Scalable Learning of Probabilistic Circuits

Renato Lui Geh[1]
Advisor: Denis Deratani Mauá[1]

[1]Institute of Mathematics and Statistics – University of São Paulo
São Paulo – SP – Brazil

renatolg@ime.usp.br

Abstract. *Probabilistic circuits (PCs) are a family of tractable probabilistic models capable of answering a wide range of queries exactly and in polytime. While inference is usually straightforward, learning PCs that both obey the needed restrictions for inference tractability and exploit their expressive power has proven to be a challenge. This dissertation aims to propose fast and scalable structure learning algorithms for PCs from two different standpoints: from a logical point of view, we efficiently construct a PC that takes certain knowledge in the form of logical constraints and scalably translate them into a probabilistic circuit; from the viewpoint of data guided structure search, we propose hierarchically building PCs from random hyperplanes. We empirically show that either approach is competitive against state-of-the-art methods of the same class in both performance and scalability.*

1. Introduction

In our dissertation, we explore and develop algorithms for learning Probabilistic Circuits (PCs), a family of deep models distinctly specified as inductive compositions of distributions through graphical formalisms. PCs are highly expressive and efficient, and as long as certain graphical properties are ensured to hold, provide a wide range of tractable inference tasks to query from, making them powerful statistical toolboxes. We approach the problem of learning PCs from two distinct lenses: one from a logic point of view, and the other from a purely data-centric perspective. In a neuro-symbolic fashion, we exploit known domain knowledge and learn PCs both from a propositional logic formula and available data. We show that not only does this combination of learning from both certain and uncertain knowledge outperform purely data-driven approaches when under a low-data regime, but also provides a much more scalable approach compared to existing algorithms for the same class of PCs. A preliminary version of our method was published at the 8th Symposium on Knowledge Discovery, Mining and Learning [Geh et al. 2020], with a later full version appearing at the 37th Conference on Uncertainty in Artificial Intelligence [Geh and Mauá 2021], the premier conference on the fields of Probabilistic Machine Learning and Reasoning under Uncertainty.

In a completely distinct approach, we look at the problem of learning PCs solely from data. We revisit random projections, a known method for dimensionality reduction, and develop a method to efficiently construct a PC by means of hierarchically stacking random projections. We empirically show that we are able to achieve state-of-the-art performance at the fraction of the (learning) time of competitors. A preliminary version of this contribution was published at the 4th Tractable Probabilistic Modeling Workshop

[Geh and Mauá 2021], the main gathering spot for specialists in the area of probabilistic modeling in Machine Learning and Artificial Intelligence. Furthermore, the dissertation was awarded 1st place at the 13th National Contest of PhD and MSc Theses on Artificial and Computational Intelligence (CTDIAC) in 2022, a biannual event recognizing the best MSc and PhD theses in Brazil on the subject of Artificial and Computational Intelligence.

Finally, as a minor contribution, we provide in the dissertation a systematic review of the most popular learning algorithms for probabilistic circuits. We write down a superficial complexity analysis — a subject that is often absent in machine learning literature — of each of the algorithms, and detail both the assumptions made and the structural guarantees required and provided by each method. Due to page limit constraints, we omit this last contribution from this report.

We structure the text as follows. We start with a brief informal introduction to probabilistic circuits in Section 2. We then follow with our first major contribution, giving a brief description of our proposed learning algorithm and showing empirical results in Section 3. Our second major contribution comes in Section 4; we show the idea behind our method and give empirical support to both its performance and scalability. Lastly, we conclude this report in Section 5, making a brief summary of our contributions.

2. Probabilistic circuits: a birds-eye view

In a nutshell, Probabilistic Circuits (PCs) are nothing more than computational graphs whose nodes represent a probability distribution made out of the (recursive) composition of all of its children. We adopt some of the usual nomenclature from Graph Theory; we shall only work with DAGs and say that, for some graph $\mathcal{G} = (\mathbf{N}, \mathbf{E})$, when two nodes $\mathsf{N}_1, \mathsf{N}_2 \in \mathbf{N}$ are connected by an edge coming from N_1 to N_2, then N_1 is the *parent* of N_2; and N_2 the *child* of N_1.

Definition 1 (Probabilistic circuit) *A* probabilistic circuit *$\mathcal{C}$ is a rooted connected DAG whose nodes describe non-negative functions: a* sum node *represents a weighted summation over its children, a* product node *multiplies all of its children, and* input nodes*, i.e. nodes with no outgoing edges, are defined as univariate probability density functions.*

Inference tasks amount to feeding values to input nodes and then evaluating the computational graph by the recursive application of each computational unit. Figure 1b shows a simple probabilistic circuit composed out of a product node ⊗, its children sum nodes ⊕ and ⊕, and four input nodes ∿, ∿, ∿ and ∿, each representing a distinct Gaussian distribution. Edges coming out from sum nodes are weighted and sum to one, effectively defining a mixture over the children's distribution. Figure 1a shows two Gaussian mixtures corresponding to nodes ⊕ and ⊕, while Figure 1c shows the computational flow from evaluating the probability of $x = 2$ and $y = 4$ as described by the circuit.

There is a natural relationship between *probabilistic* circuits and *logic* circuits. Note that sums and products act equivalently to disjunctions and conjunctions in propositional logic under the Boolean semiring. In fact, a probabilistic circuit whose input nodes are propositional literals has its support defined by its underlying logic circuit. Such PCs are often referred to as Probabilistic Sentential Decision Diagrams (PSDDs [Kisa et al. 2014]) as long as some other conditions are also met. Figure 2 shows a PSDD whose support is defined by a logic circuit representation of the formula

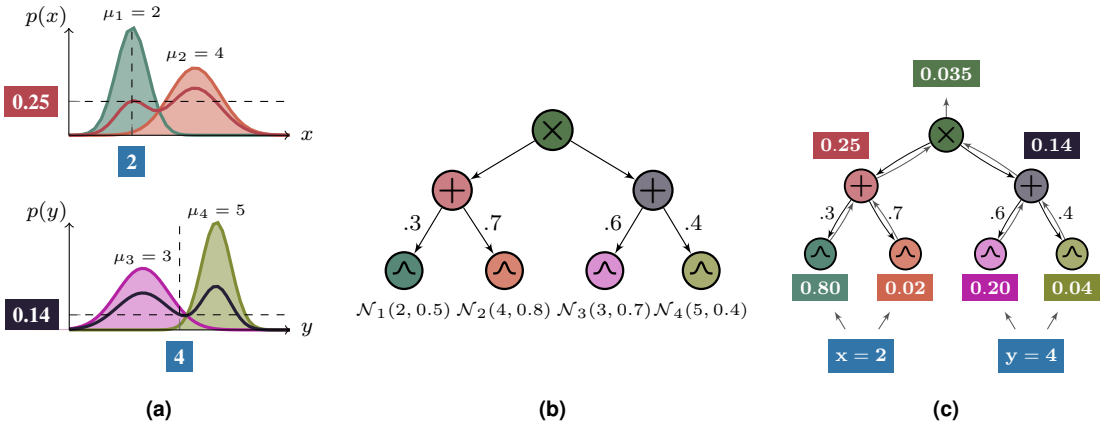

Figure 1. A product node, its computational flow, and node values for query $p(x = 2, y = 4) = 0.035$ with each of its two children as mixture models over distinct variables X and Y.

$\phi(A, B, C) = (A \vee \neg B) \wedge (\neg B \vee C)$. We write ⊕ as ∩, and ⊗ as ∪ when we mean that the support of a PC comes from a logic formula represented by its underlying logic circuit; syntactically, the two representations are identical.

An attractive feature of PCs that sets them apart from other probabilistic models is the range of queries that are possibly extracted from a circuit. Computing queries like the probability of partial assignments (i.e., marginals), conditional probabilities, *maximum a posteriori* probabilities, or even more advanced tasks such as probabilities of logical events and information theoretic queries like entropies and divergences, are all extractable in polytime (on the size of the circuit) when certain (sufficient) conditions on their structure, viz. their computational graph, are met. For the sake of simplicity and to respect the page limit, we avoid defining these conditions or even providing a more formal definition of PCs or PSDDs. Chapter 2 of the dissertation[1] is dedicated to such formalities, with an extensive discussion on the sufficiency of conditions, the connection of popular probabilistic and logic models with PCs, and the different reasoning capabilities of circuits.

Learning expressive probabilistic circuits from data under the structural constraints imposed by a combination of several conditions for tractability has proven to be quite a difficult task. This problem becomes even more challenging when domain knowledge is present, as the support of the PC must also be consistent with prior knowledge. In the next section, we explore a technique for learning PSDDs from both certain (in the form of a propositional logic formula) and uncertain knowledge (in the form of data).

3. Efficiently learning from both data and prior knowledge

Although there have been many advances in the literature on building PSDDs, the large majority of these work by either focusing purely on logical approaches for compiling CNF or DNFs into the smallest possible (i.e., canonical) circuit representation [Oztok and Darwiche 2015, Choi and Darwiche 2013], or on learning circuits for domain specific tasks [Choi et al. 2015, Shen et al. 2017]. Before our contribution, there were only two existing domain agnostic algorithms for learning the structure of PSDDs from

[1] https://www.teses.usp.br/teses/disponiveis/45/45134/tde-23052022-122922/en.php

A	B	C	$\phi(\mathbf{x})$	$p(\mathbf{x})$
0	0	0	1	0.140
1	0	0	1	0.024
0	1	0	0	0.000
1	1	0	0	0.000
0	0	1	1	0.560
1	0	1	1	0.096
0	1	1	0	0.000
1	1	1	1	0.180

$$\phi(A, B, C) = (A \vee \neg B) \wedge (\neg B \vee C)$$

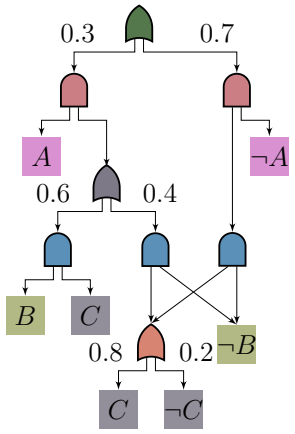

Figure 2. A probabilistic circuit encoding a logic formula as its support by means of its underlying logic circuit.

data: LEARNPSDD [Liang et al. 2017] and STRUDEL [Dang et al. 2020], both of which work by growing an initial circuit into a more expressive PC, preserving existing structural constraints and support. This, however, means that they delegate the translation of logic formula to logic circuit to a separate, independent algorithm, e.g. compiling from a CNF. The main drawback of this approach comes from the fact that variables that do not play a role in the logic formula (but may play a probabilistic role in the distribution) will be completely discarded from the logic translation procedure, meaning that the resulting initial circuit will contain a trivial representation (e.g., fully factorized) of these variables completely separate from the logical portion of the circuit.

In our published article at UAI 2021 [Geh and Mauá 2021], whose content is described in detail in Chapter 4 of the dissertation, we proposed the first PSDD learning algorithm that learns a PC both from a logic formula and from data.

3.1. Sampling and averaging circuits

As previously (implicitly) stated, our objective is to build a logic circuit $\mathcal{L}$ from a propositional logic formula ϕ and derive a probabilistic circuit $\mathcal{C}$ from $\mathcal{L}$ by adding weights to edges coming from sum nodes. We choose to do so by a top-down recursive decomposition of ϕ, where at each step we further decompose sub-formulae until either a $\top$, $\bot$ or literal is found. A naïve solution would be to construct a disjunction (i.e., sum) node followed by conjunctions (i.e., products) of all possible worlds, recursively applying the same method for each children. Unfortunately, this is clearly exponential on the number of variables. Instead, we resort to a relaxed version where we guarantee a polynomially sized circuit is constructed. We do so by bounding the number of product nodes to a constant c and randomly sampling c conjunctions of literals such that they are mutually exclusive and exhaustive. To make sure the structural conditions set by the definition of PSDDs are preserved, we must then perform a relaxation step on the logical formula, which possibly attributes nonzero probability values to impossible worlds. This relaxation does not inhibit the expressivity of the model, and due to the random nature of our method, can be mitigated by a combination of ensembling coupled with sampling guidance, as discussed in Section 4.4.5 of the dissertation. The details on this procedure are formally described in Chapter 4 of the dissertation. Figure 3 illustrates the sampling pro-

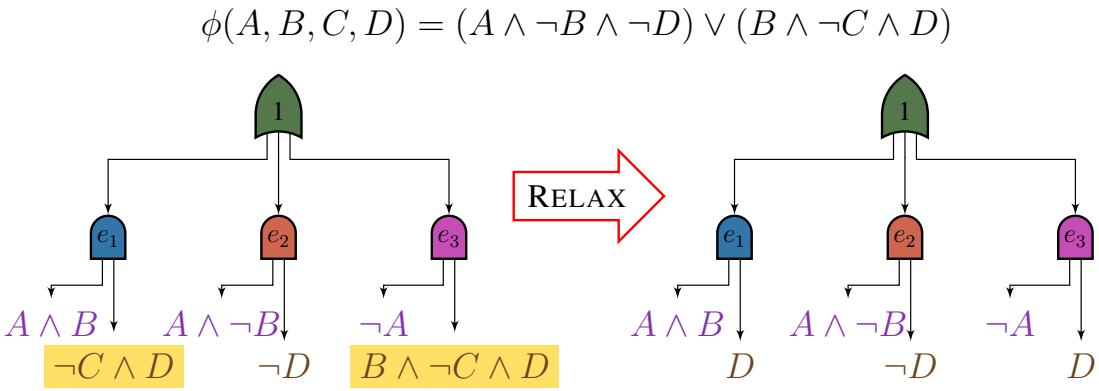

Figure 3. The recursive process of sampling conjunctions (left) followed by the relaxation (right) of the subformulae in yellow.

cess followed by the relaxation of the circuit needed for ensuring the structural constraints of PSDDs. We name the entire algorithmic procedure SAMPLEPSDD.

Since the process of building the circuit is done at random, we may sample a fixed number, say k, of PCs this way and aggregate them into an ensemble. This can be done immediately by joining every sampled circuit with a sum node; each weighted edge defines the weight given for each component of the ensemble. We experiment with the following 5 strategies for ensemble learning: (1) we set all weights to be uniform, (2) we learn weights by setting (and normalizing) them to their training set likelihood (LLW), (3) learn by expectation-maximization (EM) over only component weights [Dempster et al. 1977], (4) apply stacking for density estimation [Smyth and Wolpert 1998], and lastly (5) compose a Bayesian model combination (BMC) [Monteith et al. 2011] of PSDDs. Details on these ensemble strategies can be found in Section 4.3 of the dissertation.

3.2. Experiments

To understand how our method behaves in practice and how each ensemble strategy affects performance, we evaluated our method in 5 distinct domains. Due to the lengthy discussion it would take to describe each experimental setting, we direct the reader to Sections 4.4.1–3 of the dissertation. For this report, it suffices to say that each domain contains the data and a propositional logic formula describing its domain knowledge. We compare the performance of SAMPLEPSDD against STRUDEL [Dang et al. 2020], LEARNPSDD [Liang et al. 2017], and LEARNSPN [Gens and Domingos 2013]. It is worth noting that LEARNSPN is a popular purely data-driven method for learning a more flexible class of PCs (as opposed to the stronger structural conditions we impose for PSDDs), and is here used as a baseline for measuring performance. In the case of STRUDEL, we use as initial circuit the purely data-driven method proposed by [Dang et al. 2020], meaning that neither STRUDEL nor LEARNSPN are domain knowledge aware. We experiment with two versions of LEARNPSDD: the first one follows the method described by [Liang et al. 2017], where an initial circuit is compiled from a CNF containing the logic formula, which is then grown by LEARNPSDD; the second alternative version, for when CNF representations are intractable, replaces the CNF compilation with a BDD, a process trivially done since BDDs are a strict subset of (P)SDDs [Bova 2016].

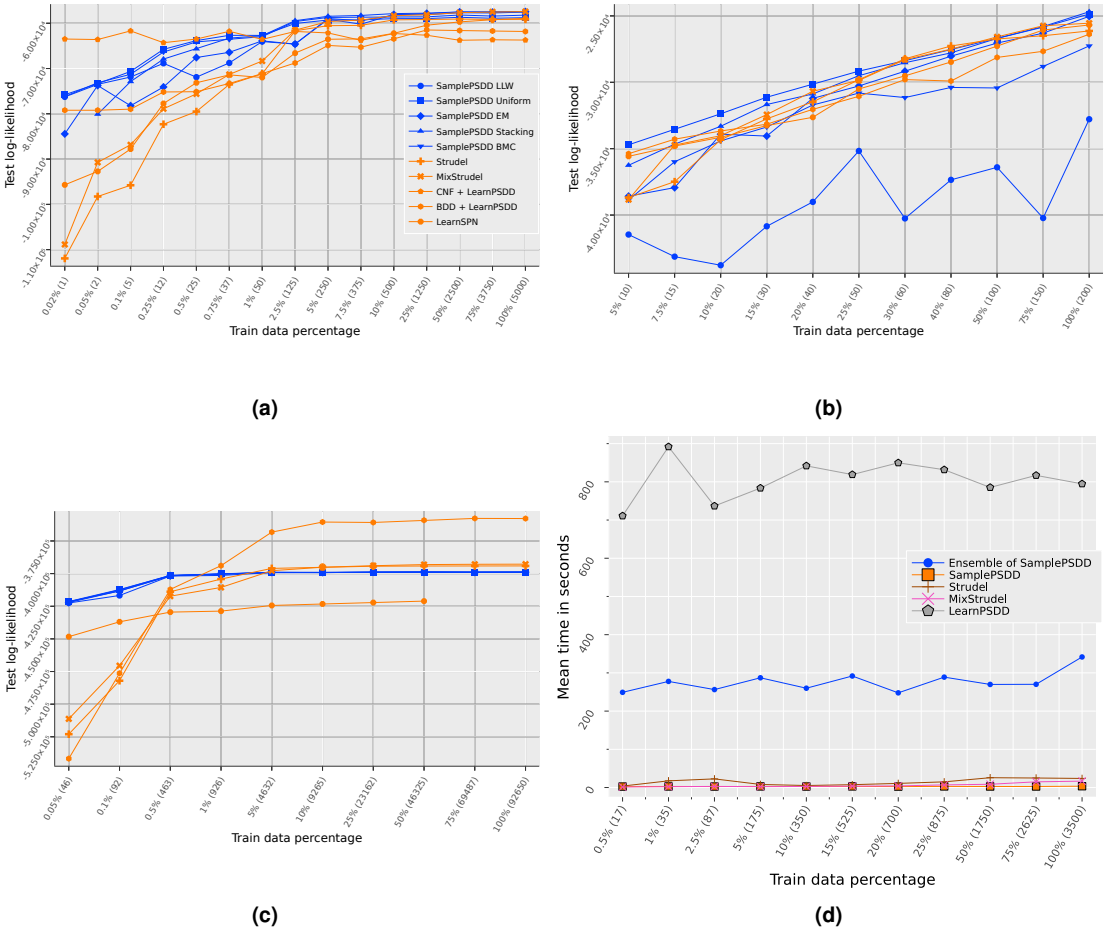

Figure 4. Likelihood performance of our methods against state-of-the-art competitors on three of the five domains evaluated (a-c). Mean average learning time performance (d).

Experiments were run on an Intel i7-8700K 3.70 GHz machine with 12 cores and 64GB RAM. Figure 4 compares the results of our methods against competitors in three of the 5 datasets (see Section 4.4 of the dissertation), where the abscissa quantifies the percentage of available data for training, and the ordinate shows the test log-likelihood (higher is better). Notably, our methods achieve competitive performance especially under low-data regimes yet still remain competitive when more data is available. Figure 4 shows how likelihood weighting had the worst performance, as ensembles trained this way often degenerate to a single component. Stacking, on the other hand, proved highly competitive. Of note is the interruption of LEARNSPN in Figure 4c due to the highly intensive memory cost of the learning algorithm; when faced with more than 50% of the dataset, the algorithm ran out of memory. Figure 4d shows the learning time of each algorithm; each curve shows the average time in seconds to generate a single circuit, except for Ensemble of SAMPLEPSDD's, which shows the time to sample 100 circuits and then learn component weights via LLW. On average, STRUDEL took 15 seconds, LEARNPSDD 13 minutes, and SAMPLEPSDD 3 seconds to learn a single PSDD.

Section 4.4.5 of the dissertation further comments on the performance of SAMPLEPSDD as a logical model, empirically showing that by modifying some of the pa-

rameters, we are able to achieve better approximations of the original formula at a cost to learning time.

4. Learning circuits with random projections

Random Projections (RPs) are widely known in the Density Estimation Tree (DET) literature for producing seemingly better data partitionings compared to the usual axis-aligned splits commonly done in k-d trees [Freund et al. 2008]. Intuitively, RPs are nothing more than randomly generated oblique hyperplanes hierarchically stacked on top of each other in order to partition data into cells. Interestingly, [Dasgupta and Freund 2008] provably showed that partitioning data into polytopes this way should allow DETs to successfully learn low dimension manifolds with reasonably high probability. Unfortunately, despite this attractive theoretical guarantee, DETs lack the expressivity, accuracy and flexibility of more modern probabilistic models.

It has recently been shown that probabilistic circuits are more closely related to DETs than previously thought [Correia et al. 2020], and are, as a matter of fact, a subset of PCs. Although an interesting subject, for the sake of brevity we reserve this discussion on the relationship of DETs and PCs to Example 2.3 and Section 5.1 of the dissertation, and instead focus on the more vital question: if DETs are PCs, can we take advantage of known learning procedures in DETs and transplant them to more general circuits? More importantly, are the results from RPs also true in the case of PCs? We seek to understand the answer to this last question from a more practical point of view, producing a learning algorithm for PCs inspired by known random projection learning algorithms for DETs.

4.1. LearnRP

An RP hyperplane of dimension $d = |\mathbf{X}|$ discriminates an assignment $\mathbf{x}$ to one of its side or the other via a Boolean mapping $f : \mathbf{x} \mapsto \sum_{i=1}^{d} x_i \cdot a_i > \theta$, where $\mathbf{a} = \{a_i\}_{i=1}^{d}$ describes a random unit vector and θ is a projection threshold found in such a way that the two splits are more or less equally sized. Let $\mathbf{D}$ be the dataset to be learned in matrix form, with rows as observations and columns listing variables. We propose a learning algorithm, which we call LearnRP, that samples a hyperplane f, partitions the dataset $\mathbf{D}$ into two splits $\mathbf{S}_1 = \{\mathbf{x} | \forall \mathbf{x} \in \mathbf{D} \wedge f(\mathbf{x})\}$ and $\mathbf{S}_2 = \{\mathbf{x} | \forall \mathbf{x} \in \mathbf{D} \wedge \neg f(\mathbf{x})\}$, and constructs a sum node ⊕ with two product nodes ⊗ and ⊗, initializing weights to the proportion of each partition $|\mathbf{S}_1|/|\mathbf{D}|$ and $|\mathbf{S}_2|/|\mathbf{D}|$. Following a given binary tree, each product node then decomposes the sub-data column-wise in agreement with the current variables. The procedure then recursively repeats for each of the products' children. Figure 5 shows a call to LearnRP, with hyperplanes matching colors with nodes; at each recursive call, data points are projected onto the space of variables until only a single variable remains, in which case a univariate distribution (e.g., a Gaussian in Figure 5) is learned. Once a PC is constructed from this procedure, we fine-tune weights by first learning weights via 100 iterations of mini-batch Expectation-Maximization (EM), followed by 30 iterations of full EM. Proper formalization of both RPs and LearnRP is contained in Chapter 5 of the dissertation.

4.2. Experiments

We compare the performance of LearnRP on the 20 well-known binary datasets for density estimation [Lowd and Davis 2010, Van Haaren and Davis 2012] against reported

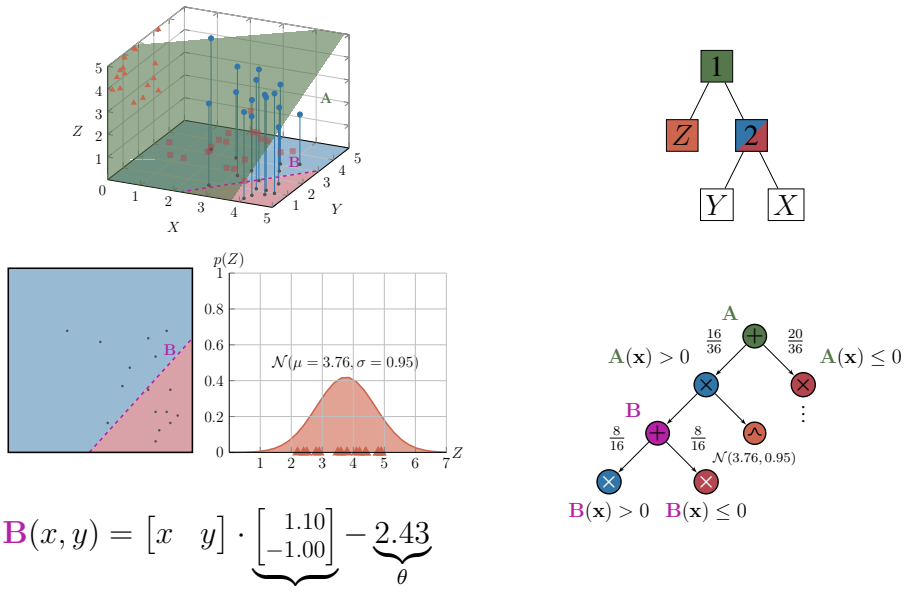

Figure 5. A call to LEARNRP where hyperplane B is sampled, partitioning the data (top left). A sum ⊕ is created, with its children decomposing variables according to the criterion defined by a binary tree (top right).

results from five state-of-the-art learning algorithms for PCs: LEARNSPN, STRUDEL, LEARNPSDD, XPC [Mauro et al. 2021], and PROMETHEUS [Jaini et al. 2018]. It should be noted that both PROMETHEUS and LEARNSPN produce less constrained PCs in terms of their structure, leading to possibly more expressive circuits (albeit with a more restricted range of inference tasks available to them). We report results for ensembles of STRUDEL, LEARNPSDD and XPC, which means that the PCs produced by these methods all fall under the same class of PCs generated by LEARNRP.

Surprisingly, we found that LEARNRP ranked, on average, second best against much more sophisticated competitors (see Table 1), only losing to PROMETHEUS. Apart from fitness of data, we also evaluated the learning time for LEARNRP and all other competitors except for PROMETHEUS, since no source code for it was available. This comparison can be visualized in Figure 5.7 and 5.8 of the dissertation, where we label LEARNRP-F as 100 iterations of mini-batch EM and 30 of full EM, LEARNRP-100 as 100 iterations of mini-batch EM, and LEARNRP-0 as the time for only learning the structure without fine-tuning weights. We found that our approach is orders of magnitude faster than most other techniques, even though we achieve better performance.

5. Conclusions

The objectives of our dissertation were three-fold. The first (minor) contribution, which we omit from this text for the sake of brevity, consisted of a systematic review of the literature of learning algorithms for probabilistic circuits, providing a better picture of the current state-of-the-art in PC learning. Our second contribution describes an algorithm for learning a specific class of PCs, called PSDDs, from both data and domain knowledge in the form of a propositional logic formula. We showed that our approach fared better compared to competitors when faced with low-data regimes yet remained competitive otherwise. In our third and last contribution, we sought to use random projections, a well-known artifice in density estimation literature, to construct PCs by a hierarchical

Dataset	LEARNSPN	STRUDEL	LEARNPSDD	XPC	PROMETHEUS	LEARNRP
NLTCS	-30.03	|-28.73|	-30.16	-31.02	**-27.91**	-28.66
MSNBC	-19.73	-16.38	-31.78	**-15.50**	-23.96	|-19.26|
KDD	-40.50	-41.50	-39.94	-40.91	**-39.80**	|-40.27|
PLANTS	|-250.68|	-254.41	-253.19	**-248.34**	-248.50	-254.15
BAUDIO	|-57.02|	-58.69	**-55.71**	-57.58	-56.47	-57.02
JESTER	-35.88	-34.99	-34.97	|-34.75|	-34.40	**-33.56**
BNETFLIX	-155.92	-154.47	-155.97	-153.75	|-154.17|	**-152.63**
ACCIDENTS	|-85.06|	-86.22	-89.61	-84.70	**-84.59**	-85.69
TRETAIL	-158.20	-155.33	-161.09	-153.67	|-155.21|	**-153.52**
PUMSB STAR	**-82.52**	-86.22	-88.01	-86.61	|-84.45|	-83.57
DNA	-75.98	-55.03	**-51.29**	-53.43	-52.80	|-52.92|
KOSAREK	-2.18	|-2.13|	**-2.11**	-2.15	-2.12	-2.14
MSWEB	-10.98	-10.68	**-10.52**	-10.77	-10.59	|-10.62|
BOOK	-6.11	**-6.04**	-6.04	-6.18	|-6.04|	-6.33
TMOVIE	-10.25	**-9.71**	|-9.89|	-9.93	-9.86	-9.90
CWEBKB	-6.11	-6.06	**-5.99**	|-6.05|	-6.01	-6.22
CR52	-12.97	|-12.98|	-13.02	-14.19	**-12.81**	-13.77
C20NG	|-24.78|	-24.12	-26.12	-26.06	**-22.75**	-26.12
BBC	|-52.48|	-53.67	-58.01	-54.82	-51.49	**-51.41**
AD	-11.04	-10.81	**-10.72**	-10.94	-10.87	|-10.84|
Avg. Rank	4.28 ± 1.50	3.75 ± 1.48	$\|3.58 \pm 2.14\|$	3.95 ± 1.61	$\mathbf{2.15 \pm 1.04}$	3.30 ± 1.65

Table 1. Average test log-likelihood benchmark and average rank with standard deviation (last row) of LEARNRP against competitors. In bold first place, underline second, \|between pipes\| third.

partitioning random method. Experiments showed our approach ranked second best on average, even though ours proved degrees of magnitude faster in terms of learning time.

References

Bova, S. (2016). Sdds are exponentially more succinct than obdds. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*.

Choi, A. and Darwiche, A. (2013). Dynamic minimization of sentential decision diagrams. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Choi, A., den Broeck, G. V., and Darwiche, A. (2015). Tractable learning for structured probability spaces: A case study in learning preference distributions. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Correia, A. H. C., Peharz, R., and de Campos, C. (2020). Joints in random forests. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*.

Dang, M., Vergari, A., and den Broeck, G. V. (2020). Strudel: Learning structured-decomposable probabilistic circuits. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models*.

Dasgupta, S. and Freund, Y. (2008). Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*.

Freund, Y., Dasgupta, S., Kabra, M., and Verma, N. (2008). Learning the structure of manifolds using random projections. In *Advances in Neural Information Processing Systems*.

Geh, R., Mauá, D., and Antonucci, A. (2020). Learning probabilistic sentential decision diagrams by sampling. In *Proceedings of the VIII Symposium on Knowledge Discovery, Mining and Learning*.

Geh, R. L. and Mauá, D. D. (2021). Fast and accurate learning of probabilistic circuits by random projections. In *The 4th Tractable Probabilistic Modeling Workshop*.

Geh, R. L. and Mauá, D. D. (2021). Learning probabilistic sentential decision diagrams under logic constraints by sampling and averaging. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*.

Gens, R. and Domingos, P. (2013). Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine Learning*.

Jaini, P., Ghose, A., and Poupart, P. (2018). Prometheus: Directly learning acyclic directed graph structures for sum-product networks. In *International Conference on Probabilistic Graphical Models*.

Kisa, D., den Broeck, G. V., Choi, A., and Darwiche, A. (2014). Probabilistic sentential decision diagrams. *Knowledge Representation and Reasoning Conference*.

Liang, Y., Bekker, J., and den Broeck, G. V. (2017). Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*.

Lowd, D. and Davis, J. (2010). Learning markov network structure with decision trees. In *2010 IEEE International Conference on Data Mining*.

Mauro, N. D., Gala, G., Iannotta, M., and Basile, T. M. A. (2021). Random probabilistic circuits. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*.

Monteith, K., Carroll, J. L., Seppi, K., and Martinez, T. (2011). Turning bayesian model averaging into bayesian model combination. In *The 2011 International Joint Conference on Neural Networks*.

Oztok, U. and Darwiche, A. (2015). A top-down compiler for sentential decision diagrams. In *Proceedings of the 24th International Conference on Artificial Intelligence*.

Shen, Y., Choi, A., and Darwiche, A. (2017). A tractable probabilistic model for subset selection. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*.

Smyth, P. and Wolpert, D. (1998). Stacked density estimation. In *Advances in Neural Information Processing Systems*.

Van Haaren, J. and Davis, J. (2012). Markov network structure learning: A randomized feature generation approach. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*.