Machine Learning
https://doi.org/10.1007/s10994-019-05813-2

Sum–product graphical models

Mattia Desana[1] · Christoph Schnörr[2]

Received: 19 September 2017 / Revised: 14 January 2019 / Accepted: 21 May 2019

Abstract
This paper introduces a probabilistic architecture called sum–product graphical model (SPGM). SPGMs represent a class of probability distributions that combines, for the first time, the semantics of probabilistic graphical models (GMs) with the evaluation efficiency of sum–product networks (SPNs): Like SPNs, SPGMs always enable tractable inference using a class of models that incorporate context specific independence. Like GMs, SPGMs provide a high-level model interpretation in terms of conditional independence assumptions and corresponding factorizations. Thus, this approach provides new connections between the fields of SPNs and GMs, and enables a high-level interpretation of the family of distributions encoded by SPNs. We provide two applications of SPGMs in density estimation with empirical results close to or surpassing state-of-the-art models. The theoretical and practical results demonstrate that jointly exploiting properties of SPNs and GMs is an interesting direction of future research.

Keywords Sum product networks · Probabilistic graphical models · Density estimation · Deep learning · Exact inference · Density estimation

1 Introduction

The compromise between model expressiveness and tractability of model evaluation (inference) is a key issue of scientific computing. Regarding probabilistic *graphical models (GMs)*, tractable inference is guaranteed for acyclic graphical models and GMs on cyclic graphs with small treewidth (Wainwright and Jordan 2008). On the other hand, inference with cyclic

Editor: James Cussens.

✉ Mattia Desana
mattia.desana@iwr.uni-heidelberg.de

Christoph Schnörr
schnoerr@math.uni-heidelberg.de

1 Heidelberg Collaboratory for Image Processing (HCI), Heidelberg University, Heidelberg, Germany

2 Image and Pattern Analysis Group (IPA), Heidelberg University, Heidelberg, Germany

graphical models[1] generally suffers from a complexity that exponentially grows with the treewidth of the underlying graph, so that approximate inference is the only viable choice.

Recently, *sum–product networks (SPNs)* (Poon and Domingos 2011) and closely related architectures including Arithmetic Circuits and And–Or Graphs (Darwiche 2002; Dechter and Mateescu 2007) have received attention in the probabilistic machine learning community, mainly due to these attractive properties:

1. The cost of marginal inference with these architectures is *linear* in the model size, hence it is always tractable. This aspect greatly simplifies approaches to learning the structure of such models, the complexity of which essentially depends on the complexity of inference as a subroutine.[2]
2. These architectures allow one to cope with probability distributions that are more complex than tractable graphical models. The key reason is that SPNs efficiently represent *contextual independence*: independence between variables that only holds in connection with some assignment of a subset of variables in the model, called "context". In contrast, the connections between nodes in a GM represent *conditional* independence, a special case of contextual independence where independence holds for all assignments of the context variables. Exploiting contextual independence allows one to drastically reduce the cost of inference whenever the modelled distribution justifies this assumption, and as a result, a subset of distributions that would be represented by graphical models with high treewidth can be represented in a tractable way (see Boutilier et al. 1996). A detailed example illustrating this key point is provided in Sect. 1.1.
3. Several recent papers showed that structure learning approaches for SPNs produce state-of-the-art results in density estimation (see e.g. Gens and Domingos 2013; Rooshenas and Lowd 2014; Rahman and Gogate 2016a, b): remarkably, performing exact inference with tractable models produced in these cases better results than approximate inference with intractable models.

However, there is a price to pay for these favorable properties: the ability of SPNs to represent efficiently contextual independence is due to a *low-level* representation of the underlying distribution. This representation comprises a Directed Acyclic Graph with sums and products as internal nodes, and with indicator variables associated with each state of each variable in the model, that become active when a variable is in a certain state (Fig. 1b). Thus, SPN graphs directly represent the flow of operations performed during the inference procedure, which is much harder to read and interpret than a factorized graphical model which expresses conditional independence. In particular, the factorization associated with the graphical model is lost after translating the model into an equivalent SPN, and can only be retrieved (when possible) through a complex hidden variable restoration procedure (Peharz et al. 2016). As a consequence of these incompatibilities, research on SPNs has largely evolved without much overlap with work on GMs.

In this paper we combine some favorable properties of GMs and SPNs in a single model, which we call *sum–product graphical model (SPGM)*. SPGMs are a hybrid between the two families of models, and explicitly inherit components from both architectures. This approach connects to previous papers that either endow graphical models with the ability to model contextual dependences, or provide methods to infer a high level representation of given SPNs. However, we start from the perspective of defining a new architecture rather than expanding on either GMs or SPNs—we will show that the new representation provides a

[1] With the exception of a subset of discrete graphical models (see e.g. Kolmogorov and Zabih 2004) where inference can be reformulated as a maximum flow problem.

[2] MAP inference is however a NP-hard problem (Conaty et al. 2017).

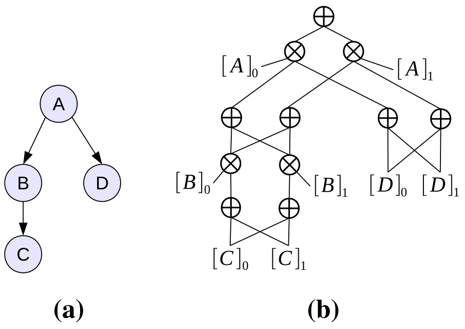

Fig. 1 Representation properties of graphical models (GMs) and sum–product networks (SPNs). The same distribution specified by Eq. (1.1) is represented by a GM in panel **a** and by a SPN in panel **b**. This illustrates that GMs represent conditional independence more compactly than SPNs

more compact and understandable representation than SPNs, close to GMs, while on the other hand maintaining the full efficiency of SPNs. Comparison with related models is discussed in detail in Sect. 1.4 and summarized in Table 1.

Ultimately, the goal of this paper is to provide new connections between GMs and SPNs, and enable researchers to exploit methods from the two families in synergy. In order to show a few preliminary ways in which this can be done in practice, we discuss two example applications in which the new properties lead to state-of-the-art results. Firstly, we devise an algorithm for learning both the structure and the model parameters of SPGMs which is inspired jointly by methods in the GM and SPN families. A comparative empirical evaluation demonstrates competitive performance of our approach in density estimation: we obtain results comparable to state-of-the-art models (which are based on SPNs), despite using a radically different approach from the established body of work and comparing against methods that rely on years of research. Secondly, we apply SPGMs to the challenging task of approximating the density of a known but intractable graphical model, which is an area where SPNs were not previously applied. Also in this case SPGMs obtain strong results in comparison to other recent methods, demonstrating the potential of the new approach.

1.1 Tradeoff between high-level representation and efficient inference: an example

We consider a distribution of discrete random variables A, B, C, D in the following form (shown in Fig. 1a as a directed Graphical Model (GM)):

$$P(A, B, C, D) = P(A)P(B|A)P(C|B)P(D|A) \tag{1.1}$$

Uppercase letters A, B, C, D denote random variables and corresponding lowercase letters a, b, c, d values in their domains $\Delta(A)$, $\Delta(B)$, $\Delta(C)$, $\Delta(D)$. We write $\sum_{a,b,c,d}$ for the sum over the joint domain $\Delta(A) \times \Delta(B) \times \Delta(C) \times \Delta(D)$. Using this notation, the distribution $P(A, B, C, D)$ can be written as a *network polynomial* (Darwiche 2003):

$$P(A, B, C, D) = \sum_{a,b,c,d} P(a, b, c, d)[A]_a[B]_b[C]_c[D]_d \tag{1.2}$$

Here $P(a, b, c, d)$ denotes the *value* of P for assignment $A = a$, $B = b$, $C = c$, $D = d$, and $[A]_a$,$[B]_b$,$[C]_c$, $[D]_d \in \{0, 1\}$ denote *indicator variables*. To compute the partition function all indicator variables of (1.2) are set to 1, and to compute the marginal probability $P(A = 1)$ one sets $[A]_1 = 1$, $[A]_0 = 0$ and all the remaining indicators to 1.

Table 1 Comparison of architectural properties discussed in Sect. 1.4 for SPGMs and related architectures

Model	TractInf	AsSPN	ExpFam	CondInd	FactProd
SPGM	✓	✓	✓	✓	✓
Graphical models			✓	✓	✓
Mixtures of trees (Meila and Jordan 2000)	✓		✓	✓	✓
Hierarchical MT (Jordan 1994)	✓		✓	✓	✓
Mix. Markov model (Fridman 2003)			✓	✓	✓
Gates (Minka and Winn 2009)		✓	✓	✓	✓
Boutilier et al. (1996)			✓	✓	✓
Case-factor diagrams (Mcallester et al. 2004)	✓	✓			✓
BayesNets local Struct (Chickering et al. 2013)	✓	✓			✓
Learn. Efficient MarkovNets (Gogate et al. 2010)	✓	✓			✓
Poole and Zhang (2011)	✓	✓			✓
Value elimination (Bacchus et al. 2012)	✓	✓			✓
SPN as Bayesian nets (Zhao et al. 2015)	✓	✓			✓
And/Or graphs (Dechter and Mateescu 2007)	✓	✓	✓		
SPN (Poon and Domingos 2011)	✓	✓	✓		
Arithmetic circuits (Darwiche 2002)	✓	✓	✓		
CNets (Rahman et al. 2014)	✓		✓	✓	

We consider the following properties: guaranteed tractable inference (TractInf); same inference efficiency as SPNs (AsSPN)—note that models with tractable inference, such as Mixtures of Trees, can have less inference efficiency than SPNs; using exponential family factors with no limitation on generality (ExpFam); high level representation of conditional independence (CondInd); representation as a product of factors as in graphical models (FactProd). Regarding the last two properties, it is important to remark that since SPGMs can represent distributions corresponding to mixtures of tree GMs with shared parts, not all conditional independence relationships represented by general GMs can be represented by SPGMs, but only those that fall in this restricted family

We can further exploit factorization and rearrange the sum of (1.2) in terms of messages μ, as follows:

$$P(A,B,C,D)=\sum_{a\in\Delta(A)}P(a)[A]_a\mu_{b,a}(a)\mu_{d,a}(a),\quad \mu_{b,a}(A)=\sum_{b\in\Delta(B)}P(b|A)[B]_b\mu_{c,b}(b), \tag{1.3a}$$

$$\mu_{c,b}(B)=\sum_{c\in\Delta(C)}P(c|B)[C]_c,\qquad \mu_{d,a}(A)=\sum_{d\in\Delta(D)}P(d|A)[D]_d. \tag{1.3b}$$

Hence, the distribution can be represented in two forms: The first one is the directed graphical model of Eq. (1.1), shown in Fig. 1a. The second one is a sum–product network (SPN) shown by Fig. 1b, which directly represents the computations expressed by Eq. (1.3), with the coefficients $P(\cdot|\cdot)$ omitted in Fig. 1b for better visibility. It is evident that the SPN does not clearly display the high level semantics due to conditional independence of the graphical model. On the other hand, the SPN makes explicit the computational structure for efficient inference and encodes more compactly than GMs a class of relevant situations described next.

Introducing Contextual Independence We consider a distribution in the form

$$P(A,B,C,D,E,Z)=P(Z)P(A)P(B,C,D|A,Z)P(E|B,C,D) \tag{1.4a}$$

with

$$P(B,C,D|A,Z)=\begin{cases}P(Z=0)P(B|A)P(C|A)P(D|A) & \text{if } Z=0,\\ P(Z=1)P(B|A)P(C|B)P(D|C) & \text{if } Z=1.\end{cases} \tag{1.4b}$$

Notice that different independence relations hold depending on the value taken by Z: if $Z=0$, then B, C and D are conditionally independent given A, whereas if $Z=1$, then they form a chain. We therefore say that this distribution exhibits *context specific independence* with *context variable* Z.

As in the example before, this distribution can be represented in different ways. Representing it as a GM (Fig. 2a) requires modelling $P(B,C,D|A,Z)$ as a single factor over 5 variables, since GMs cannot directly represent the *if* condition of (1.4b).[3]

We can also represent the distribution as a *mixture* of two tree-structured GMs (Fig. 2b),

$$P(B,C,D,A,Z)=[Z]_0P(Z=0)P(A)P(B|A)P(C|A)P(D|A)P(E|D) \tag{1.5a}$$

$$+[Z]_1P(Z=1)P(A)P(B|A)P(C|B)P(D|C)P(E|D). \tag{1.5b}$$

However, since a subsection of the two trees corresponding to factors $P(A)$ and $P(E|B)$ appears identically in *both* mixture components, computations for inference are unnecessarily replicated and representation generally loses compactness.

Finally, we may represent the distribution as SPN (Fig. 2c) following the procedure outlined in the previous example. This representation allows one to both express contextual independence and to share common parts in the two models' components, thereby decreasing the number of computations required for inference. On the other hand, as in the previous example, the SPN representation is considerably more convoluted and the probabilistic relations which are easily readable in the other models are hidden.

[3] A workaround involves factors with a complex structure, similar to SPNs, as done for instance in Mcallester et al. (2004). Although this approach would be simple enough in the present example, it generally leads to a representation with the disadvantages of SPNs. See Sect. 1.4 for further discussion.

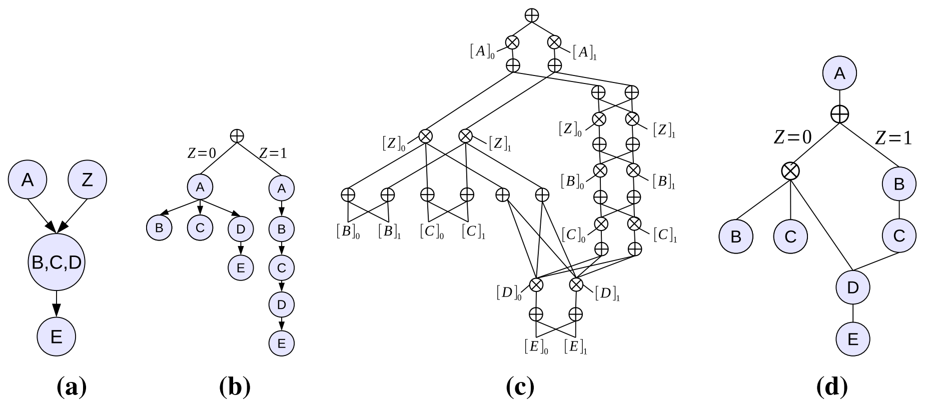

Fig. 2 The distribution in Eq. 1.4 represented (from left to right) as graphical model (GM), as mixture of GMs, as sum–product network (SPN) and as sum–product graphical model (SPGM)

1.2 Sum–product graphical models

The previous section showed that SPNs conveniently represent context specific independence and algorithm structures for inference, whereas GMs directly display conditional independence through factorization. Several attempts were made in the literature to close this gap. We discuss related work in Sect. 1.4.

Our approach to this problem is to introduce a new representation, called *sum–product graphical model (SPGM)*, that directly *inherits* the favorable traits from both GMs and SPNs. SPGMs can be seen as an extension of SPN that, along with product and sum nodes as internal nodes, also comprises *variable nodes* which have the same role as usual nodes in graphical models. Alternatively, SPGMs can be seen as an extension of directed GMs by adding sum and product nodes as internal nodes. The SPGM representing the distribution (1.4) is shown by Fig. 2d. It clearly reveals both the mixture of the two tree-structured subgraphs and the shared components. Thus, in this case SPGMs exhibit *both* the expressiveness of SPNs and the high level semantics of GMs.

More generally, we show that every SPGM implements a mixture of trees with shared subparts, as in the above example. Hence, the family of GMs that SPGMs can represent is limited to mixtures of trees,[4] with the crucial difference that very large mixtures can be modeled due to sharing factors between such trees. We also show that this is the same family of models that SPNs can represent.

In SPGMs, explicit *context variables* attached to sum nodes implement *context specific independence* (like Z in Fig. 2d) and select trees as model components to be combined. *Conditional independence* between variables, on the other hand, can be read off from the graph due to D-separation (Cowell et al. 2003). SPGMs enable one to represent in this way *very large* mixtures, whose size grows exponentially with the model size and are thus intractable if represented as a standard mixture model. On the other hand, inference in SPGM has a worst case complexity that merely is *quadratic* in the SPGM size and effectively is quasi-linear in most practical cases.[5]

[4] An extension to mixtures of *junction trees* (Cowell et al. 2003) is straightforward but does not essentially contribute to the present discussion and hence is omitted.

[5] More precisely, the complexity is $O(NM)$, where N is the number of nodes and M is the maximal number of parent nodes, of any node in the model.

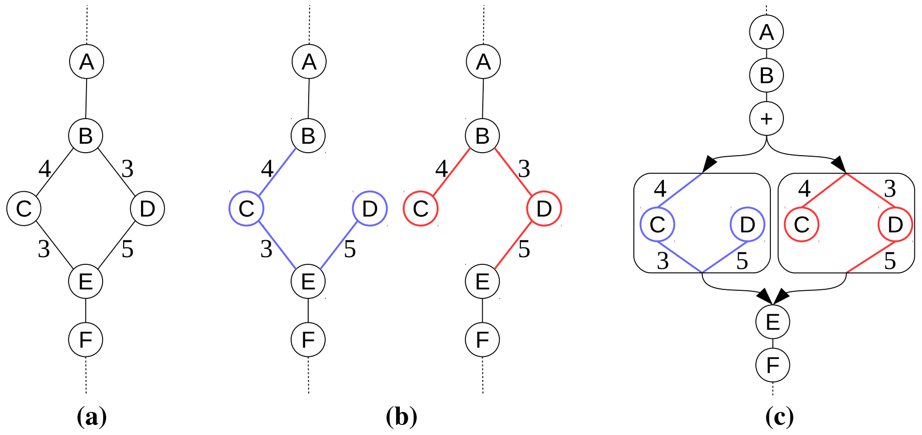

Fig. 3 Sketch of the structure learning algorithm proposed in this paper. **a** A weighted subgraph on which we compute the maximal spanning trees. **b** Two maximal spanning trees of equal weight to included as mixture components into the SPGM. They differ only in a single edge. **c** The mixture of the two trees represented by sharing all common parts

In addition, SPGMs generally provide an equivalent but more compact and high level representation of SPNs, with the additional property that the role of variables with respect to both contextual and conditional independence remains explicit. A compilation procedure through message passing allows one to convert the SPGM (Fig. 2d) into the equivalent SPN (Fig. 2c) which directly supports computational inference.

1.3 Structure learning

Learning the structure of probabilistic models obviously is easier for models with tractable inference than for intractable ones, because any model parameter learning algorithm requires inference as a subroutine. For this reason, tractable probabilistic models and especially SPNs have been widely applied for density estimation (Gens and Domingos 2013; Rooshenas and Lowd 2014; Rahman and Gogate 2016b, a) (Fig. 3).

We provide a density estimation algorithm for SPGMs that exploits the connections between GMs and SPNs enabled by the new model. Our algorithm starts with fitting a single tree GM in the classical way (Chow and Liu 1968) and iteratively inserts sub-optimal trees that have large weights (in terms of the mutual information of adjacent random variables) and share as many edges as possible with existing tree components. Each insertion is guaranteed not to decrease the global log-likelihood, thereby reaching a local optimum. As a result, all informative edges can be included into the model without compromising computational efficiency, because all shared components are evaluated only once. The former property is not true if a single tree is only fitted (Chow and Liu 1968) whereas working directly with large tree mixtures (Meila and Jordan 2000) may easily lead to a substantial fraction of redundant computations.

Our approach is different from previous methods for learning the structure of SPNs, which mostly implement recursive partitioning of the variables into (approximately) independent clusters, to be represented by sum and product nodes (Gens and Domingos 2013). The results of a comprehensive experimental evaluation will be reported in Sect. 5.

1.4 Related work

Table 1 lists and classifies prior work with a similar scope: introducing representations of probability distributions that fill to some extent the gap between GMs and SPNs. The caption of Table 1 lists the properties used to classify related work.

A major aspect of an SPGM is that it encodes a SPN through message passing. This will be made precise formally in Sect. 3.4. As a consequence, SPGMs relate to *Arithmetic Circuits* (Darwiche 2002), which differ from SPNs only in the way connection weights are represented, and to *And/Or Graphs* (Dechter and Mateescu 2007), which are structurally equivalent to Arithmetic Circuits and to SPNs despite employing a slightly different probablistic formalism—we refer to Dechter and Mateescu (2007, Section 7.6.1) and Poon and Domingos (2011) Sec. 3 for a discussion of details. As discussed in Sect. 1.2, SPGMs encode the computational structure for efficient inference like SPNs and related representations, but also preserve explicitly factorization properties of the underlying distribution due to conditional independence.

Furthermore, the concept of SPGM subtrees described in Sect. 3.3 has a close parallel in the concept of SPN subnetworks, first described in Gens and Domingos (2012) and then formalized in Zhao et al. (2016). However, while subnetworks in SPNs represent simple factorizations of the leaf distributions (which can be represented as graphical models without edges), subtrees in SPGMs represent tree graphical models.

SPGMs are closely related to hierarchical mixtures of trees (HMT) (Jordan 1994) and *generalize* them. Like SPGMs, HMTs allow a compact representation of mixtures of trees by using a hierarchy of "choice nodes" where different trees are selected at each branch (as sum nodes do in SPGMs). While in HMTs the choice nodes separate the graph into disjoint branches and thus an overall tree structure is induced, however, SPGMs enable one to use a DAG structure where parts of the graph towards the leaves can appear in children of multiple sum nodes.

The probabilistic model encoded by SPGMs also has a close connection to Gates (Minka and Winn 2009) and the similarly structured mixed Markov models (MMM) (Fridman 2003). Gates enable contextual independence in graphical models by including the possibility of activating/deactivating factors based on the state of some context variables. Regarding SPGMs, the inclusion/exclusion of factors in subtrees $\mathcal{T}(\mathbb{S}|Z)$ depending on values of Z can be seen as a gating unit that enables a full set of tree factors to be active, which suggest identifying an SPGM as a Gates model. On the other hand, SPGMs restrict inference to a family of models in which inference is tractable by construction, while inference in Gates generally is intractable. In addition, SPGMs allow an interpretation as mixture models, which is not the case for Gates and MMMs.

Finally, SPGMs are related to several methods that augment GMs by factors with complex structure in order to represent context specific independence (Boutilier et al. 1996; Mcallester et al. 2004; Chickering et al. 2013; Gogate et al. 2010; Bacchus et al. 2012). These approaches enable one to represent product of factors like graphical models. However, the additional model complexity due to contextual independence is simply encapsulated inside the factors, based on models that are equivalent to SPNs and thus exhibit corresponding limitations (Sect. 1.1). In particular, in connection with distributions that combine both conditional and contextual independence, the approaches have to resort to a low-level SPN-like representation. On the other hand, if simpler factors (such as with distributions from the exponential family) were used instead, the model would lose its expressivenes.

1.5 Contribution and organization

The contributions of this paper are as follows:

- The introduction of SPGMs, which connect SPNs to graphical models in that they possess high level *conditional independence* semantics akin to graphical models. Moreover, they enable *tractable* inference and represent *context specific* dependences and determinism as SPNs.
- A convergent structure learning algorithm for SPGMs based on mixtures of spanning trees. This algorithm is deployed in a density estimation task using 20 real-life datasets (Sect. 5). It obtains best results in 5 datasets over 20 comparing against 8 state of the art methods, and close performances in the other cases, despite using a straightforward algorithm based on mixtures of trees. Our method could be straightforwardly improved by applying the simplification and regularization techniques discussed in Vergari et al. (2015) on the SPN encoded by our model, which is however out of the focus of this paper.
- The application of SPGMs to the approximation of the distribution of a known but intractable GM, which we choose in a restricted but relevant family which is easily amenable by SPGMs. This is the first application of this kind for a model in the family of SPNs. SPGMs demonstrate strong empirical performances against competing methods, which highlights the importance of exploiting jointly contextual and conditional independence.

Structure of the Paper This paper is organized as follows. Section 2 contains notation and background on graphical models and SPNs. Section 3 describes SPGMs and discusses their properties, analyzing the dual interpretation as mixture of tree graphical models and as SPN. Section 4 describes the proposed structure learning algorithm for SPGMs. Section 5 shows applications of SPGMs exploiting the new connections between GMs and SPNs in practical settings.

2 Background

2.1 Directed graphical models

The aim of this section is to fix the notation for the rest of the paper, and we assume that the reader is already familiar with the basic concepts of graphical models. For a more comprehensive introduction, see e.g. Wainwright and Jordan (2008).

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a Directed Acyclic Graph (DAG) with vertex set $\mathcal{V} = \{1, 2, \dots, N\}$ and edge set $\mathcal{E}$. We associate to each vertex $s \in \mathcal{V}$ a discrete random variable X_s taking values in the finite domain $\Delta(X_s)$, and $X = \{X_s\}_{s\in\mathcal{V}}$ denotes the set of all variables of the model, taking values in the Cartesian product $\Delta(X) := \Delta(X_1) \times \Delta(X_2) \times \cdots \times \Delta(X_N)$. We formally define the indicator variables already introduced in Sect. 1.1, Eqs. (1.2), (1.3).

Definition 1 (*Indicator Variables*) Let $Y \subseteq X$ be a subset of variables to which the values $y \in \Delta(Y)$ are assigned: $X_v = y_v,\ \forall X_v \in Y$. Based on the assignment y, we associate with every variable $X_s \in X$ and every value $i \in \Delta(X_s)$ the *indicator variable* $[X_s]_i \in \{0, 1\}$ defined by

$$[X_s]_i = \begin{cases} 1 & \text{if } (X_s \in Y \text{ and } y_s = i) \text{ or } (X_s \notin Y), \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

We denote by $pa(s)$ the parents of s in $\mathcal{G}$: $pa(s) = \{r \in \mathcal{V}\colon (r, s) \in \mathcal{E}\}$.

A *directed graphical model (Directed GM)* on a graph $\mathcal{G}$ comprises conditional probabilities $P_{s,t}(X_t|X_s)$ for every directed edge $(s, t) \in \mathcal{E}$ and unary probabilities $P_r(X_r)$ for each vertex $r \in \mathcal{V}$ with no parent, and encodes the distribution

$$P(X) = \prod_{r\in\mathcal{V}\colon\ pa(r)=\emptyset} P_r(X_r) \prod_{(s,t)\in\mathcal{E}} P_{s,t}(X_t|X_s)\,. \tag{2.2}$$

A *directed tree graphical model (Tree GM)* is a directed GM where the underlying graph $\mathcal{G} = \mathcal{T}$ is a rooted tree $\mathcal{T}$ with root r. Since each vertex s has at most one parent $pa(s)$, the distribution (2.2) reads

$$T(X) = P_r(X_r) \prod_{s\in\mathcal{V}\colon\ pa(s)\neq\emptyset} P_{pa(s),s}\left(X_s|X_{pa(s)}\right). \tag{2.3}$$

Marginalization and Maximum a Posteriori (MAP) inference in general directed GMs has a cost that is exponential in the treewidth of the triangulated graph obtained by moralization of the original graph, and thus is intractable for graphs with cycles of non trivial size (Cowell et al. 2003; Diestel 2006). However, in *tree* GMs inference can be computed efficiently with *message passing*. Let $Y \subseteq X$ be a set of observed variables with assignment $y \in \Delta(Y)$. Let $[X_s]_j,\ s \in \mathcal{V},\ j \in \Delta(X_s)$ denote indicator variables due to Definition 1. Node t sends a message $\mu_{t\to s;j}$ to its parent s for each state $j \in \Delta(X_s)$ given by

$$\mu_{t\to s;j} = \sum_{k\in\Delta(X_t)} P_{s,t}(k|j)\,[X_t]_k \prod_{(t,q)\in\mathcal{E}} \mu_{q\to t;k}. \tag{2.4}$$

Setting $C = X \setminus Y$ and $x = (y, c)$, marginal probabilities $T(Y = y) = \sum_{c\in\Delta(C)} T(Y = y, C = c)$ can be computed using the distribution (2.3) by first setting the indicator variables according to the assignment y (Definition 1), then passing messages for every node in reverse topological order (from leaves to the root), and finally returning the value of the root message. Since message passing in trees only requires computing one message per each node, the procedure has complexity $O(|\mathcal{V}|\Delta_{max}^2)$, where $\Delta_{max} = \max\{|\Delta(X_s)|\colon s \in \mathcal{V}\}$ is the maximum domain size. As a consequence, tree GMs enable *tractable marginal inference*. Computing MAP queries in SPNs is however an NP-hard problem, for which several tractable approximations exist (see Conaty et al. 2017; Mei et al. 2018).

Graphical models conveniently encode conditional independence properties of a distribution. Conditional independence between variables in GMs are induced from the graph by D-separation (see, e.g., Cowell et al. 2003). In the case of tree graphical models, D-separation becomes particularly simple: if the path between variables A and B contains C, then A is conditionally independent from B given C.

It is well-known, however, that another form of independence is not covered well by the GMs: independence that only holds in certain contexts, i.e. depending on the assignment of values to a specific subset of so-called context variables.

Definition 2 (*Contextual Independence*) Variables A and B are said to be *contextually independent* given Z and *context* $z \in \Delta(Z)$ if $P(A, B|Z = z) = P(A|Z = z)P(B|Z = z)$.

Remark 1 Notice that *conditional* independence is a special case of *contextual* independence in which $P(A, B|Z = z) = P(A|Z = z)P(B|Z = z)$ would hold for *all* $z \in \Delta(Z)$. By contrast, *contextual* independence assures that this property only holds for a subset of value assignments that constitute the so-called *context*. In particular, different independences can hold for different values z—see Eq. (1.4b) for an illustration. We refer to Boutilier et al. (1996) for an in-depth discussion of contextual independence.

Encoding contextual independence mainly motivates the model class of sum–product networks formally introduced in Sect. 2.2.

Mixtures of Trees A *mixture of trees* is a distribution in the form $P(X) = \sum_{k=1}^{K} \lambda_k T_k(X)$ where $\{T_k(X)\}_{k=1}^{K}$ are directed tree GMs and $\{\lambda_k\}_{k=1}^{K}$ are real non-negative weights satisfying $\sum_{k=1}^{K} \lambda_k = 1$. Inference in mixture models involves taking the weighted sum of the results of inference in each tree. Hence it has K times the cost of inference in a single tree in the mixture.

A mixture model can also be expressed through a hidden variable Z with $\Delta(Z) = \{1, 2, \ldots, K\}$ by writing: $P(X, Z) = \prod_{k=1}^{K} (\lambda_k T_k(X))^{\delta(Z=k)}$. Then, it holds that $P(X) = \sum_{z \in \Delta(Z)} P(X, Z)$, $P(Z = k) = \lambda_k$ and $P(X|Z = k) = T_k(X)$. Note that different values of Z entail different independences due to different tree structures. It follows that mixtures of trees represent context-specific independence with context variable Z.

However, the family of conditional independences entailed by mixture models is limited to using a *single* context variable Z, and to the selection of different models defined on the *entire* set X for each value of Z. In contrast, the model class of sum–product networks to be introduced in the next section, allows one to model contextual independences depending on *multiple* context variables that only affect a *subset* of X—see, e.g., the example in Sect. 1.1.

2.2 Sum–product networks

Sum–product networks (SPN) were introduced in Poon and Domingos (2011). They are closely related to Arithmetic Circuits (Darwiche 2003). We adopt the definition of *decomposable* SPNs advocated by Gens and Domingos (2013). The expressiveness of these models was shown by Peharz (2015) to be equivalent to the expressiveness of non-decomposable SPNs.

Definition 3 (*sum–product network* (*SPN*)) Let $X = \{X_1, \ldots, X_N\}$ be a collection of random variables, and let $X = X^1 \cup X^2 \cup \cdots \cup X^K$ be a partition of X. A *sum–product network (SPN)* $S(X)$ is recursively defined and constructed by the following rule and operations:

1. An indicator variable $[X_s]_j$ is a SPN $S(\{X_s\})$.
2. The *product* $\prod_{k=1}^{K} S_k(X_k)$ is a SPN, with the SPNs $\{S_k(X_k)\}_{k=1}^{K}$ as factors.
3. The *weighted sum* $\sum_{k=1}^{K} w_k S_k(X)$ is a SPN, with the SPNs $\{S_k(X)\}_{k=1}^{K}$ as summands and non-negative weights $\{w_k\}_{k=1}^{K}$.

Remark 2 Due to Definition 1, indicator variables that form SPNs require the specification of an assignment y of a subset of variables $Y \subset X$, in order to be well-defined. Such assignments are specified in connection with the *evaluation* of a SPN, denoted by

$$S(y) := S(X \setminus Y, Y = y). \tag{2.5}$$

Example 1 The SPN shown by Fig. 1b displays the operations due to Definition 3. Overall, it represents the operations of the sum–product message passing procedure (2.4) with respect to the graphical model of Fig. 1a, where the indicator variables introduce evidence values (measurements, observations). Notice that indicator variables are the only variables in this SPN, and that the probabilities P_s, $P_{s,t}$ define weights attached to the sum nodes.

SPNs are evaluated in a way similar to message passing in tree graphical models (Eq. (2.4)). An evaluation $S(y)$ due to (2.5) is computed by first assigning indicator variables in

S according to y by Definition 1, then evaluating nodes in inverse topological order (leaves to root) and taking the value of the root node of S.

Poon and Domingos (2011) show that evaluating $S(y)$ corresponds to computing marginals in some valid probability distribution $P(X)$, namely: $S(y) = P(Y = y) = \sum_{x_{\setminus y} \in \Delta(X \setminus Y)} P(x)$, where $x_{\setminus y}$ denotes assignments to variables in the set $X \setminus Y$. In addition, evaluating $S(y)$ involves evaluating each node in the DAG, i.e. inference at $O(\mathcal{E})$ time and memory cost is always *tractable*.

The recursive Definition 3 enable one to build up SPNs by iteratively composing SPNs using the operations 2 and 3, starting from trivial SPNs as leaves in the form of indicator variables (rule 1). Clearly, it is not always intuitive to design a SPN by hand, which involves thinking of the product nodes as factorizations and of the sum nodes as mixtures of the underlying sub-models. For some particular applications, the SPN structure was designed and fitted in this way to the particular distribution at hand, cf. Poon and Domingos (2011), Cheng et al. (2014), Amer and Todorovic (2015). Yet, more generally, SPN approaches involve some form of automatically learning the structure of the SPN from given data (Gens and Domingos 2013). Our approach to learning the structure of a SPN is described in Sect. 4.

Besides tractability, the main motivation for considering the model class of SPNs is their ability to represent contextual independences more expressively than mixture models. A detailed analysis of the role of contextual independence in SPNs is provided in Sect. 3. Intuitively, the expressiveness of SPNs comes from interpreting sum nodes as *mixtures* of the children SPNs, which allows one to create a hierarchy of mixture models and thus a hierarchy of contextual independences, by using hidden mixture variables. In addition, sharing the same children SPNs in different sum nodes allows one to limit the effect of contextual independences to a subsection of the model, as shown in the example in Sect. 1.1.

3 Sum–product graphical models

This section contains the definition of *sum–product graphical models (SPGMs)* and a discussion of their properties. In Sect. 3.3 SPGMs are seen as GMs with additional context nodes representing conditional independence, which translates to a very large hierarchical mixture of trees. From this, a set of results available for GMs immediately translates to SPGMs. In Sect. 3.4, SPGMs are seen as a high-level representation of SPNs able to represent conditional independence compactly.

3.1 Definition

The first step in describing SPGMs is to define the nodes that appear in the underlying graph.

Definition 4 (*Sum, Product and Variable Nodes*) Let X and Z be disjoint sets of discrete variables, and let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a DAG.

- The basic nodes $s \in \mathcal{V}$ of an SPGM are called *SPGM Variable Node (Vnode)* and associated with a variable $X_s \in X$. They are graphically represented as a circle having X_s as label (Fig. 4, left).
- $s \in \mathcal{V}$ is called *Sum Node*, if it represents the corresponding operation indicated by the symbol $\oplus$. A Sum Node can be *Observed*, in which case it is associated with a variable $Z_s \in Z$ and represented by the symbol $\oplus Z_s$.

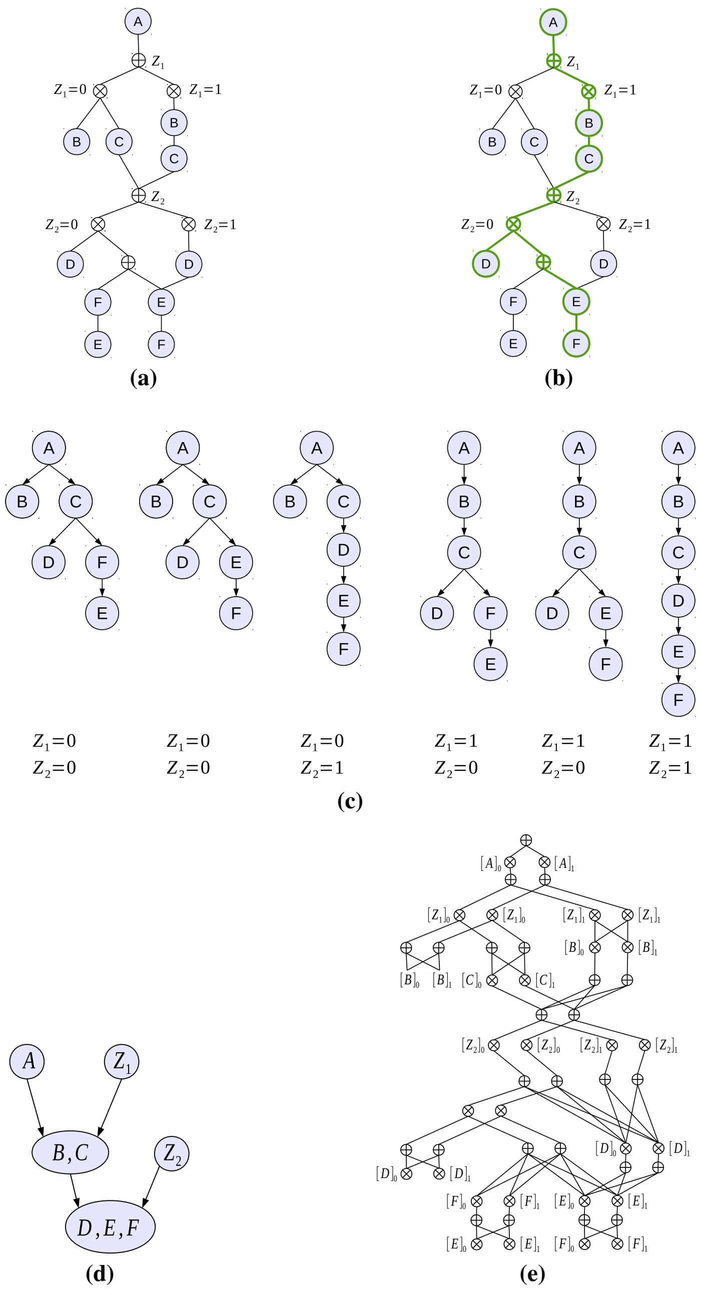

Fig. 4 Sum–product graphical model example. **a** An SPGM $\mathbb{S}(X, Z)$ with $X = \{A, B, C, D, E, F\}$, $Z = \{Z_1, Z_2\}$. **b** A subtree of $\mathbb{S}$ (in bold). **c** All subtrees of $\mathbb{S}$ represented as graphical models and corresponding context variables—note the shared parts between the trees. The distribution encoded by $\mathbb{S}$ is a mixture over these subtrees (Sect. 3.3). The same distribution can be represented as a Directed GM (**d**) or as a SPN (**e**), which highlights the trade off between efficiency and high level representation: the GM version (**d**) is less expressive due to the inability of representing contextual independences, while the SPN version (**e**) is as efficient as the SPGM but lacks the ability to represent conditional independence compactly

– $s \in \mathcal{V}$ is called *Product Node*, if it represents the corresponding operation indicated by the symbol $\otimes$.

In the following, variables Z take the role of *context variables* according to Definition 2.

Definition 5 (*Scope of a node*) Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a rooted DAG with nodes as in Definition 4, and let $s \in \mathcal{V}$.

– The *scope* of s is the set of all variables associated with nodes in the DAG rooted in s.
– The X*-scope* of s is the set of all variable associated with Vnodes in the DAG rooted in s.
– The Z*-scope* of s is the set of variables associated with Observed Sum Nodes in the DAG rooted in s.

Example 2 The scope of the Sum Node associated with Z_2 in Fig. 4a is $\{D, E, F, Z_2\}$, its Z-scope is $\{Z_2\}$, and its X-scope is $\{D, E, F\}$.

Finally, we define the set of "V-parents" of a Vnode s, which intuitively are the closest Vnode ancestors of s.

Definition 6 (*Vparent*) The *Vparent set* $vpa\,(s)$ of a Vnode s is the set of all $r \in \mathcal{V}$ such that r is a *Vnode*, and there is a directed path from r to s that does not include any other Vnode.

With the definitions above we can now define SPGMs.

Definition 7 (*SPGM*) A *sum–product graphical model (SPGM)* $\mathbb{S}\big(X, Z|\mathcal{G}, \{P_s\}, \{P_{st}\}, \{W_s\}, \{Q_s\}\big)$ or more shortly, $\mathbb{S}\,(X, Z)$ or even $\mathbb{S}$, is a rooted DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where nodes can be Sum, Product or Vnodes as in Definition 4. The SPGM is governed by the following parameters:

1. Pairwise conditional probabilities $P_{st}\,(X_t|X_s)$ associated with each Vnode $t \in \mathcal{V}$ and each Vparent $s \in vpa\,(t)$.
2. Unary probabilities $P_s\,(X_s)$ associated with each Vnode $s \in \mathcal{V}\colon\ vpa(s) = \emptyset$.
3. Unary probabilities $W_s(k)$ for $k = \{1, 2, ..., |ch(s)|\}$ associated with each non-Observed Sum Node s, with value $W_s(i)$ associated with the edge between s and its i-th child (assuming any order has been fixed).
4. Unary probability $Q_s(Z_s)$ s.t. $\Delta(Z_s) = \{1, 2, ..., |ch(s)|\}$ and associated with each Observed Sum Node s, with value $Q_s(i)$ being associated with the edge between s and its i-th child.

In addition, each node $s \in \mathcal{V}$ must satisfy the following conditions:

5. If s is a *Vnode* (associated with variable X_s), then s has *at most* one child c, and X_s does not appear in the scope of c.[6]
6. If s is a *Sum* Node, then s has *at least* one child, and the scopes of all children are the *same* set. If the Sum Node is Observed (hence associated with variable Z_s), then Z_s is not in the scope of any child.
7. If s is a *Product* Node, then s has *at least* one child, and the scopes of all children are *disjoint* sets.

[6] The one child policy for V-nodes is necessary because the associated message acts on a single node, as discussed in Sect. 3.2. This simplifies the subsequent propositions without loss in generality, since product nodes take the role of merging multiple children messages acting on disjoint variable sets.

An example SPGM is shown in Fig. 4, left. Note that the closeness with both SPNs and GMs, discussed later, can be already seen from the definition: the last three conditions in definition are closely related to SPN conditions (Definition 3), whereas the usage of pairwise and unary probabilities in 1.-4. above connects SPGMs to graphical models. In this sense, SPGMs can be seen as GMs with added context nodes: the results in the following section show how several properties of GMs immediately translate to SPGMs.

3.2 Message passing in SPGMs

We define a message passing protocol used to evaluate SPGMs, which is very similar to message passing in graphical models, with a few crucial differences related to context nodes. The following definition refers to Definitions 6 and 7.

Definition 8 (*Message passing in SPGMs*) Let $s \in \mathcal{V}$, $t \in \mathcal{V}$ and let $ch(s)_k$ denote the k-th child of s in a given order. Node t sends a message $\mu_{t\to s;j}$ to each Vparent $s \in vpa(t)$ and for each parent state $j \in \Delta(X_s)$ according to the following rules:

$$\mu_{t\to s;j} = \sum_{k=1}^{|ch(t)|} [Z_t]_k Q_t(k) \mu_{ch(t)_k \to s;j} \quad t \text{ is a Sum Node, Observed} \tag{3.1a}$$

$$\mu_{t\to s;j} = \sum_{k=1}^{|ch(t)|} W_t(k) \mu_{ch(t)_k \to s;j} \quad t \text{ is a Sum Node, not-Observed} \tag{3.1b}$$

$$\mu_{t\to s;j} = \prod_{q\in ch(t)} \mu_{q\to s;j} \quad t \text{ is a Product Node} \tag{3.1c}$$

$$\mu_{t\to s;j} = \sum_{k\in\Delta(X_t)} P_{s,t}(k|j)\,[X_t]_k \mu_{ch(t)\to t;k} \quad t \text{ is a Vnode} \tag{3.1d}$$

If $vpa(s)$ is empty, top level messages are computed as:

$$\mu_{t\to root} = \sum_{k=1}^{|ch(t)|} [Z_t]_k Q_t(k) \mu_{ch(t)_k \to root} \quad t \text{ is a Sum Node, Observed} \tag{3.1e}$$

$$\mu_{t\to root} = \sum_{k=1}^{|ch(t)|} W_t(k) \mu_{ch(t)_k \to root;j} \quad t \text{ is a Sum Node, not-Observed} \tag{3.1f}$$

$$\mu_{t\to root} = \prod_{q\in ch(t)} \mu_{q\to root;j} \quad t \text{ is a Product Node} \tag{3.1g}$$

$$\mu_{t\to root} = \sum_{k\in\Delta(X_t)} P_s(k)\,[X_t]_k \mu_{ch(t)\to t;k} \quad t \text{ is a Vnode} \tag{3.1h}$$

Vnodes at the leaves send messages as in Eqs. (3.1d) and (3.1h) after substituting the incoming messages with 1.

Note that that messages are only sent to Vnodes (or to a fictitious "root" for top level nodes), and no message is sent to Sum and Product Nodes. Notice further that Vnode messages resemble message passing in tree GMs (Eq. (2.4)), which is the base for our subsequent interpretation of SPGMs as graphical model.

Definition 9 (*Evaluation of* $\mathbb{S}(X, Z)$) Let $Y \subseteq X \cup Z$ denote evidence variables with assignment $y \in \Delta(Y)$. The evaluation of an SPGM $\mathbb{S}$ with assignment y, written as $\mathbb{S}(y)$ (cf. Remark 2 and (2.5)), is obtained by setting the indicator variables accordingly (Definition 1), followed by evaluating messages for each node from the leaves to the root due to Definition 8, and then taking the value of the message produced by the root of $\mathbb{S}$.

Proposition 1 *The evaluation of an SPGM* $\mathbb{S}$ *has complexity* $O(|\mathcal{V}|M|\Delta_{max}|^2)$, *where* M *is the maximum number of Vparents for any node in* $\mathbb{S}$, *and* $|\Delta_{max}| = \max\{\Delta(X_s) : s \in \mathcal{V}\}$ *is the maximum domain size for any variable in* X.

3.3 Interpretation of SPGMs as graphical models

In this section, we consider and discuss SPGMs as probabilistic models. We show that SPGMs encode large mixtures of trees with shared subparts and provide a high-level representation of both conditional and contextual independence through D-separation. Proofs can be found in Appendix A.

3.3.1 Subtrees

We start by introducing subtrees of SPGMs and their properties.

Definition 10 (*Subtrees of SPGMs*) Let $\mathbb{S}$ be an SPGM. A *subtree* $\tau(X, Z)$ (or more shortly τ) is an SPGM defined on a subtree of the DAG $\mathcal{G}$ underlying $\mathbb{S}$ (cf. Definition 7), that is recursively constructed based on the root of $\mathbb{S}$ and the following steps:

- If s is a *Vnode or a Product Node*, then include in τ all children of s and edges formed by s and its children. Continues this process for all included nodes.
- If s is a Sum Node, then include in τ only the k_s-th child and the corresponding connecting edge, where the choice of k_s is arbitrary. Continue this process for all included nodes.

We denote by $\mathcal{T}(\mathbb{S})$ the set of all subtrees of $\mathbb{S}$.

Example 3 One of the subtrees of the SPGM depicted in Fig. 4a is shown by Fig. 4b.

Definition 11 (*Subtrees τ and indicator variable sets z_τ*) Let $\tau \in \mathcal{T}(\mathbb{S})$ be a subtree of $\mathbb{S}$. The symbol z_τ denotes the set of all indicator variables associated with Observed Sum Nodes and their corresponding state in the subtree. Specifically, if the k_s-th child of an Observed Sum Node s is included in the tree, then $[Z_s]_{k_s} \in z_\tau$.

Example 4 The set z_τ for the subtree in Fig. 4b is $\{[Z_1]_1, [Z_2]_0\}$.

Definition 12 (*Context-compatible subtrees*) Let $Y \subseteq Z$ be a subset of context variables with assignment $y \in \Delta(Y)$, and let $[y]$ denote the set of indicator variables corresponding to $Y = y$. The set of subtrees compatible with context $Y = y$, written as $\mathcal{T}(\mathbb{S}|y)$, is the set of all subtrees $\tau \in \mathcal{T}(\mathbb{S})$ such that $[y] \subseteq z_\tau$.

Example 5 The set of subtrees $\mathcal{T}(\mathbb{S}|Z_1 = 0, Z_2 = 0)$ for the SPGM in in Fig. 4 is composed by subtree τ, shown in Fig. 4b, and the subtree obtained by modifying τ through choosing the alternate child of the lowest sum node.

We now state properties of subtrees that are essential for the subsequent discussion.

Proposition 2 *Any subtree $\tau \in \mathcal{T}(\mathbb{S})$ is a tree SPGM.*

Proposition 3 *The number $|\mathcal{T}(\mathbb{S})|$ of subtrees of $\mathbb{S}$ grows as $O(\exp(|\mathcal{E}|))$.*

Proposition 4 *The scope of any subtree $\tau(X, Z) \in \mathcal{T}(\mathbb{S}(X, Z))$ is $\{X, Z\}$.*

Table 2 Probabilistic models related to an SPGM $\mathbb{S}(X, Z)$

$$P(X, Z) = \sum_{\tau \in \mathcal{T}(\mathbb{S}|Z)} \lambda_\tau P_\tau(X) \tag{3.2}$$

$$P(X) = \sum_{\tau \in \mathcal{T}(\mathbb{S})} \lambda_\tau P_\tau(X) \tag{3.3}$$

$$P_\tau(X) = \prod_{r \in V_\tau : \, vpa(r) = \emptyset} P_r(X_r) \prod_{s \in V_\tau, t : \in V_\tau : \, s \in vpa(t)} P_{s,t}(X_t | X_s) \tag{3.4}$$

$$\lambda_\tau = \prod_{s \in O_\tau} Q_s(k_{s,\tau}) \prod_{s \in U_\tau} W_s(k_{s,\tau}) \tag{3.5}$$

Symbols V_τ, O_τ and U_τ denote respectively the set of Vnodes, Observed Sum Nodes and Unobserved Sum Nodes in a subtree $\tau \in \mathcal{T}(\mathbb{S})$. $k_{s,\tau}$ denotes the index of the child of s that is included in τ (see Definition 10). Evaluation of $\mathbb{S}$ (Definition 9) is equivalent to inference using the distribution (3.2), which is a mixture distribution (3.3) of tree graphical models (3.4), whose structure *depends on the context* as specified by the context variables Z of (3.2)

3.3.2 SPGMs as mixtures of subtrees

In this section, we show that SPGMs can be interpreted as mixtures of trees. Table 2 lists the notation and probabilistic (sub-)models relevant in this context.

As a first step, we show that inference in a subtree τ (Definition 10) is equivalent to inference in a tree graphical model of the form (2.3), multiplied for a constant factor determined by the sum nodes in the subtree.

Proposition 5 *Let $\mathbb{S} = \mathbb{S}(X, Z)$ be an SPGM, and let $\tau \in \mathcal{T}(\mathbb{S})$ be a subtree (Definition 10) with indicator variables z_τ (Definition 11). Then message passing in τ is equivalent to inference using the distribution*

$$\lambda_\tau P_\tau(X) \prod_{[Z_s]_j \in z_\tau} [Z_s]_j, \tag{3.6}$$

where $P_\tau(X)$ is a tree graphical model of the form (3.4), $\lambda_\tau > 0$ is a scalar term obtained by multiplying the weights of all sum nodes in τ given by (3.5), and $\prod_{[Z_s]_j \in z_\tau} [Z_s]_j$ is the product of all indicator variables in z_τ.

The second step consists in noting that $\mathbb{S}$ represents the same distribution as the mixture of all its subtrees.

Proposition 6 *Evaluating $\mathbb{S}(X, Z)$ is equivalent to evaluating $\sum_{\tau \in \mathcal{T}(\mathbb{S})} \tau(X, Z)$.*

We are now prepared to state the main result of this section.

Proposition 7 *Let $Y_x \subseteq X, Y_z \subseteq Z$ denote evidence variables with assignment $y_x \in \Delta(Y_x)$, $y_z \in \Delta(Y_z)$, respectively, and denote by $x_{\backslash y} \in \Delta(X \setminus Y_x)$, $z_{\backslash y} \in \Delta(Z \setminus Y_z)$ assignments to the remaining variables. Evaluating $\mathbb{S} = \mathbb{S}(X, Z)$ with assignment (y_x, y_z) (Definitions 8 and 9) is equivalent to performing marginal inference with respect to the distribution (3.2) as follows:*

$$P(Y_x = y_x, Y_z = y_z) = \sum_{\substack{x_{\backslash y} \in \Delta(X \setminus Y_x) \\ z_{\backslash y} \in \Delta(Y \setminus Y_z)}} P\big((X \setminus Y_x) = x_{\backslash y}, (Y \setminus Y_z) = z_{\backslash y}, Y_x = y_x, Y_z = y_z\big). \tag{3.7}$$

Example 6 All subtrees of the SPGM $\mathbb{S}(X, Z)$ shown by Fig. 4a are shown as tree graphical models by Fig. 4c. The probabilistic model encoded by the SPGM is a mixture of these subtrees whose structure depends on the context variables Z.

The propositions above entail the crucial result that the probabilistic model of an SPGM is a mixture of trees where the mixture size grows *exponentially* with the SPGM size (Definition 10), but in which the inference cost grows only *polynomially* (Proposition 1). Hence, *very large* mixtures models can then be modelled *tractably*. This property is obtained by modelling trees $\tau \in \mathcal{T}(\mathbb{S}|Z)$ by combining sets of *shared subtrees*, selected through context variables Z,[7] and by computing inference in shared parts only once (cf. the example in Fig. 4).

3.3.3 Conditional and contextual independence

In this section we discuss conditional and contextual independence semantics in SPGMs, based on their interpretation as mixture model.

Definition 13 (*Context-dependent paths*) Consider variables $A \in X$, $B \in X$ and a context $z \in \Delta(\overline{Z})$ with $\overline{Z} \subseteq Z$.

- The set $\pi(A, B)$ is the set of all directed paths in $\mathbb{S}$ going from a Vnode with label A to a Vnode with label B.
- The set $\pi(A, B|\overline{Z} = z) \subseteq \pi(A, B)$ is the subset of paths in $\pi(A, B)$ in which all the indicator variables over $\overline{Z}$ (Definition 11) are in state z.

Proposition 8 *(*D-separation in SPGMs*) Consider an SPGM $\mathbb{S}(X, Z)$, variables $A, B, C \in X$ and a context $z \in \Delta(\overline{Z})$ with $\overline{Z} \subseteq Z$. The following properties hold for the probabilistic model $\mathbb{S}$ corresponding to Eq. (3.2):*

1. *A and B are* independent *iff $\pi(A, B) = \emptyset$ and $\pi(B, A) = \emptyset$ (there is no directed path from A to B).*
2. *A and B are* conditionally independent *given C if all directed paths $\pi(A, B)$ and $\pi(B, A)$ contain C.*
3. *A and B are* contextually independent *given context $\overline{Z} = z$ iff $\pi(A, B|\overline{Z} = z) = \emptyset$ and $\pi(B, A|\overline{Z} = z) = \emptyset$.*
4. *A and B are* contextually and conditionally independent *given C and context $\overline{Z} = z$ iff all paths $\pi(A, B|\overline{Z} = z)$ and $\pi(B, A|\overline{Z} = z)$ contain C.*

Example 7 In Fig. 4, A and D are conditionally independent given C; A and C are conditionally independent given B and context $Z_1 = 1$.

The proposition above provides SPGMs with a high level representation of both contextual and conditional independence. This is obtained by using different variables sets X and Z for the two different roles. The set X appears in Vnodes and entails conditional independences due to D-separation, with close similarity to tree graphical models (Proposition 8). The set Z enable contextual independence through the selection of tree branches via sum nodes.

Note also that using the set of paths π allows one to infer conditional and contextual independences without the need to check all the individual subtrees, whose number can be exponentially larger than the cardinality of π.

[7] Remark: unobserved sum nodes simply compute the weighted sum of the children subtrees, hence they are equivalent to observed sum nodes where the observed variable has been marginalized out.

3.4 Interpretation as SPN

In this section we discuss SPGMs as a high level, fully general representation of SPNs as defined by Definition 3.

3.4.1 SPGMs encode SPNs

Proposition 9 *The message passing procedure in $\mathbb{S}(X, Z)$ encodes a SPN $S(X, Z)$.*

As can be seen in the proof (Sect. A.9) and in the example of Fig. 5, each SPGM message is represented by a set of sum nodes in the SPN encoded by an SPGM. A sum node represents the value of the message corresponding to a certain state of the output variable (namely, $\mu_{t\to s;j}$ for each j). This entails an increase in representation size (but not in inference cost) by a $|\Delta_{max}|^2$ factor when passing from the SPGM to SPN representation. Note also that the meaning of nodes as implementing conditional and contextual independence is lost during the conversion to SPN, since both SPGM Vnode and SPGM sum node messages translate into SPN sum nodes.

Proposition 10 *SPGMs are as* expressive *as SPNs, in the sense that if a distribution $P(X, Z)$ can be represented as an SPN with inference cost C, then it can also be represented as an SPGM with inference cost C and vice versa.*

Propositions 9 and 10, together with the connections to graphical models worked out above, enable an interpretation of an SPGM $\mathbb{S}$ as a high-level representation of the encoded SPN S where contextual independence is explicit. These results generalize what the introductory example demonstrated by comparing Fig. 2c with Fig. 2d.

The SPGM representation is more *compact* than SPNs because it can represent compactly both contextual (through sum nodes) and conditional independence (through Vnodes). Passing from an SPGM to the SPN representation entails an increase of number of nodes in the

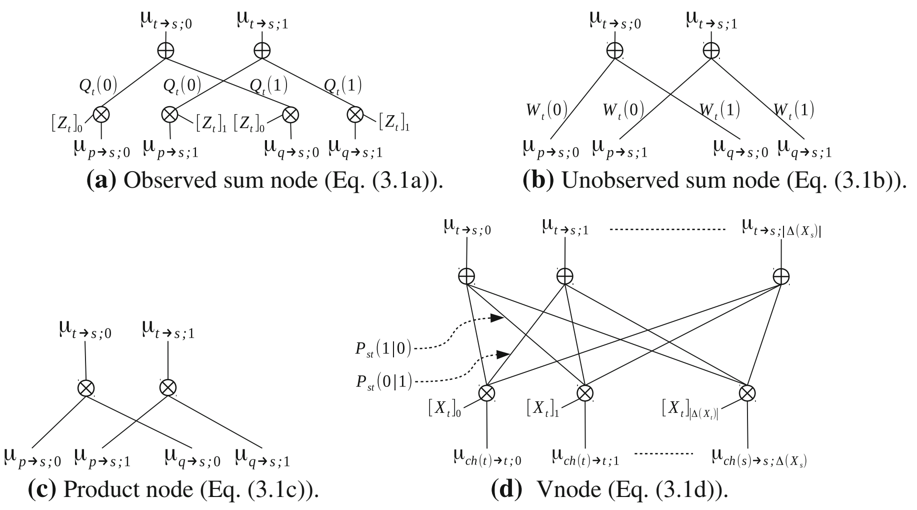

Fig. 5 The message passing equations of SPGMs induce a SPN. For better visibility, sum and product nodes are assumed to have only two children p, q and with binary variables

graph due to the expansion of messages by a $N_{pa}|\Delta_{max}|^2$ factor (see Definition 8 and Fig. 5). Note, however, that inference cost remains *identical* in $\mathbb{S}$ and S, as SPGMs are not more expressive than SPNs.

In contrast to SPGMs, the role of contextual and conditional independence in SPN nodes is hard to decipher (Fig. 2c) because there is no distinction between nodes created by messages sent from SPGM sum nodes (implementing contextual independence) and messages generated from SPGM Vnodes (implementing conditional independence), both of which are represented as a set of SPN sum and product nodes (see Fig. 5). In addition, there is no distinction between contextual variables and Vnode variables.

It is important to remark that SPGMs are *not* more compact than SPNs in situations in which there are no conditional independences that can be expressed by Vnodes. However, we postulate that the co-occurrence of conditional and contextual independences creates relevant application scenarios (as shown in Sect. 4) and enables connections between SPNs and graphical models that can be exploited in future work.

Finally, the interpretation of SPGMs as SPN allows one to translate all methods and procedures available for SPNs to SPGMs. These include jointly computing the *marginals* of all variables by derivation (Darwiche 2003), with time and memory linear cost in the number of edges in the SPN. In addition, Maximum a Posteriori queries can be computed simply by substituting the sums in Eq. (3.1) by max operations. We leave the exploration of these aspects to future work, since they are not central for our present discussion.

Thus, there is generally no drawback in employing the SPGM representation over SPNs, since SPGMs offer higher level Conditional Independence semantics and a more compact graphical representation of the distribution while still keeping the full SPN expressivity. However, one could prefer using the SPN representation when dealing with structure learning algorithms since the simpler definition of SPNs over SPGMs may be easier to work with, or simply in order to use one of the many learning algorithms available in literature for this class of models.

4 Learning SPGMs

In this section, we exploit the relations between graphical models and SPNs embodied by SPGMs and present an algorithm for learning the structure and parameters of SPGMs.

4.1 Preliminaries

Structure learning denotes the problem of learning both the parameters of a probability distribution $P(X|\mathcal{G})$ and the structure of the underlying graph $\mathcal{G}$. As both the GM and the SPN represented by a given SPGM due to Sects. 3.3 and 3.4 involve the same graph, the problem is well defined from both viewpoints.

Let $X = \{X_j\}_{j=1}^{M}$ be a set of M discrete variables. Consider a training set of N i.i.d samples $D = \left\{x^i\right\}_{i=1}^{N} \subset \Delta(X)$, used for learning. Formally, we aim to find the graph $\mathcal{G}^*$ governing the distribution $P(X)$ which maximizes the log-likelihood

$$\mathcal{G}^*(X) = \arg\max_{\mathcal{G}} \mathrm{LL}(G) = \arg\max_{\mathcal{G}} \sum_{i=1}^{N} \ln P(x^i|\mathcal{G}) \tag{4.1}$$

or the weighted log-likelihood

$$\mathcal{G}^*(X) = \arg\max_{\mathcal{G}} \mathrm{WLL}(G, w) = \arg\max_{\mathcal{G}} \sum_{i=1}^{N} w_i \ln P(x^i|\mathcal{G}), \qquad w_i \geq 0, \quad i = 1, 2, \ldots, N. \tag{4.2}$$

Learning Tree GMs Learning the structure of GMs generally is NP-hard. For discrete tree GMs however the maximum likelihood solution T^* can be found with cost $O(M^2N)$ using the Chow–Liu algorithm (Chow and Liu 1968).

Let $X_s, X_t \in X$ be discrete random variables with assignments ranging over the sets $\Delta(X_s)$, $\Delta(X_t)$. Let $N_{s;j}$ and $N_{st;jk}$ respectively count the number of times X_s appears in state j and X_s, X_t appear jointly in state j, k in the training set D. Finally, define empirical probabilities $\overline{P}_s(j) = N_{s;j}/N$ and $\overline{P}_{st}(k|j) = N_{st;jk}/N_{s;j}$. The Chow–Liu algorithm comprises the following steps:

1. Compute the *mutual information* $\mathbb{I}_{st}$ between all variable pairs X_s, X_t,
$$\mathbb{I}_{st} = \sum_{j \in \Delta(X_s)} \sum_{k \in \Delta(X_t)} \overline{P}_{s,t}(j,k) \ln \frac{\overline{P}_{s,t}(j,k)}{\overline{P}_s(j)\overline{P}_t(k)}. \tag{4.3}$$
2. Create an undirected graph $\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}})$ with adjacency matrix $\mathbb{I} = \{\mathbb{I}_{st}\}_{s,t \in \mathcal{V}}$ and compute the corresponding Maximum Spanning Tree $\overline{T}$.
3. Obtain the directed tree T^* by choosing an arbitrary node of $\overline{T}$ as root and using empirical probabilities $\overline{P}_s(X_s)$ and $\overline{P}_{st}\,(X_t|X_s)$ in place of corresponding terms in Eq. (2.2).

If the weighted log-likelihood (4.2) is used as objective function, the algorithm remains the same. The only difference concerns the use of *weighted relative frequencies* for defining the empirical probabilities of (4.3): $\hat{N}_{s,j} = \frac{1}{\hat{N}_w} \sum_{i=1}^{N} \delta(x_s^i = j) w_i$ and $\hat{N}_{st,jk} = \frac{1}{\hat{N}_w} \sum_{i=1}^{N} \delta(x_s^i = j, x_t^i = k) w_i$, where $\hat{N}_w = \sum_{i=1}^{N} w_i$ and x_s^i denotes the state of variable X_s in sample x^i.

Learning Mixtures of Trees We consider mixture models of the form $P(X) = \sum_{k=1}^{K} \lambda_k P_k(X|\theta^k)$ with tree GMs $P_k\,(X|\,\theta^k)$, $k = 1, \ldots, K$, corresponding parameters $\{\theta^k\}_{k=1}^K$ and non-negative mixture coefficients $\{\lambda_k\}_{k=1}^K$ satisfying $\sum_{k=1}^K \lambda_k = 1$. While inference with mixture models is tractable as long as it is tractable with its individual mixture components, maximum likelihood generally is NP hard. A local optimum can be found with expectation–maximization (EM) (Dempster et al. 1977), whose pseudocode is shown in Algorithm 1. The M-step (line 8) involves the weighted maximum likelihood problem and determines θ^k using the Chow–Liu algorithm described above.

It is well known that each EM iteration does not decrease the log-likelihood, hence it approaches a local optimum.

Learning SPNs Let $S(X)$ denote a SPN, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and graph with edge weights. Both *structure learning* (optimizing $\mathcal{G}$ and W) and *parameter learning* (optimizing W only) are NP-hard in SPNs (Darwiche 2002). Hence, only algorithms that seek a local optimum can be devised.

Parameter learning can be performed by directly applying the EM iteration for mixture models, while efficiently exploiting the interpretation of SPNs as a large mixture model with shared parts (Desana and Schnörr 2016).
To describe EM for SPNs, which will be used in a later section, we need some additional

Algorithm 1 EM for Mixture Models(P_k, λ_k, D)

```
Input: Initial model $P(X|\theta) = \sum_k \lambda_k P_k(X)$, training set $D$
Output: $P_k(X)$, $\lambda_k$ locally maximizing $\sum_{i=1}^N \ln P(x^i|\theta)$
1: repeat
2:     for all $k \in 1...K, i \in 1...N$ do // E-step
3:         $\gamma_k(i) \leftarrow \frac{\lambda_k P_k(x_i)}{\sum_{k'} \lambda_{k'} P_{k'}(x_i)}$
4:         $\Gamma_k \leftarrow \sum_{i=1}^N \gamma_k(i)$
5:         $w_i^k \leftarrow \gamma_k(i)/\Gamma_k$
6:     for all $k \in 1...K$ do // M-step
7:         $\lambda_k \leftarrow \Gamma_k/N$
8:         $\theta^k \leftarrow \arg\max_{\theta^k} \sum_{i=1}^N w_i^k \ln P_k(x^i|\theta^k)$
9: until convergence
```

notation. Consider a node $q \in \mathcal{V}$, and let S_q denote the sub-SPN having node q as root. If q is a Sum Node, then by Definition 3 a weight w_j^q is associated with each edge $(q, j) \in \mathcal{E}$. Note that evaluating $S(X = x)$ entails computing $S_q(X = x)$ for each node $q \in \mathcal{V}$ due to the recursive structure of SPNs. Hence $S(x)$ is function of $S_q(x)$. The derivative $\partial S(x)/\partial S_q(x)$ can be computed with a root-to-leaves pass requiring $O(|\mathcal{E}|)$ operations (Poon and Domingos 2011).
With this notation, the EM algorithm for SPNs iterates the following steps:

1. *E step*. Compute for each Sum Node $q \in \mathcal{V}$ and each $j \in ch(q)$

$$\beta_j^q = w_j^q \sum_{n=1}^{N} S\left(x^n\right)^{-1} \frac{\partial S\left(x^n\right)}{\partial S_q} S_j\left(x_n\right). \tag{4.4}$$

2. *M step*. Update weights for each Sum Node $q \in \mathcal{V}$ and each $j \in ch(q)$ by $w_j^q \leftarrow \beta_j^q / \sum_{(q,i)\in\mathcal{E}} \beta_i^q$, where $\leftarrow$ denotes assignment of a variable.

Since all the required quantities can be computed in $O(|\mathcal{E}|)$ operations, EM has a cost $O(|\mathcal{E}|)$ per iteration (the same as an SPN evaluation).
In some SPN applications, weights are shared among different edges (see e.g. Gens and Domingos 2012; Cheng et al. 2014; Amer and Todorovic 2015). Then the procedure still maximizes the likelihood locally. Let $\overline{V} \subseteq V$ be a subset of Sum Nodes with shared weights, in the sense that the set of weights $\{w_j^q\}_{j\in ch(q)}$ associated with incident edges $(q, j) \in \mathcal{E}$ is the same for each node $q \in \overline{V}$. The EM update of a shared weight w_j^q reads (cf. Desana and Schnörr 2016)

$$w_j^q \leftarrow \frac{\sum_{q\in\overline{V}} \beta_j^q}{\sum_i \sum_{q\in\overline{V}} \beta_i^q}, \qquad (q, j) \in \mathcal{E}. \tag{4.5}$$

Structure learning can be more conveniently done with SPNs than with graphical models, because tractability of inference is always guaranteed and hence not a limiting factor for learning the model's structure. Several greedy algorithms for structure learning were devised (see Sect. 5), which established SPNs as state of the art models for the estimation of probability distributions. We point out that most approaches employ a recursive procedure in which children of sum and Product Nodes are generated on *disjoint* subsets of the dataset, thus obtaining a *tree SPN*, while SPNs can be more generally defined on DAGs. Recently, Rahman and Gogate (2016b) discussed the limitations of using tree structured

SPNs as opposed to DAGs, and addressed the problem of post-processing SPNs obtained with previous methods, by merging similar branches so as to obtain a DAG.

4.2 Parameter learning in SPGMs

Parameter learning in an SPGM $\mathbb{S}$ can be done by interpreting $\mathbb{S}$ as a SPN encoded by message passing (Proposition 10) and directly using any available SPN parameter learning method (these include EM seen in Sect. 4.1 and others (Gens and Domingos 2012)). Hence, we do not discuss this aspect further.

Note however that Sum Node messages (Definition 8) require weight sharing between the SPN Sum Nodes. EM for SPNs with weight tying is addressed in Sect. 4.1.

4.3 Structure learning in SPGMs

Structure learning is an important aspect of tractable inference models, thus it is crucial to provide a structure learning algorithm for SPGMs. Furthermore, it is useful to provide a first example of how the new *connections* between GMs and SPNs can be exploited in practice.

We propose a structure learning algorithm based on the Chow–Liu algorithm for trees (Sect. 2.1). We start by observing that edges with large Mutual Information can be excluded from the Chow–Liu tree, thus losing relevant correlations between variables (Fig. 3b, left). An approach to address this problem, inspired by Meila and Jordan (2000), is to use a mixture of spanning trees such that the k-best edges are included in at least one tree. We anticipate that the trees obtained in this way *share a large part* of their structure (Fig. 3c), hence the mixture can be implemented efficiently as an SPGM.

Algorithm Description We describe next *LearnSPGM*, a procedure to learn structure and parameters of an SPGMs which locally maximizes the weighted log-likelihood (4.2). We use the notation of Sect. 4.1.
The algorithm learns an SPGM $\mathbb{S}$ in three main steps (pseudocode in Algorithm 2). First, $\mathbb{S}$ is initialized to encode the Chow–Liu tree T^* (Fig. 6a)—that is, $\mathcal{T}(\mathbb{S})$ includes a single subtree (Definition 10) $\tau*$ corresponding to T^*. Then, we order each edge $(s, t) \in \overline{\mathcal{E}}$ which was not included in T^* by decreasing mutual information $\mathbb{I}_{st}$, collecting them in the ordered set Q. Finally, we insert each edge $(s, t) \in Q$ in $\mathbb{S}$ with the sub-procedure *InsertEdge* described below, until log-likelihood convergence or a given maximum size of $\mathbb{S}$ is reached.

InsertEdge($\mathbb{S}$, T^*, (s, t)) comprises three steps:

1. Compute the maximum spanning tree over $\overline{\mathcal{G}}$ which includes (s, t), denoted as T_{st}. Finding T_{st} can be done efficiently by first inserting edge (s, t) in T^*, which creates a cycle $\mathcal{C}$ (Fig. 6a), then removing the minimum edge in $\mathcal{C}$ *except* (s, t) (Fig. 6b). The potentials in T_{st} are set as empirical probabilities $\overline{P}_{st}$ according to the Chow–Liu algorithm. Notice that trees T^* and T_{st} have identical structure up to $\mathcal{C}$ and can then be written as $T^* = T^1 \cup \mathcal{C}' \cup T^2$ and $T_{st} = T^1 \cup \mathcal{C}'' \cup T^2$, where $\mathcal{C}' = T^* \cap \mathcal{C}, \mathcal{C}'' = T_{st} \cap \mathcal{C}$.
2. Add T_{st} to the set $\mathcal{T}(\mathbb{S})$ by sharing the structure in common with T^* (T^1 and T^2 above). To do this, first identify the edge (s, t) s.t. $s \in T^1$ and $t \in \mathcal{C}'$ (e.g. (B, C) in Fig. 6b). Then, create a non-Observed Sum Node q, placing it between s and t, unless such node is already present due to previous iterations (see q in Fig. 6c).

   At this point, one of the child branches of q contains $\mathcal{C}' \cup T^2$. We now add a new

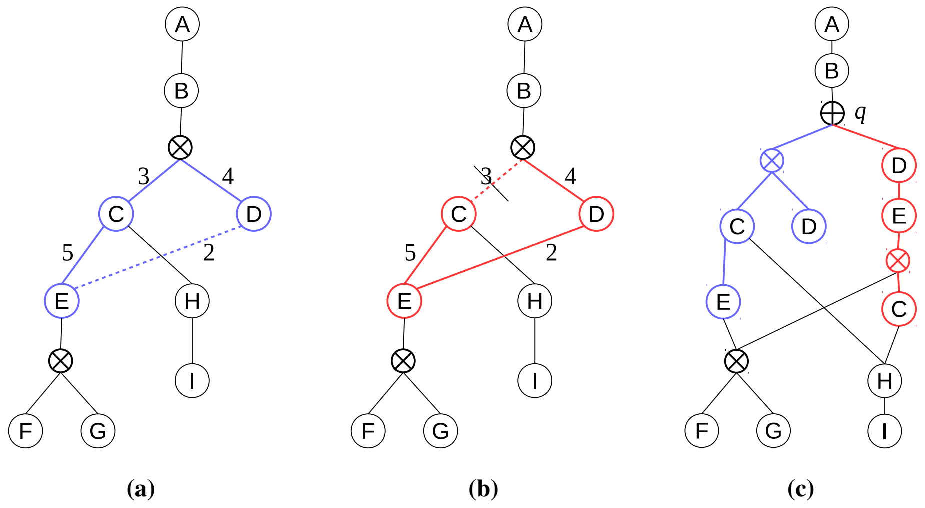

Fig. 6 Graphical representation of edge insertion. For simplicity, we start with an SPGM representing a single MST. **a** Inserting (D, E) creates a cycle (blue). **b** Removing the minimum edge in the cycle (except (D, E)) gives the MST containing (D, E). **c** This MST is inserted into $\mathbb{S}$ sharing the common parts (Color figure online)

child branch containing $\mathcal{C}''$ (Fig. 6c). Finally, we connect nodes in $\mathcal{C}''$ to their descendants in the shared section T^2. The insertion maintains $\mathbb{S}$ valid since the X and Z-scope of any node in $\mathbb{S}$ does not change. Furthermore, inserting $\mathcal{C}''$ in this way we add a subtree representing T_{st} in $\mathcal{T}(\mathbb{S})$, selected by choosing the child of s corresponding to $\mathcal{C}''$.

3. Update the weights of incoming edges of the Sum Node q by using Eq. 4.5 on the set of SPN nodes generated by q with Eq.(3.1a) during the conversion to SPN (see Proposition 9).

Convergence It is possible to prove that the log-likelihood *does not decrease* at each insertion, and thus the initial Chow–Liu tree provides a *lower bound* for the log-likelihood.

Proposition 11 *Each application of InsertEdge does not decrease the log-likelihood of the SPGM (Eq. 4.2).*

Proposition 12 *The Chow Liu tree log-likelihood is a lower bound for the log-likelihood of an SPGM obtained with LearnSPGM.*

Complexity Steps 1 and 2 of *InsertEdge* are inexpensive, only requiring a number of operations linear in the number of edges of the Chow Liu tree T^*. The per iteration complexity of *InsertEdge* is dominated by step 3, in which the computation of weights through Eq. 4.5 requires evaluating the SPGM for the whole dataset. Although evaluation of an SPGM is efficient (see Proposition 1), this can still be too costly for large datasets. We found empirically that assigning weights proportionally to the mutual information of the inserted edge provides a reliable empirical alternative, which we use in experiments.

4.4 Learning mixtures of SPGMs

LearnSPGM is apt at representing data belonging to a single cluster, since (similarly to Chow–Liu trees) the edge weights are computed from a single mutual information matrix

Algorithm 2 *LearnSPGM* $\left(D = \{x^i\}, \{w^i\}\right)$

```
Input: samples $D$, optional sample weights $w$
Output: SPGM $\mathbb{S}$ approx. maximizing $\sum_{i=1}^{N} w_i \ln P(x^i|\theta)$
1: $\mathbb{I} \leftarrow$ Mutual Information of $D$ with weights $w$
2: $T^* \leftarrow$ Chow–Liu tree with connection matrix $\mathbb{I}$
3: $\mathbb{S} \leftarrow$SPGM representing $T^*$
4: $Q \leftarrow$ queue of edges $(s,t) \notin \mathcal{E}$ ordered by decreasing $\mathbb{I}_{st}$
5: repeat
6:     $InsertEdge\left(\mathbb{S}, T^*, Q.pop()\right)$
7:     $AssignWeights$ to the modified Sum Node
8: until convergence or max size reached
```

Algorithm 3 EM for Mixtures of SPGMs($\mathbb{S}_k, \lambda_k, D$)

```
Input: Initial model $P(X) = \sum_k \lambda_k \mathbb{S}_k(X)$, samples $D$
Output: Updated $\mathbb{S}_k(X)$, $\lambda_k$ locally maximizing $\sum_{i=1}^{N} w_i \ln P(x^i|\theta)$
1: repeat
2:     for all $k \in 1...K, i \in 1...N$ do // E-step
3:         $\gamma_k(i) \leftarrow \frac{\lambda_k \mathbb{S}_k(x_i)}{\sum_{k'} \lambda_{k'} \mathbb{S}_{k'}(x_i)}$
4:         $\Gamma_k \leftarrow \sum_{i=1}^{N} \gamma_k(i)$
5:         $w_i^k \leftarrow \gamma_k(i)/\Gamma_k$
6:     for all $k \in 1...K$ do // M-step
7:         $\lambda_k \leftarrow \Gamma_k/N$
8:         $\overline{\mathbb{S}}_k \leftarrow LearnSPGM(D, w^k)$
9:         if WLL($\overline{\mathbb{S}}_k, w^k$) $\geq$ WLL($\mathbb{S}_k, w^k$) then
10:            $\mathbb{S}_k \leftarrow \overline{\mathbb{S}}_k$
11: until convergence
```

estimated on the whole dataset. To model densities with natural clusters one can use mixtures of SPGMs trained with EM. We write a mixture of SPGMs in the form $\sum_{k=1}^{K} \lambda_k P_k(X|\theta^k)$, where each term $P_k(X|\theta^k)$ is the probability distribution encoded by an SPGM $\mathbb{S}_k$ (Eq. 3.3) governed by parameters $\theta^k = \{\mathcal{G}^k, \{P_{st}^k\}, \{W_s^k\}, \{Q_s^k\}\}$ (Definition 7).

EM can be adopted for any mixture model as long as the weighted maximum log-likelihood in the M-step can be solved (Algorithm 1 line 8). In addition, Neal and Hinton (1998) show that EM converges as long as the M-step can be at least partially performed, namely if it possible to find parameters $\overline{\theta}_k$ such that (see Eq. 4.2)

$$\mathrm{WLL}(P_k(X|\overline{\theta}_k), w^k) \geq \mathrm{WLL}(P_k(X|\theta^k), w^k). \tag{4.6}$$

These observation suggest using to use *LearnSPGM* to approximately solve the weighted maximum likelihood problem. However, while *LearnSPGM* ensures that the Chow–Liu tree lower bound always increases, the actual weighted log-likelihood can decrease. To satisfy Eq. (4.6), we employ the simple shortcut of rejecting updates of component $\mathbb{S}_k$ when Eq. (4.6) is not satisfied for θ_{new}^k (Algorithm 3 line 9). Hence, the following trivially follows:

Proposition 13 *The log-likelihood of the training set does not decrease at each iteration of EM for Mixtures of SPGMs (Algorithm 3).*

5 Exploiting connections between GMs and SPNs: example applications

The key contribution of SPGMs consists in being able to exploit jointly properties of SPNs and GMs, with the aim of promoting joint research in the two fields. It is then of utmost importance to demonstrate preliminary ways in which these connections can be exploited in practice.

We present here two applications: Firstly, an application of the mixture of tree based structure learning algorithm of Sect. 5.1 to 20 real world datasets for density estimation, which are currently dominated by SPNs. Secondly, the approximation of a known but intractable GM, chosen in a restricted model class which is easily amenable with SPGMs, which is a new field of application for methods based on SPNs.

In both cases we obtain performances close to or surpassing state-of-the-art models. More interestingly, these results show that the proposed method enables viable and novel applications, which should be fully explored in future research. We propose detailed directions of improvement alongside the description of the experiments.

5.1 Density estimation

We evaluate structure learning algorithm for SPGMs proposed in Sect. 4 by comparing it on 20 real world datasets for density estimation against state-of-the-art methods and baseline methods. The datasets, described in Lowd and Domingos (2012), contain a number of variables ranging from 16 to 1556 and a number of training examples ranging from $1.6K$ to $291K$ (Table 3). All variables are binary. The methods we compare to are in the following denoted with the following abbreviations: MCNets (Mixtures of CutsetNets, Rahman et al. (2014)); ECNet (Ensembles of CutsetNets, Rahman and Gogate (2016a)); MergeSPN, Rahman and Gogate (2016b); ID-SPN, Rooshenas and Lowd (2014); SPN, Gens and Domingos (2013); ACMN, Lowd and Domingos (2012), MT (Mixtures of Trees, Meila and Jordan (2000)); LTM (Latent Tree Models, Choi et al. (2011)).

Methodology We found empirically that the best results were obtained using a two phase procedure: first we run EM updates with LearnSPGM on both structure and parameters until validation log-likelihood convergence, then we fine-tune using EM for SPNs on parameters only until convergence.

We fix the following hyperparameters by grid search on a validation set: maximum number of edge insertions $\{10, 20, 60, 120, 400, 1000, 5000\}$, mixture size $\{5, 8, 10, 20, 100, 200, 400\}$, uniform prior $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-9}\}$ (used for mutual information and sum weights). In all our experiments, SPGMs started to overfit at around 200 mixture models; the best models based on validation LL have between 5 and 100 components. Hence, we stopped our search at 400.

LearnSPGM was implemented in C++ and is available at the following URL: https://github.com/ocarinamat/SumProductGraphMod, which also contains the setup to reproduce this experiment. The average learning time per dataset is 42 min on an Intel Core i5-4570 CPU with 16 GB RAM. Inference takes up to one minute on the largest dataset.

Results analysis Test set log-likelihood results averaged over 5 random parameters initializations are shown in Table 3.[8] Our method performs best between all compared models in

[8] Standard deviation results can be found in Table 6.

Table 3 Dataset structure and test set log likelihood comparison

Dataset	#vars	#train	SPGM	ECNet	MergeSPN	MCNet	Vergari&Al.	ID-SPN	ACMN	SPN	MT	LTM
NLTCS	16	16,181	−**5.99**	− 6.00	− 6.00	− 6.00	− 6.14	− 6.02	− 6.00	− 6.11	− 6.01	− 6.49
MSNBC	17	291,326	− **6.03**	− 6.05	− 6.10	− 6.04	− **6.03**	− 6.04	− 6.04	− 6.11	− 6.07	− 6.52
KDDCup2K	64	180,092	− 2.13	− 2.13	− **2.12**	− **2.12**	− 2.12	− 2.13	− 2.17	− 2.18	− 2.13	− 2.18
Plants	69	17,412	− 12.71	− 12.19	− **12.03**	− 12.74	− 12.09	− 12.54	− 12.80	− 12.98	− 12.95	− 16.39
Audio	100	15,000	− 39.90	− 39.67	− **39.49**	− 39.73	− 39.62	− 39.79	− 40.32	− 40.50	− 40.08	− 41.90
Jester	100	9000	− 52.83	− **52.44**	− 52.47	− 52.57	− 52.87	− 52.86	− 53.31	− 53.48	− 53.08	− 55.17
Netflix	100	15,000	− 56.42	− 56.13	− **55.84**	− 56.32	− 56.37	− 56.36	− 57.22	− 57.33	− 56.74	− 58.53
Accidents	111	12,758	− **26.89**	− 29.25	− 29.32	− 29.96	− 26.98	− 26.98	− 27.11	− 30.04	− 29.63	− 33.05
Retail	135	22,041	− 10.83	− **10.78**	− 10.82	− 10.82	− 10.85	− 10.85	− 10.88	− 11.04	− 10.83	− 10.92
Pumsb-star	163	12,262	− **22.15**	− 23.34	− 23.67	− 24.18	− 22.40	− 22.40	− 23.55	− 24.78	− 23.71	− 31.32
DNA	180	1600	− **79.88**	− 80.66	− 80.89	− 85.82	− 81.21	− 81.21	− 80.03	− 82.52	− 85.14	− 87.60
Kosarek	190	33,375	− 10.57	− **10.54**	− 10.55	− 10.58	− 10.60	− 10.60	− 10.84	− 10.99	− 10.62	− 10.87
MSWeb	294	29,441	− 9.81	− 9.70	− 9.76	− 9.79	− **9.61**	− 9.73	− 9.77	− 10.25	− 9.85	− 10.21
Book	500	8700	− 34.18	− **33.78**	− 34.25	− 33.96	− 33.81	− 34.14	− 36.56	− 35.89	− 34.63	− 34.22
EachMovie	500	4524	− 54.08	− 51.14	− 50.72	− 51.39	− **50.26**	− 51.51	− 55.80	− 52.49	− 54.60	†
WebKB	839	2803	− 154.55	− 150.10	− 150.04	− 153.22	− **149.85**	− 151.84	− 159.13	− 158.20	− 156.86	− 156.84
Reuters-52	889	6532	− 85.24	− 82.19	− **80.66**	− 86.11	− 81.54	− 83.35	− 90.23	− 85.07	− 85.90	− 91.23
20Newsgrp.	910	11,293	− 153.69	− 151.75	− **150.80**	− 151.29	†	− 151.47	− 161.13	− 155.93	− 154.24	− 156.77
BBC	1058	1670	− 255.22	− 236.82	− **233.26**	− 250.58	− **226.35**	− 248.93	− 257.10	− 250.69	− 261.84	− 255.76
Ad	1556	2461	− 14.30	− 14.36	− 14.34	− 16.68	− 13.59	− 19.00	− **16.53**	− 19.73	− 16.02	†

The symbol [bold dagger symbol here] denotes that the experiment was not completed due to excessive memory or computation time requirements

5 cases out of the 20 tested datasets. For comparison, ECNets are best in 4 cases, MergeSPN in 7, MCnets in 1, Vergari et al. (2015) in 5.

Remarkably, these results were obtained with our relatively simple structure algorithm based on mixtures of trees, and they could be *straightforwardly improved* by applying the simplification and regularization techniques discussed in Vergari et al. (2015) on the SPN encoded by our model. Further possible improvements consist in using the ensemble learning as in ECNets rather than EM, as discussed in MCnets, or regularizing the encoded SPN by merging branches as in MergeSPN.

Improving the empirical performances is out of the focus of this paper, which consists of representation properties of the model. Hence, we leave these extensions to future applied work.

Finally, notice that SPGMs—that are large mixtures of trees—*always* outperform both standard mixture of trees and Latent Tree Models (Choi et al. 2011). We hypothesize that sharing tree structure *helps preventing overfitting*, which is the key limiting factor in these models (cf. also the analysis in Section).

5.2 Approximating a layered directed graphical model

In this section we use SPGMs for approximating a *known* but intractable graphical model $\mathcal{G}$ by a large mixture of its spanning trees. This is a novel approach since, to the best of our knowledge, SPNs have never been used to approximate a given GM.

This application is motivated by the observation that mixtures of spanning trees have been used extensively to approximate intractable graphs (see e.g. Meila and Jordan 2000; Bach and Jordan 2001; Pletscher et al. 2009). A further example is the success of mixtures of chain graphs for approximate inference in the Tree-Reweighted Belief Propagation (TRW) framework (see Kolmogorov 2006). SPGMs seem to be a very promising architecture to apply to this framework due to their ability to *efficiently* represent *very large* mixtures of trees due to shared parts and to encode knowledge about the original graph (as we will see).

For simplicity, we concentrate on a class of directed GMs denoted layered GMs, for which it is easy to obtain a mixture of spanning trees amenable within the SPGM framework. Extending such an approach to arbitrary GMs is a challenging but very interesting research problem outside of the scope of this paper, and it should be the subject of further research.

We define a layered GM as a directed GM composed by successive layers of variables, where variables in one layer connect only to variables in the next or previous one. This class of distributions is relevant in applications: it includes Factorial Hidden Markov Models, Multiscale Quadtrees (Wainwright and Jordan (2008)) and Deep Belief Networks (Hinton and Osindero (2006)). It is straightforward to show that inference cost in layered GMs is worst case exponential in the layer size and it is therefore intractable.

A spanning tree can be trivially taken from a layered distribution by allowing a single variable to have children at each layer (Fig. 7b). It is easy to see that if two trees T_1 and T_2 taken in this way differ only by the choice of which variable has children, then their structure is largely shared.

The mixture of many spanning trees with this structure can be modeled with the SPGM shown in Fig. 8: notice that any subtree in this model corresponds to a tree in the form of Fig. 7, right, hence the SPGM encodes the mixture of all such trees (Proposition 7).

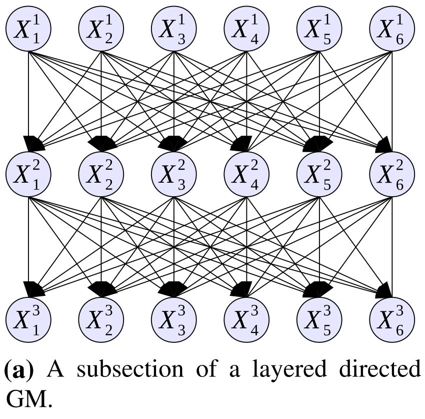
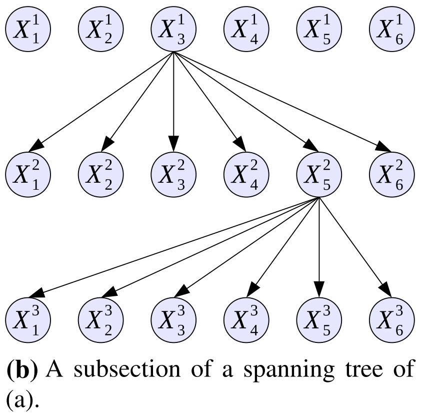
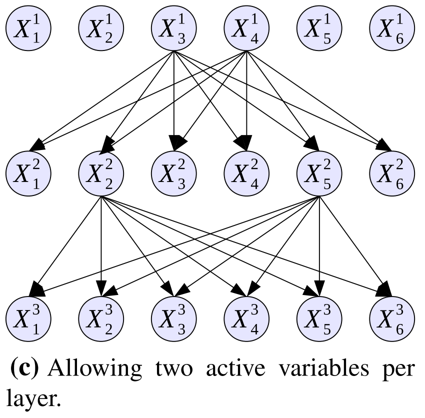
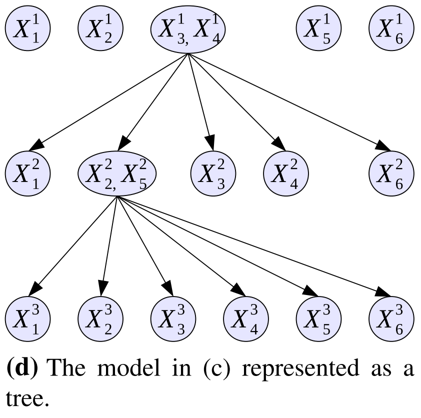

Fig. 7 A mixture of spanning trees with shared subparts obtained from a layered directed graphical model as described in Sect. 5.2. Variable X_k^l denotes the k-th variable at layer l

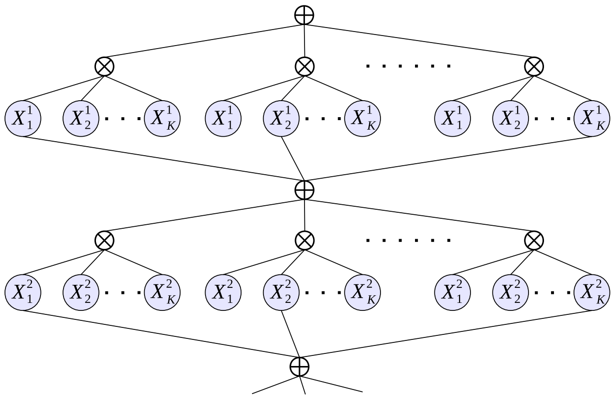

Fig. 8 First two layers of an SPGM encoding a mixture of spanning trees in a layered model with K variables per layer. X_k^l denotes the k-th variable at layer l

Table 4 Log Likelihood values for the experiment in Sect. 5.2, for different models (rows) and for different number of trees in the mixture (columns, only defined when the model is a mixture of trees)

Model/#trees	1	10	20	40	2^{10}	4^{10}	6^{10}	8^{10}
Chow–Liu tree, Train	−27.7							
Chow–Liu tree, Test	− 30.2							
Mix. Tree, Train		− 20.6	− 17.5	− 14.2				
Mix. Tree, Test		− 23.4	− 23.7	− 24.2				
SPGM, 1 act. var. per layer, Train	− 34.1				− 24.7	− 18.4	− 16.3	
SPGM, 1 act. var. per layer, Test	− 34.5				− 25.7	− 20.0	− 17.8	
SPGM, 4 act. vars. per layer, Train	− 31.5				− 21.2	− 16.5	− 15.4	− 14.8
SPGM, 4 act. vars. per layer, Test	− 32.1				− 22.6	− 18.1	− 17.1	− 16.5
SPGM, 6 act. vars. per layer, Train	− 20.7				− 14.9	− 13.6	**− 13.3**	− 13.8
SPGM, 6 act. vars. per layer, Test	− 22.1				− 16.9	− 15.6	**− 15.1**	− 15.9

Unavailable result for classical mixture of trees are due either to excessive run time with more than 2^{10} components. SPGMs can deal with very large mixtures with much less overfitting than standard mixture models. Missing results for SPGMs are due to the fact that in SPGMs the number of trees cannot be fixed at arbitrary intervals

Table 5 Comparison of Log Likelihood results for the experiment in Sect. 5.2 between SPGMs and two recent SPN learning algorithms (see Table 3)

	SPGM	MergeSPN	ID-SPN	ACMN	SPN
Train	– **13.3**	– 17.7	– 19.7	– 21.7	– 22.1
Test	– **15.1**	– 19.3	– 21.6	– 23.9	– 24.2

We compare against methods for which an implementation is publicly available

In addition, we can also allow more than one variable in a layer to have children (Fig. 7c). This can be done by creating a clique by merging variables having children into a single composite variable: e.g., if two variables A and B have children at a certain layer, then we create a Vnode associated with a new variable $\{A, B\}$ with values in $\Delta(A) \times \Delta(B)$, which merges the individual variables (Fig. 7d). The resulting model can be encoded by an SPGM where nodes represent cliques, which is a straightforward extension of standard SPGMs.

The resulting SPGM efficiently encodes a *very large* mixture of trees with shared parts: if the model contain L layers, and there are C choices of variables with children at each layer, then the number of subtrees grows exponentially as C^L. Crucially, despite the mixture being very large inference remains *tractable* due to sharing of tree subparts (Proposition 1).

Empirical Evaluation We tested the SPGM obtained as described above on a layered mixture model with 10 layers, each containing 6 binary variables. We took a set of samples using ancestral sampling from the original GM (ancestor sampling is always tractable in directed GMs) and divided them into fixed training and test sets. We then maximized the training set Log Likelihood on our SPGM via EM for SPNs, as described in Peharz (2015). We always ran the algorithm until convergence (no early stopping is used).

We test SPGM models using a different number of choices of active variables per layer (i.e. the number of sum node children), which results in an increasingly larger number of trees in the resulting mixture model. Choosing from 1 to 8 possible active variables per layer, the mixture size ranges from 1 to 8^{10}, while always being tractable: inference takes about 0.2 seconds for the largest mixture model on a Intel Core i5-4570 CPU, for our unoptimized MATLAB implementation. We also test using different numbers of variables with children at each layer (1,2 and 4).

We first compared against methods based on tree GMs (Table 4), namely optimal spanning trees (Chow and Liu 1968), mixture of trees trained with EM (Meila and Jordan 2000), and Latent Tree Models (Choi et al. 2011). We report separate results depending on the number of trees in the mixture, in case a mixture is used. Then, we compared against state-of-the-art density estimation methods for SPNs (Table 5)—please refer to Sect. 5.1 for the used abbreviations.

SPGMs amply outperform competing methods in terms of test set LL. In particular, they do not seem to suffer from the overfitting problem that plagues mixtures of tree even for moderately large mixture sizes: we obtained our best results for mixture of 6^{10} components.

This is a strong indicator that the ability of SPGMs to embed knowledge about the graph structure into the model (despite only use the connectivity and not the weights), which is lacking in SPNs, enables a better estimation of the underlying distribution.

The tasks of deriving an SPGM able to approximate a given, generic graph, and the task of providing an algorithm that directly optimizes the parameters by e.g. direct minimization of the Kullback-Leibler divergence between the two graphs, are challenging research problems beyond the scope of this paper. However, they consitute a relevant direction of future research where the new connections between SPNs and GMs could be exploited. In particular, a promising research direction consists in using SPGMs to replace the simple mixtures of chain graphs used for approximate inference in the Tree-Reweighted Belief Propagation (TRW) framework (see Kolmogorov 2006) by extending the guaranteed improvement of the approximation bounds to SPGMs.

6 Conclusions and future work

We introduced sum–product graphical model (SPGM), a new architecture bridging sum–product networks (SPNs) and Graphical Models (GMs) by inheriting the expressivity of SPNs and the high level probabilistic semantics of GMs.

We showed that SPGMs enable one to *jointly exploit* properties of the two families of models. These new connections were exploited in two preliminary applications: first, a structure learning algorithm extending the mixture of trees approach and obtaining results comparable with the state of the art in density estimation, and secondly, by a procedure to approximate a known but untractable GM chosen from the class of layered models, which is a novel application area for SPNs.

The theoretical and practical results demonstrate that jointly exploiting properties of SPNs and GMs is an interesting direction of future research. A particularly interesting consists in the generalization of the approximation of intractable GMs explored in the preliminary application, by using very large but tractable mixture models implemented by SPGMs.

Acknowledgements Support of the German Science Foundation, Grant GRK 1653, is gratefully acknowledged.

A Proofs

A.1 Proposition 1

Every message is evaluated exactly once according to Definition 8. Each of the $|\mathcal{V}|$ nodes sends at most M messages (one for each Vparent), and each message has size $|\Delta_{max}|^2$ (one value per every state of sending and receiving node). □

A.2 Proposition 2

Only Product Nodes in $\tau \in \mathcal{T}(\mathbb{S})$ can have multiple children, since Vnodes have a single child by Definition 7, case 5, and Sum Nodes have a single child in τ by Definition 10. Children of Product Nodes have disjoint graphs by Definition 7, case 7. Therefore τ contains no cycles. A rooted graph with no cycles is a tree. □

A.3 Proposition 3.

The proposition can be proven by inspection, considering an SPGM $\mathbb{S}$ built by stacking units as follows: Sum Node s_1 (the root of $\mathbb{S}$) is associated with observed variable Z_1 and has as children the set of Vnodes $V_1 = \{v_{1,1}, v_{1,2}, \dots, v_{1,M}\}$. All the nodes in V_1 have a common single child, which is the sum node $s2$. In turn, s_2 has the same structure as s_1, having Vnode children $V_1 = \{v_{2,1}, v_{2,2}, \dots, v_{2,M}\}$ which are connected to a single child $s3$, and so forth for Sum Nodes $s_3, s_4, \ldots, s_K$. The number of edges in $\mathbb{S}$ is $2MK$. On the other hand, a different subtree can be obtained for each choice of active child at each sum node. There is a combinatorial number of such choices, thus the number of different subtrees is K^M. □

A.4 Proposition 4

A subtree is obtained with Definition 10 by iteratively choosing only one child of each sum node. However, each child of a sum node has the same scope, due to Definition 7, condition 6. Hence, taking only one child one obtains the same scope as taking all the children. □

A.5 Proposition 5.

Consider a subtree $\tau \in \mathcal{T}(\mathbb{S})$ as in Definition 10.

Let us first consider messages generated by sum nodes s. Considering that s has only one child $ch(s)$ in τ (for Definition 10), corresponding to indicator variable $[Z_s]_{k_s}$, and applying Eqs. 3.1, the messages for observed and unobserved sum nodes are as follows:

$$\mu_{st;j} = [Z_s]_{k_s} Q_s(k_s)\mu_{ch(s),t;j}, \text{ Observed Sum Node,} \tag{A.1}$$

$$\mu_{st;j} = W_s(k_s)\mu_{ch(s)_k,t;j}, \quad \text{Unobserved Sum Node.} \tag{A.2}$$

Hence sum messages contribute only by introducing a multiplicative term $[Z_s]_{k_s} Q_s(k_s)$ or $W_s(k_s)$. Now, all variables in the set Z appear in τ (Proposition 4). Hence, the sum nodes together contribute with the following multiplicative term:

$$\prod_{s\in O(\tau)} Q_s(k_{s,\tau}) \prod_{s\in U(\tau)} W_s(k_{s,\tau}) \prod_{[Z_s]_{k_s}\in z_\tau} [Z_s]_{k_s} = \lambda_\tau \prod_{[Z_s]_{k_s}\in z_\tau} [Z_s]_{k_s}. \tag{A.3}$$

From this it follows that message passing in τ is equivalent to message passing in an SPGM $\overline{\tau}$ obtained by *discarding* all the sum nodes from τ, followed by multiplying the resulting messages for Eq. A.3.

Consecutive Product Nodes in $\overline{\tau}$ can be merged by adding the respective children to the parent Product Node. In addition, between each sequence Vnode-Vnode in $\overline{\tau}$ we can insert a product node with a single child. Thus, we can take $\overline{\tau}$ as containing only Vnodes and Product Nodes, such that the children of Product Nodes are Vnodes. Putting together Eqs. 3.1c and 3.1d, the message passed by each Vnode t to $s \in vpa(t)$ is:

$$\mu_{t\to s;j} = \sum_{k\in\Delta(X_t)} P_{s,t}(k|j)\,[X_t]_k \prod_{q\in ch(ch(c))} \mu_{q\to t;j}. \tag{A.4}$$

Notice that the input messages are generated from the grandchildren of t, that is the children of the Product Node child of t. This corresponds to the message passed by variables in a tree graphical model obtained by removing the Product Nodes from $\overline{\tau}$ and attaching the children of Product Nodes (here, elements $q \in ch(ch(c))$) as children of their parent Vnode (here, t), which can be seen by noticing the equivalence to Eq. (2.4). This tree GM can be immediately identified as $P_\tau(X)$. Note also that if the root of $\overline{\tau}$ is a Product Node, then $P_\tau(X)$ represents a forest of trees, one for each child of the root Product Node.

The proof is concluded reintroducing the multiplicative factor in Eq. A.3 discarded when passing from τ to $\overline{\tau}$. □

A.6 Proposition 6.

First, consider an SPGM $\mathbb{S}(X, Z)$ defined over $\mathcal{G} = \mathcal{V}, \mathcal{E}$. Let us take a node $t \in \mathcal{V}$ and let $\mathbb{S}_t(X^t, Z^t)$ denote the sub-SPGM rooted in t. Suppose that $\mathbb{S}_t$ satisfies the proposition and hence it can be written as:

$$\mathbb{S}_t = \sum_{\tau_t \in \mathcal{T}(\mathbb{S}_t)} \tau_t. \tag{A.5}$$

Let us consider messages sent from node t to a Vparent $s \in vpa(t)$ (Definition 6), and note that if form (A.5) is satisfied then messages take the form

$$\mu_{t \to s;j} = \sum_{\tau_t \in \mathcal{T}(\mathbb{S}_t)} \mu_{root(\tau_t) \to s;j}, \tag{A.6}$$

where $root(\tau_t))$ denotes the message sent from the root of subtree τ_t to s. Vice versa, if form (A.6) is satisfied then $\mathbb{S}_t$ can be written as Eq. (A.5). This extends also to the case $vpa(t) = \emptyset$, in which messages are sent to a fictitious $root$ node (Definition 8).

The proposition can now be proved by induction. First, the base case: sub-SPGMs rooted at Vnodes leaves trivially assume form (A.5), and hence also the form (A.6). Then, the inductive step: Lemma (1) can be applied recursively for all nodes from the leaves to the root. Hence, $\mathbb{S}$ assumes the form of Eq. (A.6) and thus also of (A.5). □

Lemma 1 *Consider a node $t \in \mathcal{V}$. Suppose that the messages sent from the children of node $t \in \mathcal{V}$ are in the form* (A.6). *Then, each message sent from s also assumes the form* A.6.

Let us suppose, for simplicity, that t has only two children p and q—the extension to the general case is straightforward. We distinguish the three cases of t being a Sum Node, a Product Node or a Vnode.

– Suppose t is an Observed Sum Node with weights $[Q_t(0), Q_t(1)]$ and indicator variables $[Z_t]_0$ and $[Z_t]_1$ associated with each child. By Eq. (A.2), the children send messages to their Vparent s, and since input message are in the form (A.6), t sends the following message:

$$\mu_{t \to s;j} = Q_t(0)[Z_t]_0 \sum_{\tau_p \in \mathcal{T}(\mathbb{S}_p)} \mu_{root(\tau_p) \to s;j} + Q_t(1)[Z_t]_1 \sum_{\tau_q \in \mathcal{T}(\mathbb{S}_q)} \mu_{root(\tau_q) \to s;j}.$$

  This is again in the form A.6. This can be seen because each term in the sum is in the form $Q_t(0)[Z_t]_0\mathcal{T}(\mathbb{S}_p)$ which corresponds to message passed from the subtree $\tau_t \in \mathcal{T}(\mathbb{S}_t)$ obtained choosing child p (in this case). It is easy to see that terms corresponding to each subtrees of t are present.
– If t is an Unobserved Sum Node the discussion is identical to the point above.
– If t is a Product Node, then the children send messages to their Vparent s, and since input message are in the form (A.6), t sends the following message:

$$\begin{aligned}\mu_{t \to s;j} &= \left(\sum_{\tau_p \in \mathcal{T}(\mathbb{S}_p)} \mu_{root(\tau_p) \to s;j} \right) \left(\sum_{\tau_q \in \mathcal{T}(\mathbb{S}_q)} \mu_{root(\tau_q) \to s;j} \right) \\ &= \sum_{(\tau_p, \tau_q) \in \mathcal{T}(\mathbb{S}_p) \times \mathcal{T}(\mathbb{S}_q)} \mu_{root(\tau_p) \to s;j} \mu_{root(\tau_q) \to s;j}.\end{aligned}$$

  Now, $\mu_{root(\tau_p) \to s;j} \mu_{root(\tau_q) \to s;j}$ can be seen as the message generated by a particular subtrees of t, and thus node $\mu_{t \to s;j}$ is in the form (A.6).

– If t is a Vnode, then it has a *single child* p by Definition 7, sending messages to t. Thus, the input message is in the form (A.6), and t sends the following message:

$$\mu_{t\to s;j} = \sum_{k\in\Delta(X_t)} P_{s,t}(k|j)\,[X_t]_k \sum_{\tau_p\in\mathcal{T}(\mathbb{S}_p)} \mu_{root(\tau_p)\to t;k}$$
$$= \sum_{\tau_p\in\mathcal{T}(\mathbb{S}_p)} \sum_{k\in\Delta(X_t)} P_{s,t}(k|j)\,[X_t]_k \mu_{root(\tau_p)\to t;k}.$$

Let us analyze a term of the sum for each fixed τ_p. Such term corresponds to the root message of an SPGM $\overline{\mathbb{S}}$ obtained taking τ_p and adding the Vnode t as parent of p (due to Eq. (3.1d)). But $\overline{\mathbb{S}}$ obtained in this form is a subtree of t by Definition 10. Therefore, taking the sum $\sum_{\tau_p\in\mathcal{T}(\mathbb{S}_p)}$ corresponds to summing over messages sent by all subtrees of t, in the form (A.6). □

A.7 Proposition 7

Due to Propositions 5 and 6, the evaluation of $\mathbb{S}$ corresponds to performing message passing (Eq. 2.4) with the mixture distribution

$$\sum_{\tau\in\mathcal{T}(\mathbb{S})} \left(\prod_{[Z_s]_j\in z_\tau} [Z_s]_j \right) \lambda_\tau P_\tau(X).$$

We now note that the term $\left(\prod_{[Z_s]_j\in z_\tau}[Z_s]_j\right)$ attains the value 1 only for the subset of trees compatible with the assignment $Y_z = y_z$ and 0 otherwise (since some indicator in the product is 0), that is for subtrees in the set $\mathcal{T}(\mathbb{S}|Y_z = y_z)$ (Definition 12). Therefore, the sum can be rewritten as $\sum_{\tau\in\mathcal{T}(\mathbb{S}|Z)}$, and the indicator variables (with value 1) can be removed, which results in (3.2). The proof is concluded noting that computing message passing in a mixture of trees with assignment y_x, y_z corresponds to computing marginals $P(Y_x = y_x, Y_z = y_z)$ in the corresponding distribution (Sect. 2.1), hence Eq. (3.7) follows. □

A.8 Proposition 8

Follows from results in Meila and Jordan (2000), section 3. In a mixture of trees, conditional independence of A, B given C holds *iff* for every tree in the mixture the path between A and B contains C. If D-separation holds for all paths in $\pi(B, A|z)$ then it holds for all the subtrees compatible with $Z = z$. But $P(X, Z)$ is the mixture of all subtrees compatible with assignment z (Eq. (3.2)). Hence the result follows. □

A.9 Proposition 9

The proposition can be proven by induction showing that if input messages represent valid SPNs, then also the output SPGM messages are SPNs (see Fig. 5). Formally, it is sufficient to notice that the hypothesis of Lemma 2 below is trivially true for leaf messages (inductive hypothesis), hence the lemma can be inductively applied for all nodes up to the root. □

Lemma 2 *Consider nodes $t \in \mathcal{V}$ and $s \in vpa(t)$, and let X^t and Z^t respectively denote the X scope and Z scope of node $t \in \mathcal{V}$. If the message $\mu_{t\to s;j}$ encodes a SPN $S_{t;j}(X^t, Z^t)$, then the message $\mu_{s\to r;i}$ sent from node s to any node $r \in vpa(s)$ also implements a SPN*

$S_{r;i}(X^s, Z^s)$. *This also holds when* $vpa(s) = \emptyset$, *where* r *is simply replaced by the fictitious root node (Definition 8).*

Proof Consider the message passing Eq. 3.1, referring to Fig. 5 for visualization. We distinguish the case of node t being a Sum, Product and Vnode.

- Sum Nodes (Observed and non-Observed). First, we note that every child of a Sum Node has the same scope X^s, Z^s (for Definition 7 condition 6). Employing the hypothesis, the generated message is:

$$\mu_{t\to s;j} = \sum\nolimits_{k=1}^{|ch(t)|} [Z_t]_k Q_t(k) S_{t;j}\left(X^t, Z^t\right), t \text{ is a Sum Node, Observed,} \quad \text{(A.7)}$$

$$\mu_{t\to s;j} = \sum\nolimits_{k=1}^{|ch(t)|} W_t(k) S_{t;j}\left(X^t, Z^t\right), t \text{ is a Sum Node, not-Observed.} \quad \text{(A.8)}$$

  It is straightforward to see that both equations represent valid SPNs: the sum $\sum_{k=1}^{|ch(t)|}$ becomes the root SPN Sum Node with non-negative weights $Q_t(k)$ and $W_t(k)$ respectively, and its children are SPNs having the same scope (in the form $[Z_t]_k \otimes S_j\left(X^t, Z^t\right)$ for Eq. A.7 and in the form $S_j\left(X^t, Z^t\right)$ for Eq. A.8). Note that $Z^t \cap Z_t = \emptyset$ for Definition 7 condition 6, therefore condition 3 in Definition 3 is satisfied.
- Product Nodes. Applying the hypothesis to input messages, Eq. 3.1c becomes:

$$\mu_{t\to s;j} = \prod_{q\in ch(t)} S_{q;j}\left(X^q, Z^q\right).$$

  This represents a valid SPN with a Product Node as root since the children node's scopes are disjoint (for Definition 7 condition 7), and thus condition 2 in Definition 3 is satisfied.
- Vnodes. Applying the hypothesis to input messages, Eq. 3.1d becomes:

$$\mu_{t\to s;j} = \sum_{k\in\Delta(X_t)} P_{s,t}\left(k|j\right)[X_t]_k S_{ch(t);k}\left(X^{ch(t)}, Z^{ch(t)}\right).$$

  This represents a valid SPN with a Sum Node $\sum_{k\in\Delta(X_t)}$ as root. To see this, first note that terms $P_{s,t}\left(k|j\right)$ can be interpreted as weights. The Sum Node is connected to children SPNs in the form $[X_t]_k \otimes S_{ch(t);k}\left(X^{ch(t)}, Z^{ch(t)}\right)$, which are valid SPNs since $X^i \cap X_s = \emptyset$ and thus condition 2 in Definition 3 is satisfied. In addition all child SPNs have the same scope for Definition 7 condition 6, hence condition 3 in Definition 3 is satisfied for the Sum Node.

□

A.10 Proposition 10

Proof Sketch Firstly, due to Proposition 9, SPNs are at least as expressive as SPGMs since they encode a SPN via message passing. Secondly, any SPN S can be transformed into an equivalent SPGM $\mathbb{S}$ by simply replacing the indicator variable $[A]_a$ in SPN leaves with Vnodes s associated with variable A and unary probability $P_s(A) = [A]_a$ (notice that pairwise probabilities do not appear). It is immediate to see that by doing so all the conditions of Definition 7 are satisfied, and evaluating $\mathbb{S}$ with message passing yields S. As a consequence, SPGMs are at least as expressive as SPNs.

Table 6 Standard deviations corresponding to the SPGM results of Table 3

NLTCS	MSNBC	KDDCup2K	Plants	Audio	Jester	Netflix	Accidents	Retail	Pumsb-star
0.0042	0.0076	0.0037	0.0802	0.1326	0.3125	0.3740	0.1626	0.0752	0.0234
DNA	Kosarek	MSWeb	Book	EachMovie	WebKB	Reuters-52	20Newsgrp.	BBC	Ad
0.6155	0.1002	0.0084	0.3242	0.4526	1.6230	1.3238	3.5572	4.1047	0.0425

A.11 Proposition 11

InsertEdge adds the branch $\mathcal{C}' \cup T^2$ to Sum Node q, hence it adds a new incident edge and a corresponding weight to q. We now note that computing weight values using Eq. (4.5) (step 3 in *InsertEdge* above) allows one to find the optimal weights of edges incoming to q considering the other edges fixed, as shown e.g. in Desana and Schnörr (2016).[9] Since the new locally optimal solution includes the weight configuration of the previous iteration, which is simply obtained by setting the new edge weight to 0 and keeping the remaining weights, the log-likelihood does not increase at each iteration. □

A.12 Proposition 12

Follows immediately from Proposition 11, noting that the SPGM is initialized as the Chow Liu tree T^* (Table 6). □

References

Amer, M., & Todorovic, S. (2015). Sum product networks for activity recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*, 800–813.

Bacchus, F., Dalmao, S., & Pitassi, T. (2012). Value elimination: Bayesian inference via backtracking search. *CoRR*, arXiv:1212.2452.

Bach, F. R., & Jordan, M. I. (2001). Thin junction trees. In *Advances in neural information processing systems*, vol 14. MIT Press, pp. 569–576.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. pp. 115–123.

Cheng, W.-C., Kok, S., Pham, H. V., Chieu, H. L., & Chai, K. M. (2014). Language modeling with sum–product networks. In*Annual conference of the international speech communication association 15 (INTERSPEECH 2014).*

Chickering, D. M., Heckerman, D., & Meek, C. (2013). A Bayesian approach to learning Bayesian networks with local structure. *CoRR*, arXiv:1302.1528.

Choi, M. J., Tan, V. Y. F., Anandkumar, A., & Willsky, A. S. (2011). Learning latent tree graphical models. *Journal of Machine Learning Research*, *12*, 1771–1812.

Chow, C. I., & Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, *14*, 462–467.

Conaty, D., Mauá, D. D., & de Campos, C. P. (2017). Approximation complexity of maximum a posteriori inference in sum–product networks. In Elidan, G., & Kersting, K. (Eds), *Proceedings of the thirty-third conference on uncertainty in artificial intelligence*. AUAI Press, pp. 322–331.

Cowell, R., Dawid, A., Lauritzen, S., & Spiegelhalter, D. (2003). *Probabilistic networks and expert systems*. Berlin: Springer.

Darwiche, A. (2002). A logical approach to factoring belief networks. In D. Fensel, F. Giunchiglia, D. L. McGuinness, & M.-A. Williams (Eds.), *KR* (pp. 409–420). Burlington: Morgan Kaufmann.

[9] However, this is not the globally optimal solution when the other parameters are free.

Darwiche, A. (2003). A differential approach to inference in Bayesian networks. *Journal of the ACM*, *50*(3), 280–305.

Dechter, R., & Mateescu, R. (2007). AND/OR search spaces for graphical models. *Artificial Intelligence*, *171*(2–3), 73–106.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, *39*(1), 1–38.

Desana, M. & Schnörr, C. (2016). Expectation maximization for sum–product networks as exponential family mixture models. *CoRR*, arXiv:1604.07243.

Diestel, R. (2006). *Graph theory* (3rd ed.). Berlin: Springer.

Fridman, A. (2003). Mixed Markov models. *PNAS*, *100*, 8092–8096.

Gens, R., & Domingos, P. (2012). Discriminative learning of sum–product networks. In *NIPS*, pp. 3248–3256.

Gens, R., & Domingos, P. (2013). Learning the structure of sum–product networks. *ICML*, *3*, 873–880.

Gogate, V., Webb, W., & Domingos, P. (2010). Learning efficient Markov networks. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, & A. Culotta (Eds.), *Advances in neural information processing systems* (Vol. 23, pp. 748–756). Red Hook: Curran Associates Inc.

Hinton, G. E., & Osindero, S. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 2006.

Jordan, M. I. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*, 181–214.

Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(10), 1568–1583.

Kolmogorov, V., & Zabih, R. (2004). What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(2), 147–159.

Lowd, D., & Domingos, P. (2012). Learning arithmetic circuits. *CoRR*, arXiv:1206.3271.

Mcallester, D., Collins, M., & Pereira, F. (2004). Case-factor diagrams for structured probabilistic modeling. In *Proceedings of the twentieth conference on uncertainty in artificial intelligence (UAI 04)*, pp. 382–391.

Mei, J., Jiang, Y., & Tu, K. (2018). Maximum a posteriori inference in sum–product networks.

Meila, M., & Jordan, M. I. (2000). Learning with mixtures of trees. *Journal of Machine Learning Research*, *1*, 1–48.

Minka, T., & Winn, J. (2009). Gates. In *Advances in neural information processing systems 21*.

Neal, R., & Hinton, G. E. (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*. Kluwer Academic Publishers, pp. 355–368.

Peharz, R. (2015). Foundations of sum–product networks for probabilistic modeling (PhD thesis). *Researchgate:273000973*.

Peharz, R., Gens, R., Pernkopf, F., & Domingos, P. M. (2016). On the latent variable interpretation in sum–product networks. *CoRR*, arXiv:1601.06180.

Pletscher, P., Ong, C. S., & Buhmann, J. M. (2009). Spanning tree approximations for conditional random fields. In Dyk, D. A. V., & Welling, M. (Eds.), *AISTATS*, volume 5 of *JMLR proceedings*, pp. 408–415. JMLR.org.

Poole, D. L., & Zhang, N. L. (2011). Exploiting contextual independence in probabilistic inference. *CoRR*, arXiv:1106.4864.

Poon, H., & Domingos, P. (2011). Sum–product networks: A new deep architecture. In *UAI 2011, Proceedings of the twenty-seventh conference on uncertainty in artificial intelligence, Barcelona, Spain, July 14–17, 2011*, pp. 337–346.

Rahman, T.m & Gogate, V. (2016a). Learning ensembles of cutset networks. In *Proceedings of the thirtieth AAAI conference on artificial intelligence*, February 12–17, 2016, Phoenix, AZ, USA, pp. 3301–3307.

Rahman, T., & Gogate, V. (2016b). Merging strategies for sum–product networks: From trees to graphs. In *Proceedings of the thirty-second conference on uncertainty in artificial intelligence, UAI 2016*, June 25–29, 2016, New York City, NY, USA.

Rahman, T., Kothalkar, P., & Gogate, V. (2014). Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow–Liu trees. In *Machine learning and knowledge discovery in databases—European conference, ECML PKDD 2014*, Nancy, France, September 15–19, 2014. Proceedings, Part II, pp. 630–645.

Rooshenas, A., & Lowd, D. (2014). Learning sum–product networks with direct and indirect variable interactions. In Jebara, T., & Xing, E. P., (Eds.), *Proceedings of the 31st international conference on machine learning (ICML-14)*. JMLR workshop and conference proceedings, pp. 710–718.

Vergari, A., Mauro, N. D., & Esposito, F. (2015). Simplifying, regularizing and strengthening sum–product network structure learning. In *Proceedings of the 2015th European conference on machine learning and knowledge discovery in databases—Volume Part II*, ECMLPKDD'15, Switzerland. Springer, pp. 343–358.

Wainwright, M. J., & Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, *1*(1–2), 1–305.

Zhao, H., Melibari, M., & Poupart, P. (2015). On the relationship between sum–product networks and Bayesian networks. *CoRR*, arXiv:1501.01239.

Zhao, H., Poupart, P., & Gordon, G. (2016). A unified approach for learning the parameters of sum–product networks. In *Proceedings of the 29th advances in neural information processing systems (NIPS 2016)*.