# Probabilistic Circuits with Constraints via Convex Optimization

Soroush Ghandi, Benjamin Quost, Cassio de Campos

# Probabilistic Circuits with Constraints via Convex Optimization

Soroush Ghandi[1], Benjamin Quost[2], and Cassio de Campos[1]

[1] Eindhoven Unisversity of Technology
[2] University of Technology of Compiègne

**Abstract.** This work addresses integrating probabilistic propositional logic constraints into the distribution encoded by a probabilistic circuit (PC). PCs are a class of tractable models that allow efficient computations (such as conditional and marginal probabilities) while achieving state-of-the-art performance in some domains. The proposed approach takes both a PC and constraints as inputs, and outputs a new PC that satisfies the constraints. This is done efficiently via convex optimization without the need to retrain the entire model. Empirical evaluations indicate that the combination of constraints and PCs can have multiple use cases, including the improvement of model performance under scarce or incomplete data, as well as the enforcement of machine learning fairness measures into the model without compromising model fitness. We believe that these ideas will open possibilities for multiple other applications involving the combination of logics and deep probabilistic models.

**Keywords:** Probabilistic Circuits · Probabilistic Logic · Graphical Models.

## 1 Introduction

Generative probabilistic models typically aim to learn the joint probability distribution of data, in order to perform probabilistic inference and answer queries of interest. However, not all the probabilistic models are the same in that regard. Models like variational autoencoders (VAEs) [13] and generative adversarial networks (GANs) [9] possess exceptional modeling prowess; nevertheless, their ability to perform probabilistic inference such as marginalization and conditioning remains rather limited, due to tractability issues.

In contrast, *tractable* probabilistic models, such as probabilistic circuits (PCs), including the prominent sum-product networks (SPNs) [27,31], allow for a wider range of exact inferences, arguably at the expense of some fitting power. PCs fall within the family of probabilistic graphical models (PGMs), a class of models using a graph-based representation to encode high-dimensional distributions [14]. Unlike Bayesian networks, which have a notoriously high complexity for general queries [5], PCs can produce several types of inferences in polynomial time under arguably mild assumptions [33].

Learning a PC from data $\mathcal{D}$ is defined as specifying a PC that represents the probability distribution underlying $\mathcal{D}$. This active line of research has seen several meaningful proposals in the past few years, such as [1,6,7,8,11,12,16] and [17,18,20,22,24,28,29,30,32,34,35], but remains nevertheless open, given the difficulty of the task which involves both structure and parameter learning.

We address here the issue of *enhancing* a PC learned from data by using additional information and/or learning goals. To this end, we propose an approach for combining the PC with probabilistic propositional logic (PPL) constraints. More specifically, the approach takes a learned PC and updates (some of) its parameters in order to enforce the PPL constraints globally in the represented distribution. Our strategy can be seen as a "post-learning" method, which gives the advantages of versatility (existing models need not be retrained) as well as modularity: one may train a PC using any algorithm, as long as the resulting network keeps dependent variables (which may appear together in the same PPL constraint) together within the model; that is, they cannot appear factorized in the graph (further details are given in Section 3). This allows to build convex optimization problems (more precisely, constrained KL-divergence solvers) over parts of the distribution encoded in the PC so as to improve the corresponding model parameters via an efficient tractable method.

The benefits of having user-specified constraints are multi-fold. In order to illustrate them, we employ PPL constraints in a few (non-exhaustive) scenarios: (1) we improve the quality of models by enforcing that the yielded model matches the empirical marginal distributions under situations of (a) scarce data or (b) missing data; (2) we enforce fairness constraints into the model while at the same time avoiding a decrease in fitness. Overall, the experiments indicate that using PPL constraints often yields a better model (without compromising efficiency or accuracy), which is likely possible because of typical over-parametrizations that current large machine learning models impose. We emphasize that these applications of constraints are only a few examples of possible use, as we believe there are many other possibilities ahead to be tried.

## 2 Probabilistic circuits

Probabilistic circuits (PCs) are a family of distribution representations facilitating many exact and efficient inference routines (see [2] for a nice introduction). A PC encodes a probabilistic model over a collection of variables $\mathbf{X}$; it is structured as a rooted directed acyclic graph $\mathcal{G}$, containing three types of nodes: (i) distribution nodes, (ii) sum nodes, and (iii) product nodes. Distribution nodes are the leaves of the graph $\mathcal{G}$, while sum and product nodes are the internal nodes. Each distribution node (leaf) $v$ computes a probability distribution over some subset $\mathbf{X}' \subseteq \mathbf{X}$, i.e. an integrable function $p_v(\mathbf{x}') \colon \boldsymbol{\mathcal{X}}' \to \mathbb{R}^+$ from the sample space of $\mathbf{X}'$ to the non-negative real numbers. The *scope* of $v$ is the set of variables $\mathrm{sc}(v) := \mathbf{X}'$ over which the leaf computes a distribution. The scope of any internal node $v$ (sum or product) is recursively defined as $\mathrm{sc}(v) = \cup_{u \in \mathrm{ch}(v)} \mathrm{sc}(u)$, where $\mathrm{ch}(v)$ is the set containing the children of $v$. Sum

nodes compute convex combinations over their children, i.e. if $v$ is a sum node, then $v$ computes $v(\mathbf{x}) = \sum_{u \in \mathrm{ch}(v)} w_{v,u} u(\mathbf{x})$, where $w_{v,u} \geq 0$. In a normalized PC, we have $\sum_{u \in \mathrm{ch}(v)} w_{v,u} = 1$. Product nodes compute the product over their children, i.e. if $v$ is a product node, then $v(\mathbf{x}) = \prod_{u \in \mathrm{ch}(v)} u(\mathbf{x})$. The support of a node is the region where its associated function is strictly positive.

The main feature of PCs is that they facilitate a wide range of *tractable* inference routines, which go hand in hand with certain structural properties [2,4]: (i) a sum node $v$ is called *smooth* if its children have all the same scope: $\mathrm{sc}(u) = \mathrm{sc}(u')$, for any $u, u' \in \mathrm{ch}(v)$; (ii) a product node $v$ is called *decomposable* if its children have non-overlapping scopes: $\mathrm{sc}(u) \cap \mathrm{sc}(u') = \emptyset$, for any $u, u' \in \mathrm{ch}(v)$, $u \neq u'$; (iii) a node is *consistent* if its support is non-empty. A PC is smooth (respectively decomposable) if all its sum (respectively product) nodes are smooth (respectively decomposable). A PC is consistent if all its nodes are consistent. The distribution $p(\mathrm{sc}(v))$ represented by a node $v$ in the PC is the function computed by the rules of the previous paragraph, and can be evaluated with a feed-forward pass. In order to ensure the tractability of queries, we can rely on smoothness and decomposability, but we also need leaf distribution nodes to compute inferences efficiently. This is a reason for many proposed PCs in the literature to assume that leaf nodes are univariate with some known distribution, such as Bernoulli, categorical, Gaussian, etc. Now, assume that we wish to compute a *marginal query*, that is, to evaluate the probability value over $\mathbf{X}_o \subset \mathbf{X}$ for evidence $\mathbf{X}_o = \mathbf{x}_o$, while marginalising $\mathbf{X}_{\neg o} = \mathbf{X} \setminus \mathbf{X}_o$. In smooth and decomposable PCs, this task reduces to performing marginalization at the leaves [25]: for each leaf $v$, one marginalizes $\mathrm{sc}(v) \cap \mathbf{X}_{\neg o}$, and evaluates it for the values corresponding to $\mathrm{sc}(v) \cap \mathbf{X}_o$. The desired marginal $p_{\mathbf{X}_o}(\mathbf{x}_o)$ results from evaluating internal nodes as in computing the complete distribution. Smoothness and consistency are sufficient to guarantee that the function of a PC represents a distribution. We also assume PCs are normalized.

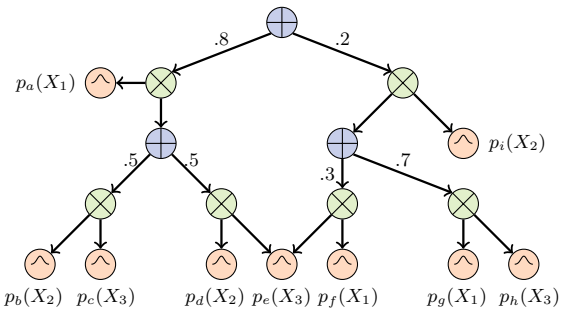



Fig. 1: Example of PC with variables $X_1, \ldots, X_3$. Sum nodes are in blue, product nodes in green, distribution leaf nodes in salmon. In this example, all leaf nodes are univariate. Subscriptions on each $p$ in the figure are used to indicate that those are different leaf distributions (even if sometimes over the same variable).

An important feature of normalized valid PCs is their interpretation as hierarchical, discrete mixture models [23,36]:

$$p(\mathbf{x}) = \sum_z p(\mathbf{x}|z)p(z) = \sum_z p_z(\mathbf{x})p(z), \tag{1}$$

where $Z$ is a discrete latent vector, which originates from the sum nodes of the structure. The number of states of $Z$, and thus of represented mixture components $p(\mathbf{x}|z)$, grows exponentially in the depth of the PC [21,36]. While we use this notation here and throughout the paper, we do not run computations directly in this formulation, but instead we make use of the graphical structure of the PC in order to perform efficient tractable inference, as usual for PCs.

## 3 Probabilistic circuits with constraints

We assume that a normalized valid PC has been produced (learned from data, designed by a human, etc.) over a domain with variables $\mathbf{X}$. Such a PC induces a joint distribution $p(\mathbf{X})$. The goal is to enforce some (linear) probabilistic propositional logic (PPL) constraints upon $p$. We work with constraints of the form:

$$\sum_{i_c} \tau_{i_c} \cdot p(F_{i_c}) \leq \alpha_c, \tag{2}$$

where each $F_{i_c}$ is a propositional logic formula defined over Boolean variables $\mathbf{X}_c = \{X_{j_c}\}_{\forall j_c} \subseteq \mathbf{X}$, $\tau_{i_c}, \alpha_c$ are real numbers, $j_c$ (and $i_c$) are indexes of variables (terms) of the constraint $c$, and $c \in C$ is an index over a set of PPL constraints. We assume that constraints are placed in buckets $B$ (mathematically a bucket can be simply an index set indicating the constraints it contains) such that $\mathbf{X}_{B_1} \cap \mathbf{X}_{B_2} = \emptyset$ for all distinct buckets $B_1, B_2$, where $\mathbf{X}_B = \cup_{c \in B} \mathbf{X}_c$ is the union of all variables appearing in a constraint inside bucket $B$. If any $\mathbf{X}_{c_1}$ and $\mathbf{X}_{c_2}$ of two constraints are not disjoint, then we put them together into the same bucket, so as to ensure that buckets have mutually exclusive sets of variables.

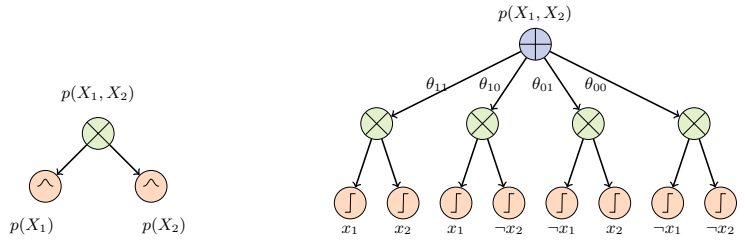
The constraints in each bucket $B$ may obviously create dependencies among the variables $\mathbf{X}_B$. In order to avoid inconsistencies between such dependencies and those arising from the graph structure of the PC, we require that the variables in a bucket appear together in nodes of the model, that is, for any $v, B$, $X \in \mathbf{X}_B \cap \mathrm{sc}(v) \Rightarrow \mathbf{X}_B \subseteq \mathrm{sc}(v)$. Therefore, Equation (1) can be recast as

$$p(\mathbf{X}) = \sum_z p(z) \prod_B p_z(\mathbf{X}_B) \prod_{X_i \in \mathbf{X} \setminus \cup \mathbf{X}_B} p_z(X_i), \tag{3}$$

where $p_z(\mathbf{X}_B)$ is a categorical distribution—note that notations $p_z(\mathbf{X}_B)$ and $p_z(X_i)$ employ a slight abuse, as the function itself is "aware" of the indexes of the variables in their arguments and may vary accordingly, for example, $p_z(X_i)$ is also a function of $i$ and not only of $X_i$; the same abuse holds elsewhere, for example in Expression (2). Equation (3) basically decomposes $p_z(\mathbf{x})$ of Equation (1) into components that involve PPL variables (which remain together) and

the other variables, which are assumed to be represented by univariate leaf-node distributions.

For ease of exposure, but also for the sake of compatibility with software that only deals with univariate leaf distributions, one can replace categorical distributions in leaf nodes with new sub-PCs. If one assumes independence among scope variables, then a product node with univariate leaf nodes suffices. If one wants to fully exploit the categorical distribution node, then a sum node with one child per parameter of the categorical distribution can be used. Figure 2 gives an example of dealing with a categorical "joint" distribution over two Boolean variables $X_1, X_2$. Figure 2a shows the independent case, while Figure 2b shows the joint approach to represent the distribution for $X_1$ and $X_2$. The reader may have already noticed that large buckets of constraints will force the model to keep together many variables, which can be problematic as the number of parameters of the categorical joint distribution of all variables in a bucket $B$ will grow exponentially in $|\mathbf{X}_B|$ (as in Figure 2b, all possible configurations of $\mathbf{X}_B$ would be listed). We will discuss this later, and ask the reader to assume that buckets (or equivalently scopes of leaf distribution nodes) are not large.




(a) Leaf structure using independence for $X_1$ and $X_2$.

(b) Leaf replacement structure without assuming independence of $X_1$ and $X_2$.

Fig. 2: Leaf distribution replacement structures that can be used to represent the parameters of a categorical variable for a bucket $B$ with $\mathbf{X}_B = \{X_1, X_2\}$.

Given a PC representing $p(\mathbf{X})$ and PPL constraints, we aim to find an *efficient* approach to discover a new PC inducing a distribution $q^*(\mathbf{X})$ that is close to $p(\mathbf{X})$ while respecting the PPL constraints:

$$\begin{aligned} q^*(\mathbf{X}) &= \operatorname*{argmin}_{q(\mathbf{X})} \mathcal{L}(p(\mathbf{X}), q(\mathbf{X})) \\ \text{s.t.} \quad & \forall B, \forall c \in B : \sum_{i_c} \tau_{i_c} \cdot q(F_{i_c}) \leq \alpha'_c, , \qquad q(\mathbf{X}) \in \mathcal{P}(\mathbf{X}) \,, \end{aligned} \tag{4}$$

where $\mathcal{L}$ measures the discrepancy between two distributions, $\mathcal{P}(\mathbf{X})$ denotes a set of probability distributions over $\mathbf{X}$ that can be represented by a PC, and $B$ are buckets of PPL constraints. Optimization (4) is impractical, as it amounts to solving a complex optimization problem to search over $\mathcal{P}(\mathbf{X})$, even if $\mathcal{L}$ is simple

enough. Therefore, we exploit the PC on which $p(\mathbf{X})$ was estimated, in order to constraint the search space of $q(\mathbf{X})$: we enforce $q(\mathbf{X})$ to have a shape similar to $p(\mathbf{X})$, i.e.

$$q(\mathbf{X}) = \sum_z p(z) \prod_B q_z(\mathbf{X}_B) \prod_{X_i \in \mathbf{X} \setminus \cup \mathbf{X}_B} p_z(X_i), \tag{5}$$

that is, only $\prod_B q_z(\mathbf{X}_B)$ will differ from the specification of $p(\mathbf{X})$. Plainly put, we only refine the distributions in the leaf nodes of the PC. Moreover, we use the Kullback-Leibler divergence $\mathcal{L}(p(\mathbf{X}), q(\mathbf{X})) = H(p(\mathbf{X}), q(\mathbf{X})) - H(p(\mathbf{X}))$ as discrepancy measure, where $H(\cdot)$ is the entropy and $H(\cdot, \cdot)$ the cross entropy. Clearly, we can focus on the cross entropy only, as the second term does not contain $q(\mathbf{X})$. Our first result is an upper bound on the cross entropy which allows us to run the optimization efficiently. The bound on the cross-entropy establishes an upper bound on the KL-divergence between $p(\mathbf{X})$ and $q(\mathbf{X})$.

**Theorem 1.** *Assume a PC representing a distribution $p(\mathbf{X})$ as in Equation (3) and PPL constraints as in Equation (2) (placed in disjoint buckets $B$) are given. Assume that $q(\mathbf{X})$ is a distribution induced by a PC with form as in Equation (5). Then, $H(p(\mathbf{X}), q(\mathbf{X})) \leq \sum_B \mathbb{E}_z[H(p_z(\mathbf{X}_B), q_z(\mathbf{X}_B))] + H(p(\mathbf{X}', Z))$, where $\mathbf{X}'$ are the variables not appearing in constraints.*

*Proof.* Note that we are particularly interested in terms with parameters in $q(\mathbf{X})$, as they will be optimized later. First, recall that

$$-H(p(\mathbf{X}), q(\mathbf{X})) = \sum_{\mathbf{x}} p(\mathbf{x}) \log q(\mathbf{x}) \,, \tag{6}$$

and for any configuration $\mathbf{x}$ of $\mathbf{X}$ and for any arbitrary $z_0 \in Z$, we have:

$$q(\mathbf{x}) = \sum_z p(z) p'_z(\mathbf{x}') \prod_B q_z(\mathbf{x}_B) \geq p(z_0) p'_{z_0}(\mathbf{x}') \prod_B q_{z_0}(\mathbf{x}_B) \,, \tag{7}$$

which holds because all terms are non-negative, where $\mathbf{X}' = \mathbf{X} \setminus \cup \mathbf{X}_B$ (variables not in any constraint), and $p'_z(\mathbf{X}') = \prod_{X_i \in \mathbf{X}'} p_z(X_i)$, for given $z$. By substituting (7) into (6), we can establish a lower bound on the negative cross-entropy term:

$$-H(p(\mathbf{X}), q(\mathbf{X})) \geq \sum_{\mathbf{x}} p(\mathbf{x}) \log[p(z_0) p'_{z_0}(\mathbf{x}') \prod_B q_{z_0}(\mathbf{x}_B)] \tag{8}$$

$$= \sum_{\mathbf{x}} \sum_z p(z) p'_z(\mathbf{x}') \prod_\beta p_z(\mathbf{x}_\beta) \log[p(z) p'_z(\mathbf{x}') \prod_B q_z(\mathbf{x}_B)], \tag{9}$$

with the arbitrary $z_0 \in Z$ in Expression (8) being chosen to be equal to $z$ for each of the elements in the summation over $z$, thus resulting in Expression (9). Then, we can split Expression (9) into two parts (using the log of products as sum of logs), where only the second term depends on $q(\mathbf{X})$:

$$-H(p(\mathbf{X}), q(\mathbf{X})) \geq \sum_{\mathbf{x}} \sum_z p(z) p'_z(\mathbf{x}') \prod_\beta p_z(\mathbf{x}_\beta) \log[p(z) p'_z(\mathbf{x}')]$$

$$+ \sum_{\mathbf{x}} \sum_z p(z) p'_z(\mathbf{x}') \prod_\beta p_z(\mathbf{x}_\beta) \log[\prod_B q_z(\mathbf{x}_B)]. \tag{10}$$

The first term in the RHS of Expression (10) can be reduced to $-H(p(\mathbf{X}', Z))$; it does not depend on $q(\mathbf{X})$, and will consequently not be analyzed further. The second term in the RHS can be manipulated as

$$
\begin{aligned}
&= \sum_B \sum_{\mathbf{x}} \sum_z p(z) p'_z(\mathbf{x}') \prod_\beta p_z(\mathbf{x}_\beta) \log q_z(\mathbf{x}_B) \\
&= \sum_B \sum_{\substack{\mathbf{x}_{B_t} \\ \forall t}} \sum_{\mathbf{x}'} \sum_z p(z) p'_z(\mathbf{x}') \prod_\beta p_z(\mathbf{x}_\beta) \log q_z(\mathbf{x}_B) \\
&= \sum_B \sum_z p(z) \sum_{\substack{\mathbf{x}_{B_t} \\ \forall t}} \prod_\beta p_z(\mathbf{x}_\beta) \log q_z(\mathbf{x}_B) \sum_{\mathbf{x}'} p'_z(\mathbf{x}') \\
&= \sum_B \sum_z p(z) \sum_{\substack{\mathbf{x}_{B_t} \\ \forall t}} \prod_\beta p_z(\mathbf{x}_\beta) \log q_z(\mathbf{x}_B) \\
&= \sum_B \sum_z p(z) \sum_{\mathbf{x}_B} p_z(\mathbf{x}_B) \log q_z(\mathbf{x}_B) \sum_{\substack{\mathbf{x}_{B_t} \\ \forall t, B_t \neq B}} \prod_{\beta \neq B} p_z(\mathbf{x}_\beta) \\
&= \sum_B \sum_z p(z) \sum_{\mathbf{x}_B} p_z(\mathbf{x}_B) \log q_z(\mathbf{x}_B) \prod_{\beta \neq B} \sum_{\mathbf{x}_\beta} p_z(\mathbf{x}_\beta) \\
&= \sum_B \sum_z p(z) \sum_{\mathbf{x}_B} p_z(\mathbf{x}_B) \log q_z(\mathbf{x}_B) \\
&= -\sum_B \mathbb{E}_z[H(p_z(\mathbf{X}_B), q_z(\mathbf{X}_B))]. \qquad (11)
\end{aligned}
$$

Hence, $-H(p(\mathbf{X}), q(\mathbf{X})) \geq -H(p(\mathbf{X}', Z)) - \sum_B \mathbb{E}_z[H(p_z(\mathbf{X}_B), q_z(\mathbf{X}_B))]$, and the result follows. □

Thus, we can adapt the PC at hand using the specified constraints by minimizing the upper bound on the desired discrepancy, leaving aside the term $H(p(\mathbf{X}', Z))$ which does not involve $q(\mathbf{X})$:

$$
\begin{aligned}
q^*(\mathbf{X}) = \operatorname*{argmin}_{q(\mathbf{X})} & \sum_B \mathbb{E}_z[H(p_z(\mathbf{X}_B), q_z(\mathbf{X}_B))] \\
\text{s.t.} \quad & \forall B, \forall c \in B : \sum_{i_c} \tau_{i_c} \cdot q(F_{i_c}) \leq \alpha_c \,, \qquad q(\mathbf{X}) \in \mathcal{P}(\mathbf{X}) \,,
\end{aligned} \qquad (12)
$$

Theorem 2 sheds light on the complexity of the procedure; it is based on considerably mild assumptions, as long as buckets do not involve too many variables.

**Theorem 2.** *Given the same inputs as Theorem 1, and assuming $|\mathbf{X}_B| \leq k$ for all buckets $B$, the solution $q^*$ to the optimization in Optimization* (12) *can be found in polynomial time in the input size (while possibly exponential in $k$).*

*Proof.* The objective function is a sum over buckets containing (mutually) disjoint sets of variables, so we can solve Optimization (12) by solving separate

optimizations for each bucket $B$:

$$\begin{aligned} \forall B: \quad q^*(\mathbf{X}_B) = \operatorname*{argmin}_{q(\mathbf{X}_B)} & -\sum_z p(z) \sum_{\mathbf{x}_B} p_z(\mathbf{x}_B) \log q_z(\mathbf{x}_B) \\ \text{s.t.} \quad & \forall c \in B : \sum_{i_c} \tau_{i_c} \cdot q(F_{i_c}) \leq \alpha_c \,, \qquad q(\mathbf{X}_B) \in \mathcal{P}(\mathbf{X}_B) \,. \end{aligned} \tag{13}$$

(Note the abuse of notation here, as $q^*(\mathbf{X}_B)$ is used to indicate the parameters of model $q^*(\mathbf{X})$ that are associated with leaf nodes containing variables $\mathbf{X}_B$.) Optimization (13) can be solved for each $B$ using convex optimization solvers, which run in polynomial time in the size of their inputs (and can be very efficient in practice). The values $p_z(\mathbf{x}_B)$ and $p(z)$ are fixed during the optimization and can be obtained directly from the PC model representing $p(\mathbf{X})$.

Assuming that $q_z(\mathbf{x}_B)$ is parameterized using values $\theta_{z,\mathbf{x}_B}$ representing a categorical distribution over $\mathbf{X}_B$ conditional to $Z = z$ (same structure as in Figure 2b), we obtain Optimization (14) for each bucket $B$. Note that in Optimization (14), each PPL formula $F_{i_c}$ is written down as the sum of the worlds that satisfy the formula (we can query $F_{i_c}(\mathbf{x}_B)$ to see if each $\mathbf{x}_B$ satisfies $F_{i_c}$, assuming $F_{i_c} = 1$ if so, and zero otherwise). Optimization (14) also connects the local parameters $\theta_{z,\mathbf{x}_B}$ with the marginal value of the candidate PC for $\mathbf{x}_B$, that is, $q(\mathbf{x}_B) = \sum_z p(z)\theta_{z,\mathbf{x}_B}$, which appears in the last expression of the optimization problem: thus, the imposed constraint is a global constraint in the joint model $q(\mathbf{X})$, and not simply a local constraint in the local parameters. Note also that $p(z) = q(z)$ (by assumption from Expression (5)).

$$\begin{aligned} \forall B: \quad q^*(\mathbf{X}_B) = \operatorname*{argmin}_{\theta_{z,\mathbf{x}_B}:\ \forall z,\mathbf{x}_B} & -\sum_z p(z) \sum_{\mathbf{x}_B} p_z(\mathbf{x}_B) \log \theta_{z,\mathbf{x}_B} \,, \\ & \forall z, \mathbf{x}_B : \ \theta_{z,\mathbf{x}_B} \geq 0 \,, \qquad \forall z : \ 1 = \sum_{\mathbf{x}_B} \theta_{z,\mathbf{x}_B} \,, \\ \text{s.t.} \quad & \forall c \in B : \sum_{i_c} \tau_{i_c} \cdot \left( \sum_{\mathbf{x}_B} F_{i_c}(\mathbf{x}_B) (\sum_z p(z) \theta_{z,\mathbf{x}_B}) \right) \leq \alpha_c \,. \end{aligned} \tag{14}$$

Optimization (14), and hence Optimizations (4) and (13), will have a feasible solution so long as the set of PPL constraints has a feasible solution. This can be checked using linear programming using the constraints in Optimization (14). Therefore, it can be checked in polynomial time if the user provided an infeasible set of constraints. The number of buckets is bounded by the number of constraints $C$, which therefore also bounds the number of optimization calls. The optimization for bucket $B$ has $O(|Z| \cdot |\mathbf{X}_B|)$ variables and $O(C \cdot |\mathbf{X}_B|)$ constraints (those are all very loose bounds), which is asymptotically bounded by the PC size plus constraints' size (that is, the input size), and convex optimization can be solved in polynomial time in the number of variables and constraints. □

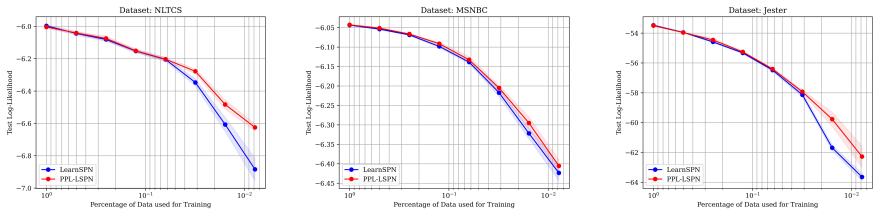



Fig. 3: LearnSPN vs. (constrained) PPL-LSPN trained on scarce datasets.

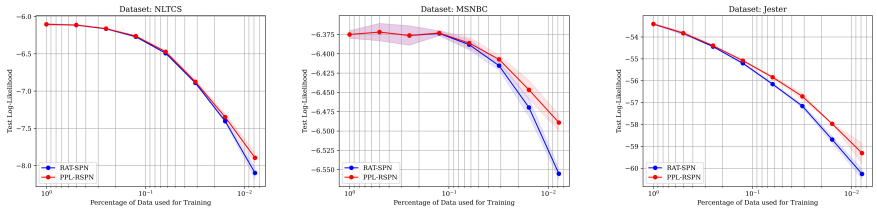



Fig. 4: RAT-SPN vs. (constrained) PPL-RSPN trained on scarce datasets.

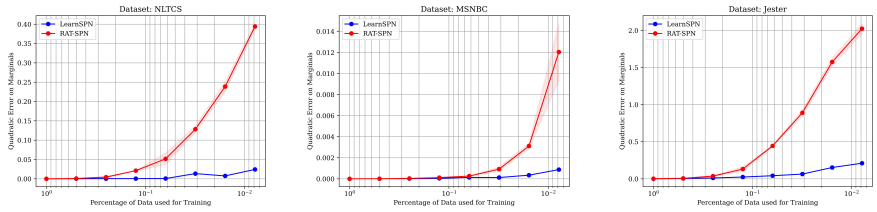



Fig. 5: Sum of quadratic differences on marginal parameters between the models with and without marginal constraints, when trained on scarce data. Constraints clearly refine the model more strongly for RAT-SPNs than for LearnSPN. Standard RAT-SPN marginals are very far from matching the empirical marginal distributions (data not shown).

## 4 Experiments

We conduct a series of experiments to illustrate how constrained optimization can be utilized to shape a desirable performance or behavior in PCs. For the sake of this illustration, we focus on two use cases of constraints, namely (i) constraints over marginals of the distribution, and (ii) constraints for enforcing fairness in distributions.

The idea behind constraints on marginals is to adjust a probabilistic model to match the empirical marginals of $\mathbf{X}$ on data $\mathcal{D}$. Typically, it is easier to accurately learn the marginal distributions over single variables rather than the whole joint distribution, in particular in cases when $\mathcal{D}$ is scarce and/or incom-

plete. In Sections 4.1 and 4.2 we explore how the use of constraints on empirical marginals affects the performance/behavior of learned PCs.

In Section 4.3, we investigate the impact of applying our method to a variety of fairness-specific classification tasks by adding fairness in the form of PPL constraints into PCs. Most common PC learning methods are not known to be inherently compatible with fairness, and being able to apply fairness constraints to PCs opens the door to utilizing these probabilistic models in areas where fairness is a priority, thus extending their domain of applicability.

Throughout the experiments, we utilize both LearnSPN [8] and RAT-SPN [26] for learning baseline PC models (one could also handcraft a PC for a purpose and use it with our approach, as we are not bound by the way the PC was obtained). We use the original implementation of RAT-SPN[3], and the implementation of [3][4] for LearnSPN. For LearnSPN, statistical test significance and the Laplace smoothing parameter are set to 0.01 over all experiments. For RAT-SPN, hyperparameters that correspond to the region graph structure are set as follows: the number of recursive splits is 10, the depth of each recursive split is 2, the number of input distributions in each partition is 8, and the number of sum nodes per partition is 8; all the other hyperparameters are set to their defaults. For each experiment, RAT-SPN is trained for 20 epochs.

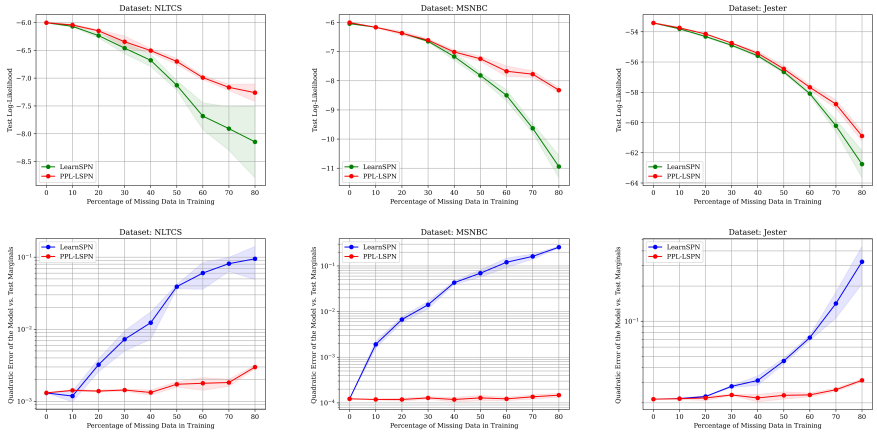



Fig. 6: LearnSPN vs. (constrained) PPL-LSPN trained on datasets with MCAR missing values. Test log-likelihood measures the joint fitness, while quadratic error shows the quality of the marginals of the model with respect to the marginals of test data, with clear superior accuracy after constraints are imposed.

[3] https://github.com/cambridge-mlg/RAT-SPN
[4] https://github.com/AlCorreia/GeFs

### 4.1 Scarce datasets

We carry out this experiment on three different binary datasets, namely NLTCS, MSNBC, and Jester [19]. Our goal is to illustrate how additional information pertaining to the empirical marginal distributions can be incorporated into the PC so as to compensate for data scarcity. In order to simulate scarce data, we randomly subsample each dataset with a varying number of data instances. We use this subsample to train the PC model using LearnSPN and RAT-SPN. We then improve the model using the procedure described above so as to match the empirical marginal distributions: we add PPL constraints of the form $p(X_i = 1) = \alpha_i$ for every variable $X_i$, which we enforce globally into the model. The test log-likelihood results are given in Figures 3 and 4. The enhanced models obtained by applying the constrained optimization are called PPL-LSPN (variant of the LearnSPN baseline), and PPL-RSPN (variant of RAT-SPN).

As can be seen, PPL-LSPN (Figure 3) and PPL-RSPN (Figure 4) slightly outperform LearnSPN (resp. RAT-SPN) as the training data become scarcer. This performance gain (in terms of testing data log-likelihood) is not similar across all datasets, which we attribute to the relative amount of information captured by the marginals. Arguably, in smaller datasets (in terms of the number of samples), matching marginals should lead to larger performance gains, as marginals encode a relatively larger amount of information. More importantly, matching marginals did not harm joint accuracy.

We argue that marginal matching is even more advantageous to PPL-RSPN compared to PPL-LSPN, as the learned marginals are far more erroneous in the case of RAT-SPN. Figure 5 displays the increase in quadratic error induced by *not* matching marginals, for both LearnSPN and RAT-SPN. Clearly, a large gap can be observed between LearnSPN and RAT-SPN on scarce data. Somewhat to our surprise, estimated marginals in RAT-SPN are far off (also when compared to LearnSPN), making that model useless for marginal inference unless the constraints are imposed. Hence, and in particular for RAT-SPN, matching marginals leads to a strong improvement on the marginals themselves while (only but still) slightly improving the test joint likelihood.

### 4.2 Experiments with missing values

We again use the three binary datasets above (NLCTS, MSNBC, and Jester). In order to simulate missing data, we train the baseline PC (via LearnSPN) in a missing completely at random (MCAR) setting, by removing entries completely at random from the data tables. After the models are trained, we enhance the learned distribution to match the training data marginals using the proposed approach. Note that the current implementation of RAT-SPN mimics the effect of missing data with dropout layers (which is different from learning in presence of missing values); as well, the original version of RAT-SPN is not equipped to deal with missing data at training time, but can be easily tweaked for that purpose. We therefore focus these experiments on models trained with LearnSPN.

Results with MCAR data are summarized in Figure 6. The top plots show the joint testing data log-likelihood, while the bottom plots show the difference in the testing data marginal distributions (whose gains are very clear). We can see that in every experiment, as the proportion of missing values increases, the PC enhanced using constraints outperforms the base model, which suggests that marginal matching can be considered as an effective way to deal with missing data, potentially as an alternative to data imputation.

### 4.3 Fairness experiments

We investigate the impact of imposing fairness constraints in PCs. For each experiment, we assume variables $\mathbf{X}$ which comprise a binary class/target variable $Y \in \mathbf{X}$ and a binary protected attribute $X' \in \mathbf{X}, X' \neq Y$. Our objective is to improve the distribution learned via a PC towards fairness for the protected attribute when predicting class labels. We consider statistical parity as our measure of fairness (we use this as an example; we will not debate on fairness measures, since it is not the main focus of the paper). The corresponding fairness constraint is $p(y = 1|x' = 1) = p(y = 1|x' = 0)$. It is clear that this constraint is not of the form $\sum_i \tau_i \cdot p(F_i) \leq \alpha$. It would actually induce a non-linear constraint and the convex optimization could not be directly applied. However, we can lift the optimization problem to a higher dimension by including a new unknown $\beta$ where we take $p(y = 1|x' = 1) = p(y = 1|x' = 0) = \beta$. This latter can be decomposed into two separate linear constraints in the desired form:

$$\begin{aligned} p((y = 1) \wedge (x' = 1)) - \beta \cdot p(x' = 1) = 0, \\ p((y = 1) \wedge (x' = 0)) - \beta \cdot p(x' = 0) = 0; \end{aligned} \tag{15}$$

and as long as $\beta$ is fixed, the optimization can be carried out to impose the constraints in Equation (15) to a learned PC using the proposed approach. In order to solve for $\beta$, we simply carry out an exhaustive search over candidate values between 0 and 1, retaining the best based on the performance of each resulting PC (obviously, this search procedure is reasonable for a single unknown $\beta$, or at most a few; otherwise, a smarter strategy would be required).

We consider six different classification datasets commonly used in fairness-aware machine learning, namely Adult, German Credit, Bank Marketing, Dutch Census, Credit Card Clients, and Law School [15]. As in Section 4.1, we refer to the variants as PPL-LSPN (for LearnSPN) and PPL-RSPN (for RAT-SPN). The details regarding the pre-processing of each dataset are provided in the appendix. The results are displayed in Tables 1 and 2. Not only PPL-LSPN and PPL-RSPN are able to achieve a "fair" distribution w.r.t the protected attribute, but they also manage to do so without losing much of their representation power compared to LearnSPN or RAT-SPN, that is, the test likelihood and 0-1 accuracy are barely affected while statistical parity is enforced by the use of constraints. We stress out that our procedure being a post-processing of the PC at hand, models already trained and potentially in use in applications could be enhanced without the need of re-training from scratch.

Table 1: Classification with LearnSPN vs. PPL-LSPN enforcing statistical parity via constraints.

| Dataset | Protected Attribute | Method | Test LL | Accuracy | Statistical Parity |
|---|---|---|---|---|---|
| **Adult** | **Sex** | LearnSPN | -13.614 | 0.8256 | 0.1754 |
| | | PPL-LSPN | -13.764 | 0.7946 | 0.0 |
| **German Credit** | **Sex** | LearnSPN | -22.802 | 0.6993 | -0.0171 |
| | | PPL-LSPN | -23.075 | 0.704 | 0.0 |
| **Bank Marketing** | **Marital Status** | LearnSPN | -16.448 | 0.8957 | -0.0305 |
| | | PPL-LSPN | -16.493 | 0.8949 | 0.0 |
| **Dutch Census** | **Sex** | LearnSPN | -9.801 | 0.8141 | 0.2520 |
| | | PPL-LSPN | -9.947 | 0.7359 | 0.0 |
| **Cr. Card Clients** | **Sex** | LearnSPN | -22.505 | 0.8164 | 0.0185 |
| | | PPL-LSPN | -22.539 | 0.8035 | 0.0 |
| **Law School** | **Race** | LearnSPN | -11.800 | 0.9076 | -0.3012 |
| | | PPL-LSPN | -11.845 | 0.9013 | 0.0 |

Table 2: Classification with RAT-SPN vs. PPL-RSPN enforcing statistical parity via constraints.

| Dataset | Protected Attribute | Method | Test LL | Accuracy | Statistical Parity |
|---|---|---|---|---|---|
| **Adult** | **Sex** | RAT-SPN | -7.767 | 0.8193 | 0.2000 |
| | | PPL-RSPN | -7.796 | 0.8148 | 0.0 |
| **German Credit** | **Sex** | RAT-SPN | -28.752 | 0.745 | -0.0309 |
| | | PPL-RSPN | -28.756 | 0.745 | 0.0 |
| **Bank Marketing** | **Marital Status** | RAT-SPN | -13.736 | 0.8820 | -0.0388 |
| | | PPL-RSPN | -13.739 | 0.8788 | 0.0 |
| **Dutch Census** | **Sex** | RAT-SPN | -12.880 | 0.7888 | 0.2620 |
| | | PPL-RSPN | -12.923 | 0.7629 | 0.0 |
| **Cr. Card Clients** | **Sex** | RAT-SPN | -3.998 | 0.7838 | 0.0053 |
| | | PPL-RSPN | -3.998 | 0.7838 | 0.0 |
| **Law School** | **Race** | RAT-SPN | -7.274 | 0.9050 | -0.2054 |
| | | PPL-RSPN | -7.294 | 0.9034 | 0.0 |

## 5 Conclusions and future work

We introduce a novel approach that allows to incorporate probabilistic propositional logic (PPL) constraints into a (pre-trained) probabilistic circuit (PC), so that the distribution encoded by the PC respects the constraints. We explain our design choices which allow for achieving tractable learning and inferences while ensuring that PPL constraints are satisfied. We also develop theoretical foundations that explain the feasibility of the optimization and how to reach an optimal solution in computationally tractable (polynomial) time. Experiments illustrate how we can take PCs and enhance them into better PCs that can

be applied to practical scenarios, for example by applying fairness measures to the learned distribution and by (arguably) better handling missing values in the training data.

We make space for a couple of reflections. The goal of this research is to enhance machine learning models with probabilistic logic assessments, in the same spirit as neurosymbolic AI. We found out that PC models are already over-parameterized: thus, one can better tune the parameters in order to satisfy external constraints. The first obvious idea is to do so via some variation of Expectation-Maximization or gradient methods, putting violation of constraints as (strong) penalties. However, it is not guaranteed that constraints are fully enforced; we therefore see that avenue as a great direction to investigate, even though the solution is likely to differ from the one described here. We managed to find a way to improve PCs a posteriori (without retraining) and efficiently (the optimization can run exactly and fast with modern convex optimization solvers). This choice comes at the expense of being able to only change the parameters of leaf distribution nodes; this—quite surprisingly—turns out to be enough to precisely enforce the constraints globally on the joint distribution while not losing model fitness. Moreover, we have no intention to claim that we are (or not) obtaining state-of-the-art results. This is an investigation of the combination of constraints into circuits, which we consider overall successful (but obviously not without limitations). We see many possibilities with that. We are aware that the bucket size limitation is a serious complication, but creative experiments show that there may be many interesting problems to solve even under such limitation. Moreover, we know that the limitation can be mitigated by using some smarter parametrization of the local distributions: this direction is definitely worthwhile, although it may lead to a decrease in accuracy and will likely not provide the same guarantees as we currently have.

Beyond these research directions, the paper opens doors for future work, as the desire to combine probabilistic logic constraints and deep machine learning methods is immense. Possible immediate avenues include extending the applicability of constraints on continuous and mixed variables, applying constraints to new tasks such as other forms of fairness measures (for instance, equalized odds [10]) in order to improve already learned PCs, improving the trade-off between accuracy and efficiency by using different optimizers, and considering extensions beyond consistent and valid PCs, to name but a few.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Adel, T., Balduzzi, D., Ghodsi, A.: Learning the structure of sum-product networks via an svd-based algorithm. In: Uncertainty in Artificial Intelligence (2015), https://api.semanticscholar.org/CorpusID:15429402
2. Van den Broeck, G., Di Mauro, N., Vergari, A.: Tractable probabilistic models: Representations, algorithms, learning, and applications. http://web.cs.ucla.

edu/~guyvdb/slides/TPMTutorialUAI19.pdf (2019), tutorial at Uncertainty in Artificial Intelligence (UAI) 2019
3. Correia, A., Peharz, R., de Campos, C.P.: Joints in random forests. Advances in Neural Information Processing Systems (NeurIPS) **33**, 11404–11415 (2020)
4. Darwiche, A.: A differential approach to inference in Bayesian networks. Journal of the ACM **50**(3), 280–305 (2003)
5. De Campos, C.P.: New complexity results for map in bayesian networks. In: International Joint Conference on Artificial Intelligence (IJCAI). vol. 11, pp. 2100–2106. Citeseer (2011)
6. Dennis, A., Ventura, D.: Learning the architecture of sum-product networks using clustering on variables. Advances in Neural Information Processing Systems **25** (2012)
7. Di Mauro, N., Vergari, A., Basile, T.M., Esposito, F.: Fast and accurate density estimation with extremely randomized cutset networks. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part I 10. pp. 203–219. Springer (2017)
8. Gens, R., Domingos, P.: Learning the structure of sum-product networks. In: International Conference on Machine Learning. pp. 873–880. PMLR (2013)
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Communications of the ACM **63**(11), 139–144 (2020)
10. Hardt, M., Price, E., Srebro, N.: Equality of opportunity in supervised learning. Advances in Neural Information Processing Systems **29** (2016)
11. Hsu, W., Kalra, A., Poupart, P.: Online structure learning for sum-product networks with gaussian leaves. arXiv preprint arXiv:1701.05265 (2017)
12. Kalra, A., Rashwan, A., Hsu, W.S., Poupart, P., Doshi, P., Trimponias, G.: Online structure learning for feed-forward and recurrent sum-product networks. Advances in Neural Information Processing Systems **31** (2018)
13. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
14. Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT press (2009)
15. Le Quy, T., Roy, A., Iosifidis, V., Zhang, W., Ntoutsi, E.: A survey on datasets for fairness-aware machine learning. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **12**(3), e1452 (2022)
16. Lee, S.W., Watkins, C., Zhang, B.T.: Non-parametric bayesian sum-product networks. In: ICML Workshop on Learning Tractable Probabilistic Models. vol. 32 (2014)
17. Liang, Y., Bekker, J., Van den Broeck, G.: Learning the structure of probabilistic sentential decision diagrams. In: Uncertainty in Artificial Intelligence (UAI) (2017)
18. Liu, A., Van den Broeck, G.: Tractable regularization of probabilistic circuits. Advances in Neural Information Processing Systems **34**, 3558–3570 (2021)
19. Lowd, D., Davis, J.: Learning markov network structure with decision trees. In: 2010 IEEE International Conference on Data Mining. pp. 334–343. IEEE (2010)
20. Molina, A., Vergari, A., Di Mauro, N., Natarajan, S., Esposito, F., Kersting, K.: Mixed sum-product networks: A deep architecture for hybrid domains. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32(1) (2018)
21. Peharz, R.: Foundations of sum-product networks for probabilistic modeling. Ph.D. thesis, PhD thesis, Graz University of Technology (2015)

22. Peharz, R., Geiger, B.C., Pernkopf, F.: Greedy part-wise learning of sum-product networks. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part II 13. pp. 612–627. Springer (2013)
23. Peharz, R., Gens, R., Pernkopf, F., Domingos, P.: On the latent variable interpretation in sum-product networks. IEEE Transactions on Pattern Analysis and Machine Intelligence **39**(10), 2030–2044 (2016)
24. Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K., Ghahramani, Z.: Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In: International Conference on Machine Learning. pp. 7563–7574. PMLR (2020)
25. Peharz, R., Tschiatschek, S., Pernkopf, F., Domingos, P.: On theoretical properties of sum-product networks. In: Artificial Intelligence and Statistics. pp. 744–752 (2015)
26. Peharz, R., Vergari, A., Stelzner, K., Molina, A., Shao, X., Trapp, M., Kersting, K., Ghahramani, Z.: Random sum-product networks: A simple and effective approach to probabilistic deep learning. In: Uncertainty in Artificial Intelligence. pp. 334–344. PMLR (2020)
27. Poon, H., Domingos, P.: Sum-product networks: A new deep architecture. In: 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops). pp. 689–690. IEEE (2011)
28. Rahman, T., Gogate, V.: Merging strategies for sum-product networks: From trees to graphs. In: Uncertainty in Artificial Intelligence (UAI) (2016)
29. Rahman, T., Jin, S., Gogate, V.: Look ma, no latent variables: Accurate cutset networks via compilation. In: International Conference on Machine Learning. pp. 5311–5320. PMLR (2019)
30. Rooshenas, A., Lowd, D.: Learning sum-product networks with direct and indirect variable interactions. In: International Conference on Machine Learning. pp. 710–718. PMLR (2014)
31. Sánchez-Cauce, R., París, I., Díez, F.J.: Sum-product networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **44**(7), 3821–3839 (2021)
32. Trapp, M., Peharz, R., Ge, H., Pernkopf, F., Ghahramani, Z.: Bayesian learning of sum-product networks. Advances in Neural Information Processing Systems **32** (2019)
33. Vergari, A., Choi, Y., Liu, A., Teso, S., Van den Broeck, G.: A compositional atlas of tractable circuit operations for probabilistic inference. Advances in Neural Information Processing Systems **34**, 13189–13201 (2021)
34. Vergari, A., Di Mauro, N., Esposito, F.: Simplifying, regularizing and strengthening sum-product network structure learning. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II 15. pp. 343–358. Springer (2015)
35. Vergari, A., Molina, A., Peharz, R., Ghahramani, Z., Kersting, K., Valera, I.: Automatic bayesian density analysis. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33(1), pp. 5207–5215 (2019)
36. Zhao, H., Poupart, P., Gordon, G.J.: A unified approach for learning the parameters of sum-product networks. Advances in Neural Information Processing Systems **29** (2016)

# A Data preprocessing for fairness experiments

In this section, we explain the preprocessing measures that are applied to each dataset for fairness experiments. We would like to note that the specific structure of RAT-SPN requires a different set of preprocessing measures compared to LearnSPN. Being able to work with tensors restricts the type of data RAT-SPN can work with, mainly since tensors put specific restrictions on the parameter space of leaf nodes. As such, in addition to the general preprocessing measures (which are applied in both cases), datasets are also discretized and binarized (using one-hot encoding) for the case of RAT-SPN.

## A.1 Adult dataset

For the case where the base learner is LearnSPN, the instances containing missing values (3620 in total, equal to 7.41 % of records) are removed from the dataset. The attribute *fnlwgt*(final weight) is discarded from the dataset. Race is encoded as a binary attribute *race* = {*white*, *non-white*}. The attribute *age* is also discretized as *age* = {25-60, <25 or >60}.

As for the case of RAT-SPN, in addition to the previous measures, the numerical attributes (*capital gain*, *capital loss*, *hours per week*) are discretized as follows: *capital gain* = {≤ 5000, >5000}, *capital loss* = {≤40, >40}, *hours per week* = {<40, 40-60, >60}. Additionally, categorical attributes are transformed as follows: *workclass* = {*private*, *non-private*}, *education* = {*high*, *low*}, *marital-status* = {*married*, *other*}, *relationship* = {*married*, *other*}, *native-country* = {*US*, *non-US*}. Finally, the resulting dataset is discretized to be compatible with RAT-SPN.

## A.2 German dataset

For the case where the base learner is LearnSPN, we extract gender information from attribute *personal-status-and-sex* (which contains information on marital status and the gender of people), which leads to an additional attribute *sex* (the protected attribute for our fairness experiments).

As for the case of RAT-SPN, additional transformations are as follows: *duration* = {≤6, 7-12, >12} (short, medium, and long-term); *credit-amount* = {≤2000, 2000-5000, >5000} (low, medium, and high income); *age* = {≤25, >25}. Finally, the resulting dataset is discretized to be compatible with RAT-SPN.

## A.3 Bank marketing dataset

For the case where the base learner is LearnSPN, the only preprocessing measure is to extract a binary representation of marital status from attribute *marital* as *marital-bin* = {*married*, *non-married*}. the attribute *marital-bin* is added to the original dataset as an additional attribute, representing the protected group for our fairness experiment.

As for the case of RAT-SPN, additional transformations are as follows: *job* = {*blue-collar*, *management*, *service*, *other*}; *balance* = {≤0, >0}; *day* = {≤15, >15}; *duration* = {≤120, 121-600, >600}; *campaign* = {≤1, 2-5, >5}; *pdays* = {≤30, 31-180, >180}; *previous* = {0, 1-5, >5}; *age* = {25-60, <25 or >60}. Finally, the resulting dataset is discretized to be compatible with RAT-SPN.

### A.4 Dutch census dataset

For this dataset, no particular preprocessing has been done. For LearnSPN, the dataset is utilized in its original form, and for the case of RAT-SPN, the only process is to binarize the dataset to address compatibility issues.

### A.5 Credit card clients dataset

For the case where the base learner is LearnSPN, the only preprocessing is to drop the attribute *id*, as it does not contain any useful information about the task. For the of RAT-SPN, additional transformations are as follows: *age* = {≤35, 36-60, >60}; the amount of the given credit (*limit_ bal*),the amount of the bill statements (*bill_ amt_ 1*,...,*bill_ amt_ 6*), and the amount of the previous payments (*pay_ amt_ 1*,...,*pay_ amt_ 6*) = {≤50000, 50001-200000, >200000} (corresponding to *low*, *medium*, *high* levels); history of the past payments *pay_ 0*, ..., *pay_ 6* = {*pay dully*, *1-3 months*, *>3 months*}.

### A.6 Law school dataset

For the case where the base learner is LearnSPN, the dataset is utilized in its original form. For the case of RAT-SPN, additional transformations are as follows: *decile1b* = {≤5, >5}, *decile3* = {≤5, >5}, *lsat* = {≤37, >37}, *ugpa* = {<3.3, ≥3.3}, *zgpa* = {≤0, >0}, *zfygpa* = {≤0, >0}.