ciardo, Andrew S. Miner, and Benjamin Smith. Variable reordering in binary decision diagrams. In *26th International Workshop on Logic & Synthesis*, 2017.

[6] D. Fierens, G. Van den Broeck, J. Renkens, and et al. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

[7] Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[8] Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13)*, pages 187–194, 2013.