JUICE: A Julia Package for Logic and Probabilistic Circuits

Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, Guy Van den Broeck
Computer Science Department
University of California, Los Angeles
{mhdang, pashak, yliang, aver, guyvdb}@cs.ucla.edu

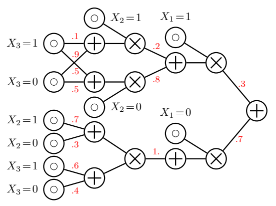
Abstract

JUICE is an open-source Julia package providing tools for logic and probabilistic reasoning and learning based on logic circuits (LCs) and probabilistic circuits (PCs). It provides a range of efficient algorithms for probabilistic inference queries, such as computing marginal probabilities (MAR), as well as many more advanced queries. Certain structural circuit properties are needed to achieve this tractability, which JUICE helps validate. Additionally, it supports several parameter and structure learning algorithms proposed in the recent literature. By leveraging parallelism (on both CPU and GPU), JUICE provides a fast implementation of circuit-based algorithms, which makes it suitable for tackling large-scale datasets and models.

Introduction

Decision making in the real world requires the ability to compactly represent and easily learn complex models, such as probability distributions and logical knowledge bases. It also requires the ability to perform logic and probabilistic reasoning about complex models. Moreover, in many sensitive domains, these computations need to be carried out exactly and efficiently. Classical models like Bayesian and Markov networks, and recent neural density estimators like variational autoencoders (Kingma and Welling 2013) and normalizing flows (Papamakarios et al. 2019) can capture complex distributions, but at the cost of being highly intractable. They only offer approximate answers to inference queries.

In contrast, probabilistic circuits (PCs) (Choi, Vergari, and Van den Broeck 2020) are expressive, deep, yet tractable probabilistic models that permit exact inference for many types of queries, in time linear in the size of their representation. They are increasingly used in downstream applications that require efficient probabilistic inference: algorithmic fairness (Choi, Dang, and Van den Broeck 2020), missing data (Khosravi et al. 2019, 2020; Correia, Peharz, and de Campos 2020), graphical model inference (Shih and Ermon 2020), probabilistic programming (Holtzen, Van den Broeck, and Millstein 2020; Skryagin et al. 2020), activity recognition (Galindez Olascoaga et al. 2019), vision (Stelzner, Peharz, and Kersting 2019), and explainability (Nourani et al. 2020; Wang, Khosravi, and Van den Broeck 2020).



```
X1,X2,X3=literals(ProbCircuit,3)
pc=.3*(X1[1]*
      (.2*(X2[1]
           *(.1X3[1]+.9X3[2]))
      +.8*(X2[2]
           *(.5X3[1]+.5X3[2]))))+
   .7*(X1[2]*
       +((.7X2[1]+.3X2[2])*
         (.6X3[1]+.4X3[2])))
```

Figure 1: A smooth deterministic structured-decomposable PC representing $\mathrm{p}(X_1, X_2, X_3)$ and JUICE code to create it.

PCs are the probabilistic counterpart to tractable logic circuits (LCs) (Darwiche and Marquis 2002). They are compact representations of Boolean functions that are successfully employed in many knowledge representation and reasoning scenarios, often as a target for compilation (Darwiche and Marquis 2002; Van den Broeck and Suciu 2017).

We introduce JUICE,[1] a Julia package that offers researchers and practitioners efficient routines to construct, compile, learn and reason with LCs and PCs. It provides a wide range of functionality, while being easy to extend and customize. Next, we give brief theoretical background, and discuss the high-level design and implementation of JUICE.

Background on Tractable Circuits

Representation Circuits, including *LCs* and *PCs*, are computational graphs where nodes are computation units and edges define an order of execution. A LC $\mathcal{G}$ is a directed acyclic graph encoding a logical formula where each inner node represents either *disjunction* or *conjunction*, and each *input* node is a logical literal, e.g., $X = 1$. Examples of LCs are d-DNNF circuits, binary, and sentential decision diagrams (Darwiche and Marquis 2002; Darwiche 2011).

A PC $(\mathcal{G}, \boldsymbol{\theta})$ over random variables (RVs) $\mathbf{X}$ represents a joint probability distribution $\mathrm{p}(\mathbf{X})$ with parameters $\boldsymbol{\theta}$. PCs are a family of tractable probabilistic models that includes arithmetic circuits (Darwiche 2002), sum-product networks (Poon and Domingos 2011), and cutset networks (Rahman, Kothalkar, and Gogate 2014). A PC structure $\mathcal{G}$ is similar to a LC structure, except that conjunction nodes are written as *products*, disjunction nodes are written as *sums*,

[1]JUICE source code is available at https://github.com/Juice-jl

Models	Logic Circuits	Probabilistic Circuits	Pairs of Circuits
Algorithms	forward & backward traversal smooth, condition, split, merge, clone (weighted) model counting compilation, SAT	EVI, MAR, CON, MPE (conditional) sampling MLE/ EM parameter learning hill climbing structure learning	multiply KL-divergence expectations moments

Table 1: The list of major functionalities that JUICE supports. Many routines benefit from SIMD/GPU parallelization.

and input nodes n are associated with some tractable distribution f_n – for example, an indicator function for discrete variables, or a Gaussian for continuous variables. Parameters $\boldsymbol{\theta}$ are associated with each sum node and input distribution.

Concretely, let $\mathsf{in}(n)$ be the set of inputs of an inner node n. Then $\mathrm{p}(\mathbf{X})$ is defined as follows:

$$\mathrm{p}_n(\mathbf{x}) = \begin{cases} f_n(\mathbf{x}) & \text{if } n \text{ is an input node} \\ \prod_{c \in \mathsf{in}(n)} \mathrm{p}_c(\mathbf{x}) & \text{if } n \text{ is a product node} \\ \sum_{c \in \mathsf{in}(n)} \theta_{n,c}\, \mathrm{p}_c(\mathbf{x}) & \text{if } n \text{ is a sum node} \end{cases}$$

Intuitively, a product node n defines a factorized distribution, and a sum node n defines a mixture model parameterized by weights $\theta_{n,c}$. See Figure 1 for an example of a PC.

Inference Given certain structural properties, PCs support tractable probabilistic inference. JUICE makes it easy to test for these properties and construct circuits that have them.

A circuit is said to be *smooth* if for every sum node all of its inputs depend on the same set of RVs; it is *decomposable* if for every product node its inputs depend on disjoint sets of RVs (Darwiche and Marquis 2002). Given a smooth and decomposable PC, computing marginal probabilities (MAR) becomes tractable for any partial evidence. The computation reduces a customized feed forward evaluations of the circuit. This also implies tractable computation of conditional probabilities (CON), which are ratios of marginals.

A circuit is deterministic if for every complete input assignment $\mathbf{x}$, at most one input of every sum node has a non-zero output. Decomposability and determinism enables tractable exact inference of most-probable explanations (MPE) (Choi and Darwiche 2017), also called MAP inference.

Another useful property is *structured decomposability*; it means that product nodes with the same scope decompose their variables in the same way (Kisa et al. 2014). Forms of structured decomposability guarantee tractable inference for advanced queries and manipulations: multiplying PCs (Shen, Choi, and Darwiche 2016), or computing expectations (Choi, Van den Broeck, and Darwiche 2015; Khosravi et al. 2019) and KL divergences (Liang and Van den Broeck 2017).

An Overview of the JUICE Package

Table 1 summarizes the main compilation, reasoning and learning functionality implemented in the JUICE package.

Learning JUICE supports highly efficient parameter learning given complete or incomplete data (Kisa et al. 2014; Peharz et al. 2016). For example, the following code snippet learns the parameters of a PC on GPU and evaluates the likelihoods given training examples. We report runtimes for the training set PLANTS (Lowd and Davis 2010) with 17,412 examples and a PC with around 150 thousand nodes.

```
estimate_parameters(pc, data) # MLE; 88 ms
EVI(pc, data) # Full Evidence; 83 ms
```

JUICE also implements several structure learners that adopt hill-climbing strategies (Liang, Bekker, and Van den Broeck 2017; Dang, Vergari, and Van den Broeck 2020).

Design We model circuits as linked node structures. Inference routines iterate over the circuit forward or backward, passing results from node to node. Arbitrary inference algorithms can be implemented by providing different lambda functions, corresponding to different computations, to a general-purpose, optimized circuit traversal and propagation infrastructure. When modifying circuits, we leverage Julia's automatic garbage collection, and the fact that nodes only link to their inputs, not the consumers of their output.

Parallel computing on CPU and GPU A linked node representation is an intuitive data structure for circuits. However, it has the drawback that it makes computations sparse, making it harder to leverage parallelism to speed up computation. To optimize performance during inference and learning, we translate the circuit's DAG into a layered computational graph, starting with the input layer, and each layer only depending on the previous layers. Since the computations on the nodes in the same layer can be cached in one large vector, we can simultaneously parallelize our computation over the nodes in the layer on the one hand, and training examples or inference task data on the other hand. Additionally, we leverage Julia's multiple dispatch to provide customized kernels to accelerate computation on both CPUs and GPUs (using SIMD and CUDA kernels respectively). Experiments show that CPU parallelism gives significant speed-ups, which even become an order of magnitude faster with GPU parallelism, all using the same underlying data structures.

Other circuit types Finally, because JUICE separates structures (LCs and PCs) and algorithms (e.g., inference and learning), making custom circuit models with specific semantics is relatively easy. For example, JUICE also provides an implementation of *logistic circuits* (Liang and Van den Broeck 2019) which is a type of discriminative PC for classification.

Acknowledgments

This work is partially supported by NSF (IIS-1943641, IIS-1633857, CCF-1837129), DARPA (N66001-17-2-4032), a Sloan Fellowship, Intel & Facebook.

References

Choi, A.; and Darwiche, A. 2017. On Relaxing Determinism in Arithmetic Circuits. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML)*.

Choi, A.; Van den Broeck, G.; and Darwiche, A. 2015. Tractable Learning for Structured Probability Spaces: A Case Study in Learning Preference Distributions. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*.

Choi, Y.; Dang, M.; and Van den Broeck, G. 2020. Group Fairness by Probabilistic Modeling with Latent Fair Decisions. In *Algorithmic Fairness through the Lens of Causality and Interpretability Workshop at NeurIPS (AFCI)*.

Choi, Y.; Vergari, A.; and Van den Broeck, G. 2020. Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models URL http://starai.cs.ucla.edu/papers/ProbCirc20.pdf.

Correia, A.; Peharz, R.; and de Campos, C. P. 2020. Joints in Random Forests. *Advances in Neural Information Processing Systems 33 (NeurIPS)* .

Dang, M.; Vergari, A.; and Van den Broeck, G. 2020. Strudel: Learning Structured-Decomposable Probabilistic Circuits. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models (PGM)*.

Darwiche, A. 2002. A Logical Approach to Factoring Belief Networks. In *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR)*, 409–420.

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

Darwiche, A.; and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17(1): 229–264.

Galindez Olascoaga, L. I.; Meert, W.; Shah, N.; Verhelst, M.; and Van den Broeck, G. 2019. Towards Hardware-Aware Tractable Learning of Probabilistic Models. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*.

Holtzen, S.; Van den Broeck, G.; and Millstein, T. 2020. Scaling Exact Inference for Discrete Probabilistic Programs. *Proc. ACM Program. Lang. (OOPSLA)* doi:https://doi.org/10.1145/342820.

Khosravi, P.; Choi, Y.; Liang, Y.; Vergari, A.; and Van den Broeck, G. 2019. On Tractable Computation of Expected Predictions. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*.

Khosravi, P.; Vergari, A.; Choi, Y.; Liang, Y.; and Van den Broeck, G. 2020. Handling Missing Data in Decision Trees: A Probabilistic Approach. In *The Art of Learning with Missing Values Workshop at ICML (Artemiss)*.

Kingma, D. P.; and Welling, M. 2013. Auto-Encoding Variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014. Probabilistic sentential decision diagrams. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Liang, Y.; Bekker, J.; and Van den Broeck, G. 2017. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*.

Liang, Y.; and Van den Broeck, G. 2017. Towards Compact Interpretable Models: Shrinking of Learned Probabilistic Sentential Decision Diagrams. In *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*.

Liang, Y.; and Van den Broeck, G. 2019. Learning Logistic Circuits. In *Proceedings of the 33rd Conference on Artificial Intelligence (AAAI)*.

Lowd, D.; and Davis, J. 2010. Learning Markov Network Structure with Decision Trees. In *Proceedings of the 10th IEEE International Conference on Data Mining*, 334–343. IEEE Computer Society Press.

Nourani, M.; Roy, C.; Rahman, T.; Ragan, E. D.; Ruozzi, N.; and Gogate, V. 2020. Don't Explain without Verifying Veracity: An Evaluation of Explainable AI with Video Activity Recognition.

Papamakarios, G.; Nalisnick, E.; Rezende, D. J.; Mohamed, S.; and Lakshminarayanan, B. 2019. Normalizing Flows for Probabilistic Modeling and Inference. *arXiv preprint 1912.02762* .

Peharz, R.; Gens, R.; Pernkopf, F.; and Domingos, P. 2016. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence* .

Poon, H.; and Domingos, P. 2011. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 689–690. IEEE.

Rahman, T.; Kothalkar, P.; and Gogate, V. 2014. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees. In *Joint European conference on machine learning and knowledge discovery in databases*, 630–645. Springer.

Shen, Y.; Choi, A.; and Darwiche, A. 2016. Tractable Operations for Arithmetic Circuits of Probabilistic Models. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 3936–3944.

Shih, A.; and Ermon, S. 2020. Probabilistic Circuits for Variational Inference in Discrete Graphical Models. *Advances in Neural Information Processing Systems 33 (NeurIPS)* .

Skryagin, A.; Stelzner, K.; Molina, A.; Ventola, F.; Yu, Z.; and Kersting, K. 2020. Sum-Product Logic: Integrating Probabilistic Circuits into DeepProbLog. In *Working Notes of* *the ICML 2020 Workshop on Bridge Between Perception and Reasoning: Graph Neural Networks and Beyond*.

Stelzner, K.; Peharz, R.; and Kersting, K. 2019. Faster Attend-Infer-Repeat with Tractable Probabilistic Models. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR.

Van den Broeck, G.; and Suciu, D. 2017. *Query Processing on Probabilistic Data: A Survey*. Foundations and Trends in Databases. Now Publishers. doi:10.1561/1900000052.

Wang, E.; Khosravi, P.; and Van den Broeck, G. 2020. Towards Probabilistic Sufficient Explanations. In *Extending Explainable AI Beyond Deep Models and Classifiers Workshop at ICML (XXAI)*.