Two Reformulation Approaches to Maximum-A-Posteriori Inference in Sum-Product Networks

Denis Deratani Mauá DENIS.MAUA@USP.BR

Heitor Reis Ribeiro HEITOR.RIBEIRO@USP.BR

Gustavo Perez Katague GUSTAVO.KATAGUE@USP.BR
Institute of Mathematics and Statistics, University of São Paulo, Brazil

Alessandro Antonucci ALESSANDRO@IDSIA.CH
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Switzerland

Abstract

Sum-product networks are expressive efficient probabilistic graphical models that allow for tractable marginal inference. Many tasks however require the computation of maximum-a-posteriori configurations, an NP-hard problem for such models. To date there have been very few proposals for computing maximum-a-posteriori configurations in sum-product networks. This is in sharp difference with other probabilistic frameworks such as Bayesian networks and random Markov fields, where the problem is also NP-hard. In this work we propose two approaches to reformulate maximum-a-posteriori inference as other combinatorial optimization problems with widely available solvers. The first approach casts the problem as a similar inference problem in Bayesian networks, overcoming some limitations of previous similar translations. In addition to making available the toolset of maximum-a-posteriori inference on Bayesian networks to sum-product networks, our reformulation also provides further insight into the connections of these two classes of models. The second approach casts the problem as a mixed-integer linear program, for which there exists very efficient solvers. This allows such inferences to be enriched with integer-linear constraints, increasing the expressivity of the models. We compare our reformulation approaches in a large collection of problems, and against state-of-the-art approaches. The results show that reformulation approaches are competitive.

Keywords: Tractable Probabilistic Models; Sum-Product Networks; Probabilistic Circuits; Maximum-a-Posteriori Inference; Mixed-Integer Linear Programming; Bayesian Networks.

1. Introduction

Sum-Product Networks (SPNs) are praised for their ability to capture complex probabilistic knowledge while enabling tractable computation of marginal probabilities (Poon and Domingos, 2011; Gens and Domingos, 2013). Among other benefits, the ability to efficiently deliver marginal inferences allows for principled handling of missing data, and more accurate parameter and structure learning (Peharz et al., 2016).

Many problems are best approached by finding a configuration of a subset of variables which maximizes its conditional probability given some evidence. Examples include image segmentation, missing data imputation and fault diagnosis, to name a few. This type of computation, known as Maximum-A-Posteriori (MAP) inference, is NP-hard in SPNs (Peharz et al., 2016), even to approximate (Conaty et al., 2017). It remains hard even for selective networks, if some variables are marginalized.

Despite its practical relevance, very few algorithms have been formulated for the task. Poon and Domingos (2011)'s MaxProduct algorithm finds an approximate solution in linear time in the size of the network. The algorithm is best described as replacing sums by maximizations during the network evaluation, and is equivalent to maximizing also over latent variables in Bayesian networks. Conaty et al. (2017) developed a quadratic runtime method, called ArgMaxProduct, that improves over MaxProduct by considering multiple greedy solutions while traversing the network. Mei et al. (2018) developed heuristic search approaches including beam-search, for approximate and fast inference, and branch-and-bound search for exact inference. These methods allow the user to set a compromise between accuracy and time complexity. Nevertheless, MaxProduct remains the method of choice of practitioners; this is so even considering that the method has no guarantees, and is empirically and theoretically outperformed by other simple methods (Conaty et al., 2017).

In this work, we develop two reformulation approaches for MAP inference in Sum-Product Networks. The first approach translates an SPN into a a distribution-equivalent Bayesian Network. The translation differs from previous approaches (Zhao et al., 2015; Peharz et al., 2016; dos Santos et al., 2017) in that it generates small in-degree nodes and dispenses with the need of special data structures for compactly representing probability tables. This allows solving MAP in the translated BNs by any marginal MAP algorithm available (e.g., Liu and Ihler (2011); Marinescu et al. (2018); Mauá and de Campos (2012)), and in particular by message-passing algorithms (Liu and Ihler, 2013). Moreover, as the translation preserves the SPN structure, it allows for cross-fertilization of algorithmic ideas between the two formalisms. The second approach combines ideas from Zhao et al. (2015) and de Campos and Ji (2008) to translate an SPN directly into a mixed-integer linear program (MILP). This is performed by extracting for each variable the conditional distribution induced by the SPN, compactly represented as an Algebraic Decision Diagram (ADD). Similar ADDs are obtained to represent the distributions associated to sum nodes in the network. By performing a symbolic variable elimination with those ADDs, we obtain a MILP that can be solved by any off-the-shelf solver. In addition to benefiting from the high-performance commercial solvers available, the translation increases the expressivity of the inference with integer-linear constraints. For example, one can impose that no more that a certain number of variables can agree on their values with minimal effort. A side contribution of this work is the release of a benchmark of challenging MAP inference problems for SPNs.

2. Background

We start by reviewing some definitions about SPNs and Bayesian networks, and fixing notation.

An SPN is a weighted rooted graph S where each inner node is either a sum node (⊕) or a product node (⊗), and each leaf node (○) is associated with a univariate distribution (Gens and Domingos, 2013). In this work, we consider only SPNs with finite-valued random variables. In this case, we can assume, without loss of generality, that all univariate distributions are indicator functions $I_{X=x}$, that return 1 at $X = x$ and 0 at $X \neq x$.

If i is a node in S, we write S_i to denote the SPN rooted at i. The arcs $i \to j$ from a sum node i to a child j are associated with a weight $\omega_{ij} \geq 0$. The remaining arcs have weight 1. The *scope* of an SPN rooted at i is the set of random variables $X = \{X_1, \ldots, X_n\}$ over which the distributions/indicators at the leaves are defined. We associate every node i of the graph with an index set $\sigma(i)$ such that $X_{\sigma(i)}$ is the scope of $S_{\sigma(i)}$ (if needed we assume an arbitrary ordering over

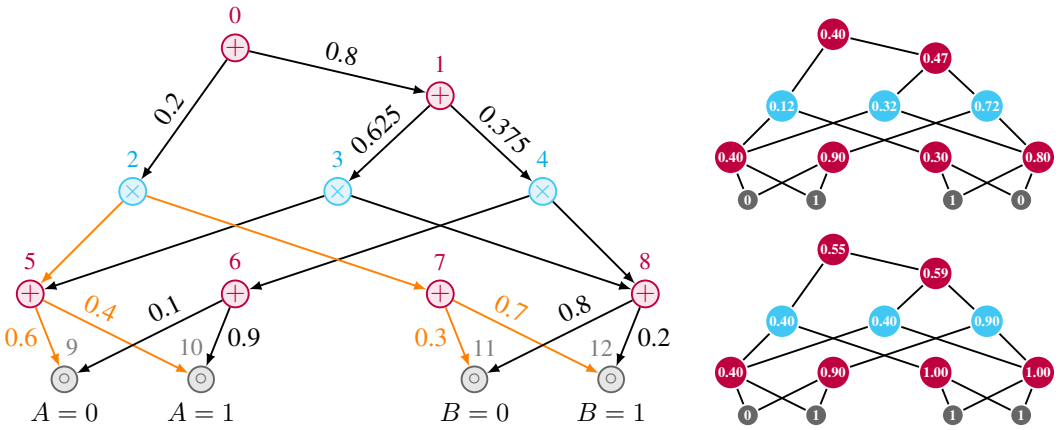

Figure 1: Left: A Sum-Product Network. Top right: The evaluation of the SPN at $A = 1, B = 0$. Bottom right: the computation of the marginal $P(A = 1) = S(1, \star)$.

the variables, and interpret $X_{\sigma(i)}$ as a random vector). The graph in the left side of Figure 1 is an example of an SPN with scope $X = \{A, B\}$. Unit weights are omitted in the figure.

An SPN defines a *computation graph* such that each inner node i computes the function $S_i(x) = \bigcirc_{i \to j} \omega_{ij} S_j(x)$, where x is a an assignment to the scope of i and $\bigcirc$ is either $\sum$ or $\prod$ according to the type of the respective node ($\oplus$ or $\otimes$, resp.). Note that for convenience, we write x to denote the projection $x_{\sigma(j)}$ of assignment x onto the scope of S_j, even if $\sigma_j \subset \sigma_i$. The value of a leaf node is the value returned by the associated indicator function. For example, the values of each node of the SPN in Figure 1(left) for the assignment $A = 1$ and $B = 0$ are shown in the graph on the top right. As shown, the network computes $S(1, 0) = 0.4$ at its root node. Let $X_E = e$ be an assignment to a subset of the scope of SPN S, and define x_e to be the configuration of the variables X in the scope such that $x_i = \star$ for any i not in E, where $\star$ is a special symbol, and x_i assigns the same value as e for i in E. We define the value $S(x_e)$ as the computation obtained by setting any leaf node i whose scope is not in E to one (while the remaining nodes are computed as before). For example, the bottom-right graph in Figure 1 shows the computation of $S(1, \star)$. When it is clear from context, we write $S(e)$ to denote $S(x_e)$.

We impose the following constraints on an SPN S. *Completeness (a.k.a. smoothness):* For any child j of a sum node i, we have that $\sigma(i) = \sigma(j)$. *Decomposability:* For any children j and k of a product node, we have that $\sigma(j) \cap \sigma(k) = \emptyset$. *Normality:* For any sum node i, the weights ω_{ij} add to 1. Those assumptions ensure that the SPN represents a joint probability distribution $P(X = x) = S(x)$, and that any marginal $P(X_E = e) = \sum_{x \sim e} S(x)$ is computed efficiently as $S(e)$. The bottom-right graph of Figure 1 shows the computation of $P(A = 1)$ as $S(1, \star)$. The *MAP inference* problem is to compute $\max_{x_Q} P(X_Q = x_Q, X_E = e) = \max_{x_Q} S(x_Q, e)$ for a given subset of the variables $X_Q \subseteq X \setminus X_E$. Note that we allow for marginalized variables.

A *Bayesian Network* is a tuple (X, G, P) where X is a set of random variables, G is a directed acyclic graph whose nodes form an index set for the variables in X and P is a joint probability distribution that factorizes with respect to G, that is, $P(X = x) = \prod_{i=1}^{n} P(X_i = x_i | X_{\pi(i)} = x_{\pi(i)})$, where $\pi(i)$ denotes the set of parent nodes of node i in G and $P(X_i = x_i | X_{\pi(i)} = x_{\pi(i)}) =$

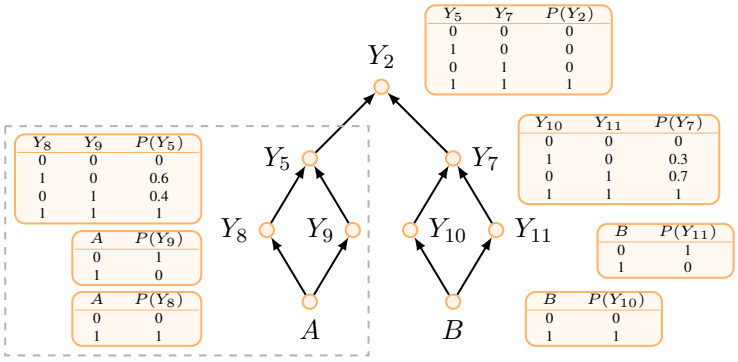

Figure 2: Bayesian network equivalent of the highlighted sub-network in Figure 1.

$P(X_i = x_i)$ if i has no parents. A Bayesian network over categorical variables with few parents can thus be compactly represented by its conditional probability tables (CPTs) $P(X_i|X_{\pi(i)})$. Drawing inferences with a given Bayesian network is usually demanding. For instance, computing marginal inferences is #P-hard, and MAP inference is NP^{PP}-hard (Park and Darwiche, 2004).

3. Sum-Product Network to Bayesian Network

We now describe our first reformulation approach, which obtains a Bayesian network whose distribution, after marginalizing latent variables, equals the distribution induced by the SPN. In a nutshell the transformation consists in reverting the direction of the arcs in the SPN, and reading the resulting computation graph as a Bayesian network over binary variables. The sum and product functions are simulated using special encodings of the corresponding CPTs.

To start with an example, consider the SPN rooted at the sum node 5 in Figure 1. That is, the SPN consists of a sum node linked to two indicator nodes on variable A by weights 0.6 and 0.4. Build a Bayesian network with the structure and parameters inside the box in Figure 2. Note that all variables are binary. Provided that $P(A = a) > 0$, we have that

$$\begin{aligned} P(Y_5 = 1|a) &= \sum_{y_8,y_9} P(Y_5 = 1|y_8, y_9)P(Y_8 = y_8|A = a)P(Y_9 = y_9|A = a) \\ &= (0.6 + 0.4)P(Y_8 = 1|a)P(Y_9 = 1|a)+ \\ &\qquad 0.6P(Y_8 = 1|a)P(Y_9 = 0|a) + 0.4P(Y_8 = 0|a)P(Y_9 = 1|a) \\ &= 0.6I_{A=0}(a) + 0.4I_{A=1}(a) = S_5(a)\,. \end{aligned}$$

Now consider the SPN rooted at the product node 2 in Figure 1 and build the Bayesian network in Figure 2. One can verify that $P(Y_2 = 1|A = a, B = b) = P(Y_5 = 1|a)P(Y_7 = 1|b) = S_2(a, b)$. Hence, the Bayesian network computes the same distribution as the SPN through the marginal of $Y_2 = 1$ for any specification of $P(A) > 0$ and $P(B) > 0$. In particular, if we assign uniform marginals $P(A) = P(B) = 1/2$ to the root variables of the Bayesian network, then it follows that $\max_{a,b} S_2(a, b) = \max_{a,b} P(Y_2 = 1|A = a, B = b) \propto \max_{a,b} P(Y_2 = 1, A = a, B = b)$.

The general algorithm is a straightforward generalization of these ideas. Take an SPN S where each inner node has exactly two children. This can always be enforced by efficiently modifying the structure. First create root nodes $X_1, \ldots, X_n$, one for each variable in the scope of S, and assign

them uniform distributions $P(X_i = x_i) \propto 1$. Then, for each node i in S, create a binary node Y_i in the Bayesian network with parents Y_l and Y_r, where l and r denote the children of i in the SPN. If i is a sum node with weights ω_l and ω_r, specify $P(Y_i = 1|Y_l = 0, Y_r = 0) = 0$, $P(Y_i = 1|Y_l = 1, Y_r = 0) = \omega_l$, $P(Y_i = 1|Y_l = 0, Y_r = 1)$ and $P(Y_i = 1|Y_l = 1, Y_r = 1) = 1$. If i is a product node, then specify $P(Y_i = 1|Y_l = a, Y_r = b) = 1$ if $a = b = 1$ and $P(Y_i = 1|Y_l = a, Y_r = b) = 0$ otherwise. Last, if i is an indicator at value x_j of variable X_j then specify $P(Y_i = 1|X_j) = I_{x_j}(X_j)$. We have:

Theorem 1 *Let P be the probability distribution induced by the Bayesian network obtained by the above transformation given an SPN S rooted at node 0. For any evidence $X_E = e$ on the scope of S, it follows that $k \cdot P(Y_0 = 1|X_e = e) = S(e)$, where k is the product of the cardinalities of the variables not in E. Moreover, we can compute $P(Y_0 = 1|X_e = e)$ in linear time in the size of S (i.e., in the number of nodes and arcs).*

Proof We show that $P(Y_0 = 1|e) = P(X_e = e)$ by induction in the size/depth of the Bayesian network. The base case is immediate as $P(Y_0 = 1|x) = I_{X=x}(x)$ by construction. Suppose that the root of S is a sum node. Call by Y_1 and Y_2 the parents of Y_0 in the corresponding Bayesian network. Note that Y_1 and Y_2 are d-separated by $X = x$ (hence conditionally independent), and that Y_0 and X are d-separated by Y_1, Y_2. Thus

$$\begin{aligned} P(Y_0 = 1|e) &= \sum_{x \sim e} \sum_{y_1, y_2} P(Y_0 = 1|Y_1 = y_1, Y_2 = y_2) P(Y_1 = y_1|x) P(Y_2 = y_2|x) P(x|e) \\ &= P(x \sim e) \sum_{x \sim e} (\omega_{01} P(Y_1 = 1|x) + \omega_{02} P(Y_2 = 1|x)) = P(x \sim e) S(e)\,, \end{aligned}$$

where $P(x \sim e)$ is the reciprocal of the product of the cardinalities of the variables not in E, and in the last equality we used the inductive hypothesis on the Bayesian networks rooted at Y_1, Y_2. If instead 0 is a product node then $P(Y_0 = 1|e) = \sum_{x \sim e} \sum_{y_1, y_2} P(Y_0 = 1|y_1, y_2) P(y_1, y_2|x) P(x) = P(x \sim e) S(e)$. To compute $P(Y_0 = 1|e)$ in linear time, iteratively compute $P(Y_i = 1|e)$ using a topological ordering, effectively simulating the computations performed by S. ■

A corollary of the previous result is that MAP inference in the SPN S reduces to MAP inference in the corresponding Bayesian network by

$$\max_{x_Q} S(x_Q, e) \propto \max_{x_Q} P(Y_0 = 1|x_Q, e) \propto \max_{x_Q} P(Y_0 = 1, x_Q, e)\,,$$

where P is the probability distribution induced by the Bayesian network whose single leaf is Y_0.

4. Sum-Product Network to Mixed-Integer Linear Program

We now present a technique to build a MILP program that solves MAP inference in a given SPN.

Zhao et al. (2015) showed how SPNs can be translated into two-layer bipartite Bayesian networks, where the root nodes are latent variables representing the sum nodes in the SPN, and the leaves are manifest variables representing the variables in the scope of the SPN. To avoid an exponential blow up in space, the CPTs in the Bayesian network need to be represented as Algebraic Decision Diagrams (Bahar et al., 1997), which allow for context-specific independences to be exploited. Roughly speaking, an Algebraic Decision Diagram (ADD) is a canonical directed acyclic

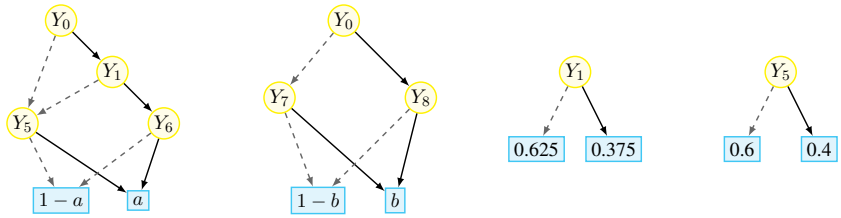

Figure 3: Parametrized Algebraic Decision Diagrams extracted by restricting the SPN in Figure 1 to A, B and Y_1, Y_5, respectively.

graph representation of real-valued functions of binary variables. The inner nodes of the diagram represent variables, the outgoing edges represent assignments of values, and the leaves store the image of the function. Notably, ADDs (as well as the generalization we use) allow for multiplication of functions and marginalization in polynomial time in the size of the input diagrams. The rightmost graph in Figure 3 is an ADD representation of the function $f(y_5) = 0.4(1 - y_5) + 0.6y_5$.

de Campos and Ji (2008) showed how to reformulate sequential decision making problems (which can be reduced to/from MAP inference (Mauá, 2016)) as MILP problems using a symbolic variable elimination procedure. We combine ideas from both works to show how MAP inference in SPNs can be reformulated as MILP problems. We formalize the symbolic variable elimination procedure using Parametrized Algebraic Decision Diagrams (PADD) (Delgado et al., 2011); these models extend Algebraic Decision Diagrams to allow for multilinear expression at the leaves. Figure 3 shows examples of PADDs.

Our proposed translation works as follows. Take an SPN S where each sum node has two children. Associate with every sum node i a (fresh) binary random variable Y_i (note the difference: here only sum nodes have associated latent variables). Interpret $Y_i = 0$ (resp., $Y_i = 1$) as selecting the left (resp., right) child of node i. Let Y denote the set of all such variables. As in the work of Zhao et al. (2015), construct an ADD for each Y_i in Y that represents the function $f(y_i) = (1-y_i)\omega_l + y_i\omega_r$, where ω_l and ω_r are the weights of the outgoing edges of i in S. For each variable X_i in the scope of S, obtain a PADD as follows (this differs from Zhao et al. (2015)'s construction). First, construct the restriction $S_{|X_i}$ of S to X_i by removing any node whose scope does not contain X_i. Then replace sum nodes with the corresponding variables Y_i. Finally, replace each indicator leaf node $I_{x_i}(X_i)$ with the (linear) expression x_i. The result is a PADD over the variables Y whose leafs are linear functions of optimization variables x_i. Figure 3 shows the PADDs extracted by restricting the SPN in Figure 1 to variables A and B, and to variables Y_1 and Y_5 representing sum nodes 1 and 5, respectively. The dashed edges denote assignments of value 0, and the solid edges denote assignments of value 1. For convenience, we use a (resp., b) as the variable for the indicator $A = 1$ (resp., $B = 1$), and $1 - a$ (resp., $1 - b$) for the indictor $A = 0$ (resp., $B = 0$).

The domain graph of a collection of functions (or their representations as ADDs) is the graph whose nodes are the variables in the domain and two variables are connected iff they co-occur in some function. A path decomposition of an undirected graph $G = (V, E)$ is a collection of node subsets $Z_1, \ldots, Z_m$ (called clusters or paths) such that the endpoints of any edge in E co-occur in some Z_i, and for any Z_i, Z_j, Z_k, $Z_i \cap Z_k \subseteq Z_j$ (the running intersection property). While obtaining a path-decomposition of small size (given by the size of the largest Z_i) is NP-hard, there are efficient heuristics (Bodlaender et al., 1995; Catell et al., 1996).

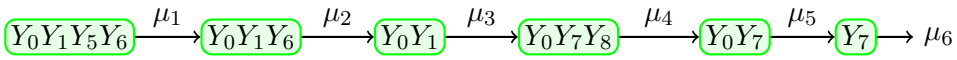

Figure 4: Path decomposition for the variable elimination procedure described in the text.

The MILP is generated by processing the generated PADDs using a variable elimination procedure that operates according to a path decomposition of the domain graph of the PADDs. The use of a path decomposition ensures that at most one PADD obtained by restricting a variable X_i is combined at each step, which ensures that the generated constraints can be cast into mixed integer-linear form. To this end, associate to each cluster of the path decomposition a collection of PADDs whose scope is contained in that cluster (with at most one PADD generated by a restriction of a variable X_i per cluster). Assume that clusters are numbered from left to right in the path decomposition. Let Φ_i denote the collection of PADDs associated with cluster i, and $Z_i \subseteq Y$ denote the variables in that cluster. We obtain a new PADD μ_i by multiplying all PADDs in Φ_i and marginalizing the variables $Z_{i+1} \setminus Z_i$:

$$\mu_i(Z_i \cap Z_{i+1}) = \sum_{Z_i \setminus Z_{i+1}} \prod_{\phi \in \Phi_i} \phi \,. \tag{1}$$

Create a new PADD $\tilde{\mu}_i$ by replacing every leaf of μ_i by a new fresh variable u_k. Insert $\tilde{\mu}_i$ in the collection Φ_{i+1} and repeat the operation. When the rightmost cluster m is reached, generate a PADD μ_m by eliminating all variables Z_m. The MILP is obtained as the constraints

$$\tilde{\mu}_i(y) = \mu_i(y) \quad \text{for each leaf } y \text{ in } \mu_i \,,\; i = 1, \ldots, m, \tag{2}$$

where $\tilde{\mu}_i(y)$ (resp., $\mu_i(y)$) is the expression at leaf y. Recall that a leaf in an PADD corresponds to the set of configurations of its scope which map to the same value. In particular the PADD μ_m contains a single node (as all variables have been marginalized) with an expression that is used as the objective of the program.

A concrete example helps in understanding the procedure. Consider the path decomposition in Figure 4, where cliques are numbered from left to right. The algorithm first computes the PADD μ_1 below by combining the PADDs in the the first clique and eliminating Y_5, for example:

$$\mu_1(Y_0, Y_1, Y_6) = \quad = \sum_{Y_5}$$

A new PADD $\tilde{\mu}_1(Y_0, Y_1, Y_6)$ is then produced by duplicating the structure of $\mu_1(Y_0, Y_1, Y_6)$ and associating fresh variables to the leaves:

$$\mu_2(Y_0, Y_1) = \quad = \sum_{Y_6}$$

The process continues until the PADD μ_6 is produced at the rightmost clique:

$$\mu_6() = \boxed{0.3u_{11} + 0.7u_{12}} = \sum_{Y_7}$$

By equating the leaves of μ_i and $\tilde{\mu}_i$, we obtain the corresponding bilinear program:

$$\text{Maximize} \quad 0.3u_{11} + 0.7u_{12}\,, \quad \text{subject to}$$

$$\begin{aligned}
&u_1 = 0.6 - 0.2a\,, && u_2 = 1 - a\,, && u_3 = a\,,\\
&u_4 = u_1\,, && u_5 = 0.1u_2 + 0.9u_3\,, && u_6 = u_4\,,\\
&u_7 = 0.625u_4 + 0.375u_5\,, && u_8 = u_6 - bu_6\,, && u_9 = bu_6\,,\\
&u_{10} = 0.8u_7 - 0.6bu_7\,, && u_{11} = 0.2u_8 + 0.8u_{10}\,, && u_{12} = 0.2u_9 + 0.8u_{10}\,,\\
&a, b \in \{0, 1\}\,, && u_1, \ldots, u_{12} \in [0, 1]\,.
\end{aligned}$$

The above program is not yet a MILP program because of the products of binary-valued variables and linear variables such as in the equality $u_8 = bu_6$. The fact that non-linear terms involve a binary variable and a linear variable is not coincidental. Recall that products appear due to the multiplication/combination of PADDs when computing μ_i. And because the variable elimination procedure uses a path decomposition to eliminate variables, at most one such message is multiplied when combining PADDs. Also, we can always create a path decomposition that allows us to associate at most one PADD containing a binary variable per clique. Hence we are sure to only generate at most bilinear constraints where products contain one binary variable and one linear variable. These bilinear products can be converted into a set of linear constraints by the well-known McCormick relaxation.[1] We therefore have the following result.

Theorem 2 *The MILP program is equivalent to the MAP inference problem in the SPN: The optimum object value equals the MAP inference value, and the configuration of the integer variables encode the configuration of a MAP configuration.*

Proof Zhao et al. (2015) showed that the SPN is distribution-equivalent to a Bayesian network with root nodes Y_i and leaves X_i, whose CPTs are specified by the ADDs as we obtain, except that the linear expressions at the leaves are represented as nodes in the ADD (so that it is also a function of X_i). Thus, the symbolic variable elimination performed with the ADDs simulates the computations of variable elimination with that Bayesian network. ■

5. Experiments

We performed experiments with SPNs learned from a selected collection of datasets.[2] All variables in these datasets are discrete or have been discretized. We learn (tree-shaped) SPNs with our own implementation of the LearnSPN algorithm (Gens and Domingos, 2013), selecting hyperparameters by grid search. We then modified the networks so that each node has at most two parents. Table 1 contains the characteristics of the datasets and the SPNs we use, sorted by the number of indicators (which are related to the size of the space of configuration).

For each SPN/dataset we generate MAP inference tasks as follows. First we randomly partition the variables into equally sized sets of optimized, marginalized and fixed (the partition is constant for each SPN). Then we select values for the fixed/evidence variables using the test set such

1. E.g., the bilinear term $b \cdot u_6$ can be encoded by $u' \leq b$, $u' \leq u_6$, $u_6 - 1 + b \leq u'$, where u' is a fresh linear variable.
2. The networks and datasets are available at `https://gitlab.com/pgm-usp/learned-spns.git`

SPN	Var.	Nodes	⊕	⊗	◎	Local Search	ArgMaxProduct	Belief Propagation	MaxSearch
Nltcs	16	3939	2315	1592	32	1.86±1.79	2.35±1.56	2.35±1.56	2.35±1.56
Mushrooms	112	8489	4587	3678	224	1.06±0.17	1.44±0.30	1.25±0.33	1.44±0.29
Molecular	58	10207	7515	2462	230	1.02±0.05	3.01±2.88	2.59±3.01	1.00±0.00
DNA	180	33465	17953	15152	360	1.00±0.01	1510.9±1247.7	1055.1±1218.3	313.2±363.9
US Census	68	162118	140722	20752	644	1.07±0.18	1.81±0.63	1.72±0.63	1.81±0.63
NIPS	500	13563	6876	5687	1000	1.06±0.01	27325.3±33348.6	20981.6±26383.0	1.00±0.00
Optdigits	65	23008	20739	1171	1098	6.44±20.76	80.19±210.30	75.39±188.97	1.02±0.13

Table 1: Relative performance of different methods compared to MaxProduct (see text).

that MaxProduct is suboptimal. This procedure generated from 10 to 30 *interesting* instances per SPN/dataset (the maximum of 30 instances was imposed for computational convenience). We verify sub-optimality by obtaining a higher value solution with ArgMaxProduct. Thus, our results here approximate a worst-case performance for MaxProduct rather than expected performance. They also slightly favor MaxProduct (as we might discard instances where ArgMaxProduct does not find an improving solution and some other competing method does.)

We compare the reformulation approaches against MaxProduct (MP) (Poon and Domingos, 2011), MaxProduct followed by a one-neighborhood local search in the space of configurations (LS), ArgMaxProduct followed by local search (AMP) (Conaty et al., 2017), and MaxSearch (MS) (Mei et al., 2018).[3] We initialize MS with the solution found my MP, and use the forward checking heuristic. With the exception of MS, which performs branch-and-bound search, all other methods are approximate and run reasonably fast. We limit the runtime of MS to 1 hour.

Regarding the Bayesian Network reformulation, we tested with three different approximate MAP solvers: the Weighted Mini-Bucket elimination algorithm (WMB) (Liu and Ihler, 2011) implemented in the Merlin library, the Anytime Best-First AND/OR Search (Marinescu et al., 2018) and the Hybrid-Product Belief Propagation (HBP) (Liu and Ihler, 2013). WMB produces approximate solutions whose accuracy depends on the given i-bound, which limits the sizes of the tables generated and the number of iterations of message-passing. With small i-bounds (< 10) WMB did not produce competitive solutions (generally worse than LS), and with larger i-bounds it did not finish within 1 hour. AND/OR search also could not finish within the time limit for almost all networks. This can be explained by it using a constrained elimination order, which is prohibitively costly for the high-treewidth Bayesian networks we generate. HBP performs message passing over the SPN structure, and therefore takes linear time in the size of the network. We use the unweighted version of the algorithm, and perform two different initialization strategies: the standard uniform initialization, and biasing the messages (of the root nodes) according to the configuration of MaxProduct. The latter strategy often produce higher quality solutions. As with the other methods, we perform a local search from the found solution. We observed that in some instances while the result of HBP with uniform initialization did not improve over the informed initialization, the subsequent local search obtained higher values. As HBP is cheap, we run both variants for 10 iterations and consider the best solution (over all iterations and initialization strategies) for each instance. We also experimented with random initializations, but did not observe any improvements over the other strategies. We left for the future the possibility of using the convexified form of HBP to producing upper bounds (that could be used by e.g. MaxSearch).

3. Our implementations are available at `https://gitlab.com/pgm-usp/pyspn`.

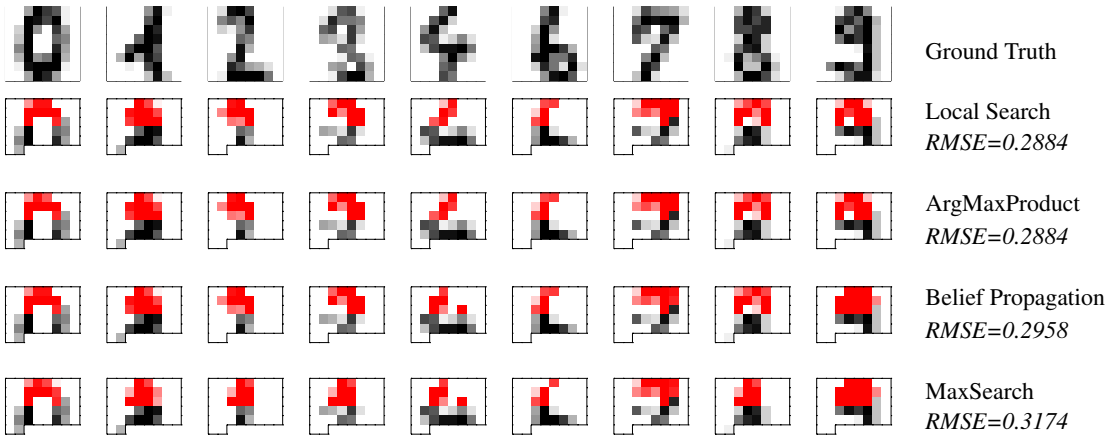

Figure 5: Examples of MAP inference for the Optdigits dataset (see text).

For the MILP reformulation, we obtain a path decomposition by eliminating variables according to the min-degree heuristic. Future work should evaluate different heuristics for obtaining path decompositions. We use Gurobi to optimize the corresponding MILP program. We multiply the objective function and the constraints by some large constant (e.g., 10^6), to improve numerical stability. For the two smaller networks (Nltcs and Mushrooms), the translation to a MILP program was very fast, and the optimization finished finished within a couple of minutes with the (guaranteed) optimal solution. The translation was also very fast for the Molecular network, but Gurobi failed at finding an optimal solution due to numerical issues. The MAP values for that network ranged from 10^{-23} to 10^{-14}, below the minimum tolerance value allowed by Gurobi (10^{-6}) for verifying constraint feasibility and integrality. For the larger networks, the translation took more than 1 hour, at which point it was interrupted.

The results for the different approaches appear in Table 1. We omit the results of MILP, as they are uninformative (either optimal or timed-out). The numbers show the mean and standard deviation of the ratio between the value of a given solution and the value of the corresponding solution obtained by MP. As previously observed (Conaty et al., 2017), AMP obtains excellent performance in terms of accuracy. It found the best solution in all tasks, which were optimal for the smallest networks Nltcs and Mushrooms. Note that as the size of the MAP problem increases, the gains over MP become very pronounced, reaching 2 orders of magnitude in Optdigits, 3 orders of magnitude for DNA and 4 orders of magnitude for NIPS. At the same time, the performance of MaxSearch degrades quickly. For the largest networks (NIPS and Optdigits), it was outperformed by local search (given the same initial solution). For DNA, it obtained on average solutions of much lower value than HBP and AMP. HBP is very competitive, also finding solutions which are often orders of magnitude better than MP and LS, while still displaying linear time complexity. We observed that HBP occasionally finds better solutions than AMP. Thus, when time complexity is not crucial, running both approaches can improve over the performance of either.

To observe the effect of improved MAP values in an end task, we analyze the solutions of the different algorithms for the Optdigits dataset. This dataset contains 8x8 digitalized images of handwritten digits, where each variable represents the intensity of a pixel (in a 0 to 16 scale). We build MAP problems by predicting the top part of each image while fixing the configuration for some

middle part and marginalizing the remaining. Some selected predictions are shown in Figure 5. Red pixels are solutions to the MAP inference, gray pixels are evidence, missing fragments represent marginalized variables. While the predictions look very similar, they differ in important aspects. For example, MaxProduct and MS were not able to predict the "hole" in digits nine. They also did not preduct a hole in digit eight, that while missing from the ground truth image is to be expected. We also used the ground truth to compute the Relative Mean Square Error (RMSE) for each algorithm. In sum, higher MAP values reflected in smaller errors, although the discrepancy in errors is not as large as the discrepancy in MAP values (but note that the images contain a lot of white pixels).

6. Conclusions

Sum-Product Networks (SPNs) are often justified for their ability to deliver complex probabilistic queries in polynomial time. However, several successful examples of applications of SPNs require solving a (marginal) maximum-a-posteriori inference (MAP), that is, maximizing over part of the variables given the values of some others. This is the case for example, when performing data imputation with SPNs, completing images, building multidimensional classifiers, and so on. Solving such a task is however NP-hard.

In this work we showed how to cast MAP in SPNs as equivalent problems in two other formalisms. Our first contribution was to develop a new translation of SPNs into Bayesian networks; unlike previous translations (Zhao et al., 2015; Peharz et al., 2016; dos Santos et al., 2017), the new translation does not require special data structures. This allows us to leverage the large availability of (marginal) MAP algorithms for Bayesian networks. It also brings further insights into how to adapt the extensive work on message-passing and variational inference to SPNs. We hope this connection foster new research in that direction. Our second contribution was a reformulation of the MAP problem as a mixed-integer linear program. This opens up the vast work on mathematical programming, as well as access to the very efficient (commercial) solvers available.

Experiments with a varied collection of learned SPNs showed that the Bayesian Network reformulation approach offers a competitive alternative to current algorithms, extending the toolset of MAP for SPNs.

Acknowledgments

This work was partly supported by the CNPq grant 304012/2019-0 and CAPES finance code 001.

References

I. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2-3):171–206, 1997.

H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

K. Catell, M. J. Dinneen, and M. R. Fellows. A simple linear-time algorithm for finding path-decompositions of small width. *Information Processing Letters*, 57:197–204, 1996.

D. Conaty, D. D. Mauá, and C. P. de Campos. Approximation complexity of maximum a posteriori inference in sum-product networks. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence*, pages 322–331, 2017.

C. de Campos and Q. Ji. Strategy selection in influence diagrams using imprecise probabilities. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pages 121–128, 2008.

K. V. Delgado, S. Sanner, and L. N. De Barros. Efficient solutions to factored MDPs with imprecise transition probabilities. *Artificial Intelligence*, 175(9-10):1498–1527, 2011.

A. E. dos Santos, C. J. Butz, and J. S. Oliveira. On converting sum-product networks into Bayesian networks. In *Proceedings of the Canadian Conference on Artificial Intelligence*, pages 329–334, 2017.

R. Gens and P. Domingos. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 873–880, 2013.

Q. Liu and A. Ihler. Bounding the partition function using Hölder's inequality. In *Proceedings of the 28th International Conference on Machine Learning*, pages 849–856, 2011.

Q. Liu and A. T. Ihler. Variational algorithms for marginal MAP. *Journal of Machine Learning Research*, 14:3165–3200, 2013.

R. Marinescu, J. Lee, R. Dechter, and A. Ihler. And/or search for marginal MAP. *Journal of Artificial Intelligence Research*, 63:875–921, 2018.

D. D. Mauá. Equivalences between maximum a posteriori inference in bayesian networks and maximum expected utility computation in influence diagrams. *Int. J. Approximate Reasoning*, 68:211–229, 2016.

D. D. Mauá and C. P. de Campos. Anytime marginal MAP inference. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1471–1478, 2012.

J. Mei, Y. Jiang, and K. Tu. Maximum a posteriori inference in sum-product networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

J. Park and A. Darwiche. Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research*, 21:101–133, 2004.

R. Peharz, R. Gens, F. Pernkopf, and P. Domingos. On the latent variable interpretation in sum-product networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–14, 2016.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pages 337–346, 2011.

H. Zhao, M. Melibari, and P. Poupart. On the relationship between sum-product networks and Bayesian networks. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 116–124, 2015.