Learning Accurate Cutset Networks by Exploiting Decomposability

Nicola Di Mauro (✉), Antonio Vergari, and Floriana Esposito

University of Bari "Aldo Moro", Bari, Italy
{nicola.dimauro,antonio.vergari,floriana.esposito}@uniba.it

Abstract. The rising interest around tractable Probabilistic Graphical Models is due to the guarantees on inference feasibility they provide. Among them, Cutset Networks (CNets) have recently been introduced as models embedding Pearl's cutset conditioning algorithm in the form of weighted probabilistic model trees with tree-structured models as leaves. Learning the structure of CNets has been tackled as a greedy search leveraging heuristics from decision tree learning. Even if efficient, the learned models are far from being accurate in terms of likelihood. Here, we exploit the decomposable score of CNets to learn their structure and parameters by directly maximizing the likelihood, including the BIC criterion and informative priors on smoothing parameters. In addition, we show how to create mixtures of CNets by adopting a well known bagging method from the discriminative framework as an effective and cheap alternative to the classical EM. We compare our algorithms against the original variants on a set of standard benchmarks for graphical model structure learning, empirically proving our claims.

1 Introduction

Probabilistic Graphical Models (PGMs) [8] provide a powerful formalism to model and reason about rich and structured domains. They capture the conditional independence assumptions among random variables into a graph based representation, sometimes called network (as in Bayesian Networks). Answering *inference* queries in PGMs often results in computing the probability of observing some evidence according the provided graphical structure. However, in general, to compute *exact* inference is a NP-Hard problem, and also some approximate inference routines are intractable in practice [19].

The pursuit for exact and efficient inference procedures led to the recently growing interest in the AI community around *tractable* PGMs. In exchange for inference tractability guarantees, they are less expressive, in the sense that they cannot possibly capture all the conditional probabilistic independences in the data. Tractable PGMs encompass *tree-structured models*, like those learned by the classical Chow-Liu algorithm [3] or by introducing latent variables [2], or even a bound on the treewidth of the model [1]; Bayesian and Markov Networks compiled into Arithmetic Circuits (ACs) [11,12]; and Sum-Product Networks (SPNs) [15] as deep architectures encoding probability distributions by layering

hidden variables as mixtures of independent components. As the expressiveness of these models increases, the complexity of learning their parameters and structure from data increases as well; as a matter of fact the overall performances degrade as data grows in size.

Cutset Networks (CNets) have been introduced recently in [17] as tractable PGMs with the aim of making learning efficient and scalable. They are weighted probabilistic model trees in the form of OR-trees having tree-structured models as leaves, and non-negative weights on inner edges, resulting into an architecture embedding Pearl's conditioning algorithm [14]. Inner nodes, i.e., conditioning OR nodes, are associated to random variables and outgoing branches represent conditioning on the values for those variables domains.

In [17], well known decision tree learning algorithms are leveraged to build a Cutset Network from data. In a nutshell, iteratively, training instances are split conditioning on the values of the best variable chosen as to maximize the reduction in an approximation of the joint entropy over all variables. While the computation of such a heuristic is cheap, it is not principled in a generative framework where model accuracy is measured as the scored data likelihood. The need to directly estimate the data likelihood is shown when a form of tree post pruning is introduced in [17] as a way to alleviate overfitting. Competitive results against state-of-the-art tractable PGM structure learners [13,18] are achieved when introducing mixtures of Cutset Networks via the Expectation Maximization algorithm (EM).

In this work we introduce a more principled way to learn Cutset Networks. We reformulate the search in the structure space as an optimization task directly maximizing data likelihood in a Bayesian framework. Regularization is achieved through the introduction of the Bayesian Information Criterion (BIC) in the likelihood score and by informative Dirichlet Priors on counting parameters while learning tree-structured models. Therefore, we avoid overfitting without adopting costly techniques like post pruning as in [17]. The direct optimization of the CNet likelihood has been obtained by exploiting its decomposability, leading to a tractable evaluation of the models during learning by limiting computations only on portions of data. We then introduce a very simple yet effective way to learn mixtures of Cutset Networks by exploiting *bagging* [6], opposed to the classical generative use of EM. We empirically verified the gain in terms of likelihood for the learned models with this new proposed approach against the original algorithm variants proposed in [17], with and without pruning, and MT, a solid competitor learning mixtures of trees as proposed in [13], on 18 datasets commonly used as benchmarks for graphical models structure learning.

2 Background

We define $\mathcal{D} = \{\xi_1, \ldots, \xi_M\}$ as a set of M i.i.d. instances over the discrete variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, whose domains are the sets $\text{Val}(X_i) = \{x_i^j\}_{j=1}^{k_i}, i = 1, \ldots, n$. We refer to the value assumed by an instance ξ_m in correspondence of a particular variable X_i as $\xi_m[X_i]$.

2.1 Tree-structured models

A *directed tree-structured model* [13] is a Bayesian Network in which each variable has at most one parent. Therefore, the joint probability distribution over $\mathbf{X}$ represented by such models can be written in the form of a factorization as:

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathrm{Pa}_i) \tag{1}$$

where Pa_i stands for the parent variable of X_i, if present. From Eq. 1 it follows immediately that inference for complete or marginal queries has complexity linear in the number of variables, hence the tractability of tree-structured models.

One classic result in learning tree-structured models is that presented by Chow and Liu in [3], where they prove that maximizing the Mutual Information (MI) among random variables in $\mathbf{X}$ leads to the best tree, in an information-theoretic sense, approximating the underlying probability distribution of $\mathcal{D}$ in terms of the Kullback-Leibler divergence.

Algorithm 1 LearnCLTree($\mathcal{D}$, $\mathbf{X}$, α)

1: **Input:** a set of instances $\mathcal{D}$ over a set of features $\mathbf{X}$; α smoothing parameter
2: **Output:** $\langle \mathcal{T}, \boldsymbol{\theta} \rangle$, a tree $\mathcal{T}$ with parameters $\boldsymbol{\theta}$ encoding a pdf over $\mathbf{X}$
3: $M \leftarrow \mathbf{0}_{|\mathbf{X}|\times|\mathbf{X}|}$
4: **for each** $X_u, X_v \in \mathbf{X}$ **do**
5: $\quad M_{u,v} \leftarrow$ estimateMutualInformation$(X_u, X_v, \mathcal{D}, \alpha)$
6: $T \leftarrow$ maximumSpanningTree(M)
7: $\mathcal{T} \leftarrow$ traverseTree(T)
8: $\boldsymbol{\theta} \leftarrow$ computeFactors$(\mathcal{D}, \mathcal{T})$
9: **return** $\langle \mathcal{T}, \boldsymbol{\theta} \rangle$

Algorithm 1 shows a sketch of the learning process. Firstly, for each pair of variables in $\mathbf{X}$, their MI is estimated from $\mathcal{D}$, optionally by introducing a smoothing factor α (line 5); then a maximum spanning tree is built on the weighted graph induced by the MI as an adjacency matrix (line 6). Rooting the tree in a randomly chosen variable and traversing it leads to the learned Bayesian Network (lines 7-9). In the following we will refer to tree-structured models simply as CLtrees.

CLtrees have been widely employed in AI, both as efficient approximations in density estimation tasks and as the building blocks of more expressive and yet tractable PGMs. A very simple and accurate algorithm to learn mixtures of CLtrees is MT, as presented in [13]. MT learns a distribution of the form: $Q(\mathbf{x}) = \sum_{i=1}^{k} \lambda_i \mathcal{T}_i(\mathbf{x})$, where the tree distributions $\mathcal{T}_i$, are the mixture components and $\lambda_i \geq 0$, such that $\sum_{i=1}^{k} \lambda_i = 1$, are their coefficients. In [13], the best components and weights are the (local) likelihood maxima learned by EM, with k being a parameter fixed in advance.

2.2 Cutset Networks

As introduced in [17], *Cutset Networks* (CNets) are a hybrid of rooted OR trees and CLtrees, with OR nodes as internal nodes and CLtrees as leaves. Each node in an OR tree is labeled by a variable X_i, and each edge emanating from it represents the conditioning of X_i by a value $x_i^j \in Val(X_i)$, weighted by the probability $w_{i,j}$ of conditioning the variable X_i to the value x_i^j.

Formally, a cutset network is a pair $\langle \mathcal{G}, \boldsymbol{\gamma} \rangle$, where $\mathcal{G} = \mathcal{O} \cup \{\mathcal{T}_1, \ldots, \mathcal{T}_L\}$ is composed of the rooted OR tree, $\mathcal{O}$, plus the leaf CLtrees $\mathcal{T}_l$, and $\boldsymbol{\gamma} = \boldsymbol{w} \cup \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L\}$ corresponds to the parameters $\boldsymbol{w}$ of the OR tree and $\boldsymbol{\theta}_l$ of the CLTrees. The *scope* of a CNet $\mathcal{G}$ (resp. a CLtree $\mathcal{T}_l$), denoted as scope($\mathcal{G}$) (resp. scope($\mathcal{T}_l$)), is the set of random variables that appear in it. A CNet may be defined recursively as follows.

Definition 1 (Cutset network). *Given* $\mathbf{X}$ *be a set of discrete variables, a* Cutset Network *is defined as follows:*

1. *a CLtree, with scope* $\mathbf{X}$*, is a CNet;*
2. *given* $X_i \in \mathbf{X}$ *a variable with* $|Val(X_i)| = k$*, graphically conditioned in an OR node, a weighted disjunction of* k *CNets* $\mathcal{G}_i$ *with same scope* $\mathbf{X}_{\setminus i}$ *is a CNet, where all weights* $w_{i,j}$*,* $j = 1, \ldots, k$*, sum up to one, and* $\mathbf{X}_{\setminus i}$ *denotes the set* $\mathbf{X}$ *minus the variable* X_i*.*

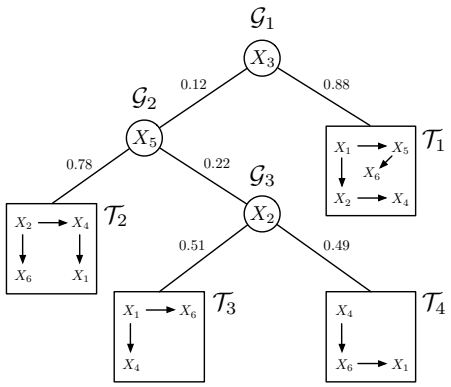

Fig. 1: Example of a binary CNet model. Internal nodes on variables X_i are OR nodes, while leaf nodes are CLtrees encoding a direct graphical model.

Figure 1 reports a CNet model for binary valued variables, where the internal nodes denote a conditioning on a variable (i.e., an OR node), while the leaves correspond to the CLtrees of the model. Note that each node in the model corresponds to a root of a sub-CNet $\mathcal{G}_i$ or to a CLtree $\mathcal{T}_j$, thus the recursive definition of a CNnet model.

The log-likelihood function of a CNet may be decomposed as follows.

Proposition 1 (CNet log-likelihood decomposition). *Given a CNet $\langle \mathcal{G}, \boldsymbol{\gamma} \rangle$ over variables $\mathbf{X}$ and a set of instances $\mathcal{D}$, its log-likelihood $\ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle)$ can be computed as follows:*

$$\ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) = \sum_{\xi \in \mathcal{D}} \sum_{i=1,\ldots,n} \log P(\xi[X_i] | \xi[\mathrm{Pa}_i]) \tag{2}$$

when $\mathcal{G}$ corresponds to a CLtree. While, in the case of a OR tree rooted on the variable X_i, with $|Val(X_i)| = k$, the log-likelihood is:

$$\ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) = \sum_{j=1,\ldots,k} M_j \log w_{i,j} + \ell_{\mathcal{D}_j}(\langle \mathcal{G}_j, \boldsymbol{\gamma}_{\mathcal{G}_j} \rangle) \tag{3}$$

where for each $j = 1, \ldots, k$, $\mathcal{G}_j$ is the CNet involved in the disjunction with parameters $\boldsymbol{\gamma}_{\mathcal{G}_j}$, and $\mathcal{D}_j$ is a slice of the dataset $\mathcal{D}$ obtained as $\mathcal{D}_j = \{\xi \in \mathcal{D} : \xi[X_i] = x_i^j\}$, $M_j = |\mathcal{D}_j|$ corresponds to the number of instances in $\mathcal{D}_j$, and $\ell_{\mathcal{D}_j}(\langle \mathcal{G}_j, \boldsymbol{\gamma}_{\mathcal{G}_j} \rangle)$ denotes the log-likelihood of the sub-CNet $\mathcal{G}_j$ on the slice $\mathcal{D}_j$.

Proof. For Equation (2):

$$\ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) = \sum_{\xi \in \mathcal{D}} \log \prod_{i=1,\ldots,n} P(\xi[X_i] | \xi[\mathrm{Pa}_i]) = \sum_{\xi \in \mathcal{D}} \sum_{i=1,\ldots,n} \log P(\xi[X_i] | \xi[\mathrm{Pa}_i])$$

While, for Equation (3):

$$\begin{aligned}
\ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) &= \sum_{\xi \in \mathcal{D}} \log w_{i,\xi[X_i]} + \log P(\langle \mathcal{G}_{\xi[X_i]}, \boldsymbol{\gamma}_{\mathcal{G}_{\xi[X_i]}} \rangle) \\
&= \sum_{j=1,\ldots,k} M_j \log w_{i,j} + \sum_{\xi \in \mathcal{D}_j} \log P(\langle \mathcal{G}_j, \boldsymbol{\gamma}_{\mathcal{G}_j} \rangle) \\
&= \sum_{j=1,\ldots,k} M_j \log w_{i,j} + \ell_{\mathcal{D}_j}(\langle \mathcal{G}_j, \boldsymbol{\gamma}_{\mathcal{G}_j} \rangle)
\end{aligned}$$

■

Structure Learning The algorithm to learn CNet structures proposed in [17] performs a greedy top-down search in the OR-trees space. It recursively tries to partition $\mathcal{D}$ into the instance subsets $\{\mathcal{D}_j = \{\xi \in \mathcal{D} : \xi[X_s] = x_s^j\}\}_{j=1}^{|Val(X_s)|}$ over the current scope $\mathbf{X}_{\setminus s}$ by selecting heuristically the best variable X_s maximizing a reformulation of the information gain in a generative context:

$$X_s = \operatorname*{argmax}_{X_i \in \mathbf{X}} \left(\hat{H}_{\mathcal{D}}(\mathbf{X}) - \sum_{x_i^j \in Val(X_i)} \frac{|\mathcal{D}_j|}{|\mathcal{D}|} \hat{H}_{\mathcal{D}_j}(\mathbf{X}_{\setminus i}) \right)$$

where $\hat{H}_{\mathcal{D}_j}(\mathbf{X}) = -\frac{1}{|\mathbf{X}|} \sum_{X_i \in \mathbf{X}} \sum_{x_i^j \in Val(X_i)} P_{\mathcal{D}_j}(x_i^j) \log P_{\mathcal{D}_j}(x_i^j)$ is the average entropy over the single variables in the current scope, limited to the subset $\mathcal{D}_j$, which is introduced as a way to approximate the computation of the joint entropy over the current scope.

Found X_s, the algorithm creates a corresponding inner node C_s whose children will be the nodes $\{C_j\}_{j=1}^{k_s}$ returned by recursive calls on the instance subsets $\{\mathcal{D}_j\}_{j=1}^{k_s}$, with $k_s = |Val(X_s)|$. The weights $\{w_{sj}\}_{j=1}^{k_s}$ are estimated as the proportion of instances falling into each partition. As reported in [17], termination can be achieved when for the current partition $\mathcal{D}$ the number of instances falls under a tunable parameter m, or when the total entropy is less than a threshold λ. In this case a leaf node is added as a CLtree learned on the current instance partition over the current scope according to Algorithm 1. To cope with the risk of overfitting, always in [17], post-pruning based on a validation set is introduced in the form of reduced error pruning [16]. Leveraging this decision tree technique, after a CNet is fully grown, by advancing bottom-up, leaves are pruned and inner nodes without children replaced with a CLtree, if the network validation data likelihood after this operation is higher than that scored by the unpruned network. Following experimental evidence, it appears clear that a search step directly guided by the data likelihood, in this case the pruning stage, is crucial for the accuracy of the learned models; otherwise representing very poor local optima in the terms of likelihood. However, as the same authors report, the additional cost of growing a full network and then traversing it while reevaluating inner nodes, is demanding.

3 dCSN: decomposability based CNets learning

Here, we propose the dCSN algorithm that exploits a different approach from that in [17]: we avoid decision tree heuristics and instead choose the best variable directly maximizing the data log-likelihood. By exploiting the recursive Definition 1, we grow a CNet top-down, allowing further expansion, i.e. the substitution of a CLtree with an OR node, only if it improves the structure log-likelihood, since it is clear to see that maximizing the second term in Equation 3, results in maximizing the global score. In detail, we start with a single CLtree, for variables $\mathbf{X}$, learned from $\mathcal{D}$ and we check whether there is a decomposition, i.e. an OR node applied on as many CLtrees as the values of the best variable X_i, providing a better log-likelihood than that scored by the initial tree. If such a decomposition exists, than the decomposition process is recursively applied to the sub-slices $\mathcal{D}_i$, testing each leaf for a possible substitution. A sketch of the process is shown in Algorithm 2.

In this principled learning framework we do not need to employ post-pruning techniques while we can embed a regularization term in the structure score used in the decomposition process. To penalize complex structures we adopt the Bayesian Information Criterion BIC, we now show how to derive it in our framework and what are its properties. Using another criterion like the BDe [7] could be possible. Following [4], the BIC score of a CNet $\langle \mathcal{G}, \boldsymbol{\gamma} \rangle$ on data $\mathcal{D}$ is defined as: $\text{score}_{\text{BIC}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) = \log P_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) - \frac{\log M}{2} \text{Dim}(\mathcal{G})$, where $\text{Dim}(\mathcal{G})$ is the model dimension, i.e., the number of independent parameters used for the structure representation $\mathcal{G}$, and M is the size of the dataset $\mathcal{D}$. Here, we set $\text{Dim}(\mathcal{G})$ to the number $O_{\mathcal{G}}$ of OR nodes appearing in $\mathcal{G}$.

Given $\mathcal{G}$ and $\mathcal{G}'$ be two CNets, where $\mathcal{G}'$ has been obtained from $\mathcal{G}$ substituting a leaf tree by adding a new sub-CNet rooted in an OR node, then:

$$\begin{aligned}\text{score}_{\text{BIC}}(\langle \mathcal{G}', \boldsymbol{\gamma}' \rangle) - \text{score}_{\text{BIC}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) = \\ \ell_{\mathcal{D}}(\langle \mathcal{G}', \boldsymbol{\gamma}' \rangle) - \ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) - \frac{\log M}{2}(\text{Dim}(\mathcal{G}') - \text{Dim}(\mathcal{G})) = \\ \ell_{\mathcal{D}}(\langle \mathcal{G}', \boldsymbol{\gamma}' \rangle) - \ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) - \frac{\log M}{2}, \end{aligned} \tag{4}$$

since $\text{Dim}(\mathcal{G}') - \text{Dim}(\mathcal{G}) = O_{\mathcal{G}'} - O_{\mathcal{G}} = (O_{\mathcal{G}} + 1) - O_{\mathcal{G}} = 1$. Hence, $\mathcal{G}'$ is accepted when $\ell_{\mathcal{D}}(\langle \mathcal{G}', \boldsymbol{\gamma}' \rangle) - \ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) > \frac{\log M}{2}$. This means that a leaf node may be decomposed (or, a new OR node may be added), when the improvement on the global loglikelihood is greater than $\frac{\log M}{2}$.

The decomposability property of the log-likelihood of a CNet can lead to similar results for the BIC score.

Proposition 2 (CNet BIC score decomposition). *Given a CNet $\langle \mathcal{G}, \boldsymbol{\gamma} \rangle$, over variables $\mathbf{X}$ and instances $\mathcal{D}$, made up of $\{\mathcal{T}_l\}_{l=1}^{L}$ CLtrees, a decomposition of a tree $\mathcal{T}_l$, having scope $\mathbf{X}_l \subset \mathbf{X}$, with parameters $\boldsymbol{\theta}_l$, with a sub-CNet $\mathcal{G}_i$ rooted in a OR node associated to the variable $X_i \in \mathbf{X}_l$ with parameters $\boldsymbol{\gamma}_i$, leading to a new CNet $\langle \mathcal{G}', \boldsymbol{\gamma}' \rangle$, is accepted iff:*

$$\ell_{\mathcal{D}_l}(\langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle) - \ell_{\mathcal{D}_l}(\langle \mathcal{T}_l, \boldsymbol{\theta}_l \rangle) > \frac{\log M}{2}. \tag{5}$$

where $M = |\mathcal{D}|$, and D_l is the subset of D containing only instances associated to the tree $\mathcal{T}_l$.

Proof. Each leaf tree node $\mathcal{T}_l$, $l \in \{1, \ldots, L\}$ is reachable from the root through the path $x_{i_1}^{j_{i_1}}, x_{i_2}^{j_{i_2}}, \ldots, x_{i_P}^{j_{i_P}}$ of length P where $\langle i_1, \ldots, i_P \rangle$ is the sequence of indices for the random variables $\mathbf{X}_{\setminus l} = \mathbf{X} \setminus \mathbf{X}_l$ found in the path. Instances reaching the tree $\mathcal{T}_l$ form the set $\mathcal{D}_l = \{\xi \in \mathcal{D} : \xi[X_{i_p}] = x_{i_p}^{j_{i_p}}, \forall p = 1, \ldots, P\}$, that is, for each random variable X_{i_p} in the path, they take the conditioned branch according to their value for X_{i_p}.

If $\ell_{\mathcal{D}_l}(\langle \mathcal{T}_l, \boldsymbol{\theta}_l \rangle)$ indicates the *local* log-likelihood of $\mathcal{T}_l$ with respect to $\mathcal{D}_i$, then its contribution to the *global* log-likelihood $\ell_{\mathcal{D}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle)$ corresponds to:

$$\left(M_l \sum_{p=1}^{P} \log w_{i_p, j_{i_p}} \right) + \ell_{\mathcal{D}_l}(\langle \mathcal{T}_l, \boldsymbol{\theta}_l \rangle), \tag{6}$$

where $M_l = |\mathcal{D}_l|$. If we decompose the tree $\mathcal{T}_l$ into a sub-CNet $\mathcal{G}_i$, using the CNet log-likelihood decomposition as reported in Equation 3, then the global contribution reported in Equation 6 becomes:

$$\left(M_l \sum_{p=1}^{P} \log w_{i_p, j_{i_p}} \right) + \ell_{\mathcal{D}_l}(\langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle), \tag{7}$$

We have that $\text{score}_{\text{BIC}}(\langle \mathcal{G}', \boldsymbol{\gamma}' \rangle) - \text{score}_{\text{BIC}}(\langle \mathcal{G}, \boldsymbol{\gamma} \rangle) = \ell_{\mathcal{D}_l}(\langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle) - \ell_{\mathcal{D}_l}(\langle \mathcal{T}_l, \boldsymbol{\theta}_l \rangle) - \frac{\log M}{2}$. ∎

Again, instead of recomputing the likelihood on the complete dataset $\mathcal{D}$, due to the decomposability of the likelihood, we can evaluate only the local improvement. Moreover, the decomposition of $\mathcal{T}_l$ is independent from all other $\mathcal{T}_j, j \neq l$ being their local contributions to the global log-likelihood independent. Hence, it is not significant the order we choose to decompose leaf nodes.

Bayesian Parameter Smoothing As regards the learning of the CLtrees parameters we adopted a Bayesian approach. To learn the structure of a CLtree $\mathcal{T}$ from data $\mathcal{D}$ with parameters $\boldsymbol{\theta}$, the Bayesian approach employs as a scoring function the posterior probability of the graph given the data: $P(\boldsymbol{\theta}|\mathcal{D}) \approx P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})$. The marginal $P(\mathcal{D}|\boldsymbol{\theta})$ can be expressed in closed form when using the Dirichlet prior over the model parameters $\boldsymbol{\theta}_{X_i|\mathrm{Pa}_i}$, the only distribution that ensures likelihood equivalence, i.e., the hyper-parameters $\boldsymbol{\alpha}_{X_i|\mathrm{Pa}_i}$ of the Dirichlet prior can be expressed as $\alpha_{X_i,\mathrm{Pa}_i} = \alpha q_{X_i,\mathrm{Pa}_i}$, where q is a prior distribution over X, and α, the so called *equivalent sample size* (ESS), is a positive constant independent of i. In this Bayesian approach with the Dirichlet prior, the regularized parameter estimates are:

$$\theta_{x_i|\mathrm{Pa}_i} \approx E_{P(\theta_{x_i|\mathrm{Pa}_i}|\mathcal{D},\mathcal{T})}[\theta_{x_i|\mathrm{Pa}_i}] = \frac{M_{x_i,\mathrm{Pa}_i} + \alpha_{x_i|\mathrm{Pa}_i}}{M_{\mathrm{Pa}_i} + \alpha_{\mathrm{Pa}_i}}, \tag{8}$$

where $M_{\mathbf{z}}$ is the number of entries in a dataset $\mathcal{D}_{\mathbf{z}}$ having the set of variables $\mathbf{Z}$ instantiated to $\mathbf{z}$. As pointed out in [4], we can use a different Dirichlet prior for each distribution of X_i given a particular value of its parents, leading to choose the regularized parameter estimates as:

$$\hat{\boldsymbol{\theta}}_{X_i|\mathrm{Pa}_i} = \frac{M \cdot P(\mathrm{Pa}_i)P(X_i|\mathrm{Pa}_i)}{M \cdot P(\mathrm{Pa}_i) + \boldsymbol{\alpha}_{X_i|\mathrm{Pa}_i}} + \frac{\boldsymbol{\alpha}_{X_i|\mathrm{Pa}_i}\boldsymbol{\theta}^0(X_i|\mathrm{Pa}_i)}{M \cdot P(\mathrm{Pa}_i) + \boldsymbol{\alpha}_{X_i|\mathrm{Pa}_i}},$$

where $\boldsymbol{\theta}^0(X_i|\mathrm{Pa}_i)$ is the prior estimate of $P(X_i|\mathrm{Pa}_i)$ and $\boldsymbol{\alpha}_{X_i|\mathrm{Pa}_i}$ is the confidence associated with that prior.

In the case of uniform priors, the estimates correspond to the additive of Laplace smoothing. A reasonable choice uses the marginal probability of X_i in the data as the prior probability. This choice is based on the assumption that most conditional probabilities are close to the observed marginal. Thus, we can set $\boldsymbol{\theta}^0(X_i|\mathrm{Pa}_i) = P_D(X_i)$. With fixed $\alpha_{x_i|\mathrm{Pa}_i} = \alpha$, we have: $\hat{\theta}_{X_i|\mathrm{Pa}_i} = \frac{M_{x_i,\mathrm{Pa}_i} + \alpha P_D(X_i)}{M_{\mathrm{Pa}_i} + \alpha}$.

Algorithm 2 reports the pseudocode of dCSN. The dCSN algorithm starts by learning a single CLTree on the whole dataset $\mathcal{D}$ (line 4), and then calls the decomposition procedure on this tree (line 6). The input parameters δ and σ are used for regularization in order to avoid overfitting. σ, resp. δ, is the minimum number of instances, resp. of features, in a slice required to try a decomposition.

Given a CLtree, Algorithm 3 tries to decompose it in a sub-CNet. The aim of dCSN is to attempt to extend the model by replacing one of the CLtree leaf nodes with a new CNet on the same variables. In particular, the decompose procedure checks for each variable X_i on the slice $\mathcal{D}$ (line 5), whether the OR decomposition

Algorithm 2 dCSN($\mathcal{D}$, $\mathbf{X}$, α_f, δ, σ)

1: **Input:** a set of instances $\mathcal{D}$ over a set of features $\mathbf{X}$; $\alpha_f \in [0,1]$: ESS factor; δ minimum number of instances to decompose, σ minimum number of features to decompose
2: **Output:** a CNet $\langle \mathcal{G}, \boldsymbol{\gamma} \rangle$ encoding a pdf over $\mathbf{X}$ learned from $\mathcal{D}$
3: $\alpha \leftarrow \alpha_f |\mathcal{D}|$
4: $\langle \mathcal{T}, \boldsymbol{\theta} \rangle \leftarrow$ LearnCLTree($\mathcal{D}, \mathbf{X}, \alpha$)
5: $\boldsymbol{w} \leftarrow \emptyset$
6: $\langle \mathcal{G}, \boldsymbol{\gamma} \rangle \leftarrow$ decompose($\mathcal{D}, \mathbf{X}, \alpha, \mathcal{T}, \boldsymbol{\theta}, \boldsymbol{w}, \delta, \sigma$)

associated to that variable (a new CNet) has a log-likelihood better than that of the input CLtree (line 16). If a better decomposition is found, it then recursively (line 21) tries to decompose the sub-CLtrees of the newly introduced CNet. In dCSN α is set to $\alpha_f|\mathcal{D}|$, where $\alpha_f \in [0,1]$ is an input parameter. When we proceed with the decomposition on the slices, α is proportionally reduced, in the procedure decompose, to the number of instances in the slices. In particular, if we initially assume that there are $\alpha = \alpha_f|\mathcal{D}|$ fictitious instances for computing the priors, then we should assume that a proportion $\alpha|\mathcal{D}_i|/|\mathcal{D}|$ falls into the slice $\mathcal{D}_i$, in order to make the priors in $\mathcal{D}_i$ consistent with those in $\mathcal{D}$.

Algorithm 3 decompose($\mathcal{D}$, $\mathbf{X}$, α, $\mathcal{T}, \boldsymbol{\theta}, \boldsymbol{w}, \delta, \sigma$)

1: **Input:** a set of instances $\mathcal{D}$ over a set of features $\mathbf{X}$; α: ESS; $\mathcal{T}$: the tree structured model to decompose and its parameters $\boldsymbol{\theta}$; δ minimum number of instances to decompose, σ minimum number of features to decompose
2: **Output:** a CNet encoding a pdf over $\mathbf{X}$ learned from $\mathcal{D}$
3: **if** $|\mathcal{D}| > \delta$ and $|\mathbf{X}| > \sigma$ **then**
4: $\quad \ell_{\text{best}} \leftarrow -\infty$
5: $\quad$ **for** $X_i \in \mathbf{X}$ **do**
6: $\quad\quad \mathcal{G}_i \leftarrow \emptyset, \boldsymbol{w}_i \leftarrow \emptyset, \boldsymbol{\theta}_i \leftarrow \emptyset, C_i$ is the OR Node associated to X_i
7: $\quad\quad$ **for** $x_i^j \in Val(X_i)$ **do**
8: $\quad\quad\quad \mathcal{D}_j \leftarrow \{\xi \in \mathcal{D} : \xi[X_s] = x_s^j\}$
9: $\quad\quad\quad w_{ij} \leftarrow |\mathcal{D}_j|/|\mathcal{D}|$
10: $\quad\quad\quad \langle \mathcal{T}_j, \boldsymbol{\theta}_{ij} \rangle \leftarrow$ LearnCLTree($\mathcal{D}_j, \mathbf{X}_{\setminus s}, \alpha w_{ij}$)
11: $\quad\quad\quad \mathcal{G}_i \leftarrow$ addSubTree($C_i, \mathcal{T}_j$)
12: $\quad\quad\quad \boldsymbol{w}_i \leftarrow \boldsymbol{w}_i \cup \{w_{ij}\}, \boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i \cup \{\theta_{ij}\}$
13: $\quad\quad \ell_i \leftarrow \ell_{D_i}(\langle \mathcal{G}_i, \boldsymbol{w}_i \cup \boldsymbol{\theta}_i \rangle)$
14: $\quad\quad$ **if** $\ell_i > \ell_{\text{best}}$ and $\ell_i > \ell_{D_i}(\langle \mathcal{T}, \boldsymbol{\theta} \rangle)$ **then**
15: $\quad\quad\quad \ell_{\text{best}} \leftarrow \ell_i$, $X_{\text{best}} \leftarrow X_i$, $\mathcal{G}_{best} \leftarrow \mathcal{G}_i$, $\boldsymbol{\theta}_{best} \leftarrow \boldsymbol{\theta}_i, \boldsymbol{w}_{best} \leftarrow \boldsymbol{w}_i$
16: $\quad$ **if** $\ell_{\text{best}} - \ell_{\mathcal{D}}(\langle \mathcal{T}, \boldsymbol{\theta} \rangle) > (log|\mathcal{D}|)/2$ **then**
17: $\quad\quad$ substitute $\mathcal{T}$ with $\mathcal{G}_i$
18: $\quad\quad \boldsymbol{w} \leftarrow \boldsymbol{w} \cup \boldsymbol{w}_{best}$
19: $\quad\quad$ **for** $x_b^j \in Val(X_{\text{best}})$ **do**
20: $\quad\quad\quad \mathcal{D}_j \leftarrow \{\xi \in \mathcal{D} : \xi[X_{\text{best}}] = x_b^j\}$
21: $\quad\quad\quad$ decompose($\mathcal{D}_j, \mathbf{X}_{\setminus \text{best}}, \alpha w_{ij}, \mathcal{T}_j, \boldsymbol{\theta}_j, \boldsymbol{w}, \delta, \sigma$)

	$\|\mathbf{X}\|$	$\|T_{train}\|$	$\|T_{val}\|$	$\|T_{test}\|$		$\|\mathbf{X}\|$	$\|T_{train}\|$	$\|T_{val}\|$	$\|T_{test}\|$
NLTCS	16	16181	2157	3236	**DNA**	180	1600	400	1186
MSNBC	17	291326	38843	58265	**Kosarek**	190	33375	4450	6675
Plants	69	17412	2321	3482	**MSWeb**	294	29441	3270	5000
Audio	100	15000	2000	3000	**Book**	500	8700	1159	1739
Jester	100	9000	1000	4116	**EachMovie**	500	4525	1002	591
Netflix	100	15000	2000	3000	**WebKB**	839	2803	558	838
Accidents	111	12758	1700	2551	**Reuters-52**	889	6532	1028	1540
Retail	135	22041	2938	4408	**BBC**	1058	1670	225	330
Pumsb-star	163	12262	1635	2452	**Ad**	1556	2461	327	491

Table 1: Datasets used and their number of features and instances.

Finally, in order to improve the accuracy of the CNet models we adopted a bagging procedure in order to obtain a mixture of CNets. We draw k bootstrapped samples $\mathcal{D}_i$ from the dataset $\mathcal{D}$, sampling $|\mathcal{D}|$ instances with replacements, and on each of those we call dCSN, thus leading to k CNets $\mathcal{G}_i$. The resulting bagged CNet $\mathcal{G}$ corresponds to a weighted sum of all the learned CNets $\mathcal{G}_i$. We set the weights proportional to the likelihood score obtained by each bootstrapped component. In particular, for each instance $\xi \in \mathcal{D}$, the bagged CNet $\mathcal{G}$ would result in the more robust estimation $\hat{P}(\xi : \mathcal{G}) = \sum_{i=1}^{k} \mu_i P(\xi : \mathcal{G}_i)$, where $\mu_i = \ell_{\mathcal{D}}(\langle \mathcal{G}_i, \boldsymbol{\gamma}_i \rangle) / \sum_{j=1}^{k} \ell_{\mathcal{D}}(\langle \mathcal{G}_j, \boldsymbol{\gamma}_j \rangle)$.

4 Experiments

Since the CNet, and the variant CNetP embedding the pruning on validation, as reported in [17] are not publicly available, we implemented them as well as our dCSN and its bagging dCSN-B variant in Python[1]. We were not able to reproduce the results of the mixtures learned with EM as showed in [17], therefore we will just report them (as MCNet). To make the comparison fair in testing the mixture accuracies, we also extended CNet and CNetP by embedding mixtures by bagging, leading to versions CNet-B and CNetP-B respectively. We introduce MT as the last competitor as it is reported to be one of the most competitive tractable PGM [18,17]; for it we used the implementation available in the Libra toolkit [9].

We evaluated the proposed algorithms on an array of 18 datasets that are now standard benchmarks for graphical model structure learners. They have been introduced in [10] and [5] as binarized versions of datasets from different tasks like frequent itemset mining, recommendation and classification. Their names and statistics for their training, validation, test splits are reported in Table 1.

We run both CNet and CNetP with $m = 10$ and $\lambda = 0.01$ fixed to exactly reproduce the original experiments in [17]. We run CNet-B and CNetP-B by learning a number of components k ranging from 5 to 40, with a step of 5.

For dCSN we run a grid search in the space formed by $\alpha_f \in \{.01, .02, .03, .04, .05, .06, .08, .1, .15, .2, .3, .4, .5\}$ and $\delta \in \{200, 300, 400, 500\}$; for dCSN-B we set

[1] Source code is available at `http://www.di.uniba.it/~ndm/dcsn/`.

	CNet	CNetP	dCSN	CNet-B	CNetP-B	dCSN-B	MT	MCNet
NLTCS	-6.11	-6.06	**-6.04**	-6.09	-6.02	-6.02	**-6.01**	-6.00
MSNBC	-6.06	**-6.05**	-6.05	-6.06	**-6.04**	-6.04	-6.08	-6.04
Plants	**-13.24**	-13.25	-13.35	-12.30	-12.38	**-12.21**	-12.93	-12.78
Audio	-44.58	**-42.05**	**-42.06**	-42.09	-40.71	**-40.17**	**-40.14**	-39.73
Jester	-61.71	**-55.56**	**-55.30**	-57.76	-53.17	**-52.99**	-53.06	-52.57
Netflix	-65.61	**-58.71**	**-58.57**	-63.08	-57.63	**-56.63**	-56.71	-56.32
Accidents	-30.97	-30.69	**-30.17**	-30.25	-30.28	**-28.99**	-29.69	-29.96
Retail	-11.07	**-10.94**	-11.00	-10.99	-10.88	-10.87	**-10.84**	-10.82
Pumsb-star	-24.65	-24.42	**-23.83**	-24.39	-24.19	**-23.32**	-23.70	-24.18
DNA	-90.48	-87.59	**-87.19**	-90.66	-86.85	**-84.93**	**-85.57**	-85.82
Kosarek	-11.19	**-11.04**	-11.14	-10.97	-10.85	-10.85	**-10.62**	-10.58
MSWeb	-10.07	-10.07	**-9.94**	-9.95	-9.91	-9.86	**-9.82**	-9.79
Book	-37.62	-37.35	**-37.22**	-35.88	-35.62	-35.92	**-34.69**	-33.96
EachMovie	**-59.19**	**-58.37**	**-58.47**	-54.22	-54.02	**-53.91**	-54.51	-51.39
WebKB	-162.85	-162.17	**-161.16**	-156.79	-156.94	**-155.20**	-157.00	-153.22
Reuters-52	**-88.72**	**-88.55**	**-88.60**	-86.22	-86.89	**-85.69**	-86.53	-86.11
BBC	**-262.08**	**-263.08**	**-262.08**	**-252.01**	-257.72	**-251.14**	-259.96	-250.58
Ad	-16.92	-16.92	**-14.81**	-15.94	-16.02	**-13.73**	-16.01	-16.68

Table 2: Empirical risk for all algorithms.

instead $\alpha_f \in \{.05, .1\}$ and $\delta \in \{100, 200, 300, 400, 500, 1000\}$, running the algorithm for a number of components k ranging from 5 to 40, with a step of 5. For both dCSN and dCSN-B we fixed $\sigma = 3$. For MT we reproduced the experiment in [18], setting k from 2 to 30 by steps of 2. For all mixture variants, for each mixture configuration, we selected the best one based on the validation likelihood score.

In Table 2 is reported the empirical risk, defined as $1/|\mathcal{D}| \sum_{\xi \in \mathcal{D}} \log P(\xi|\mathcal{G}, \boldsymbol{\gamma})$ averaged over the set of test instances for all the experiments over the listed datasets. We provide in last column the original scores of MCNet as reported in [17] as a reference. For all the implemented versions we run a pairwise Wilcoxon signed rank test to assess the statistical significance of the scores. In bold are reported the best values, compared to all others, for each dataset. As we can see dCSN is significantly better than CNet and CNetP on 8 datasets, and significantly worse than CNet and CNetP on 1 and 3 datasets, respectively. Considering the bagging version for the mixtures, we see that dCSN-B is significantly better than CNet-B, CNetP-B and MT on 11, 11, and 10 datasets, and significantly worse on 1, 1, and 5 datasets, respectively.

5 Conclusions

Here we proposed a new approach to learn the structure of the recently introduced CNets model. We exploited the decomposable score of CNets to learn their structure and parameters by directly maximizing the likelihood, formulating a score including the BIC criterion and by introducing informative priors on smoothing parameters. Moreover, we presented how to create mixtures of CNets by adopting the bagging method as an alternative to EM. We compared our algorithm against the original variants on a large set of standard benchmarks proving the validity of our claims.

Acknowledgements

This work has been partially founded by the PON02 00563 3489339 project PUGLIA@SERVICE financed by the Italian Ministry of University and Research (MIUR).

References

1. Bach, F.R., Jordan, M.I.: Thin junction trees. In: Advances in Neural Information Processing Systems 14. pp. 569–576. MIT Press (2001)
2. Choi, M.J., Tan, V.Y.F., Anandkumar, A., Willsky, A.S.: Learning latent tree graphical models. Journal of Machine Learning Research 12, 1771–1812 (2011)
3. Chow, C., Liu, C.: Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory 14(3), 462–467 (1968)
4. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine learning 29(2-3), 131–163 (1997)
5. Haaren, J.V., Davis, J.: Markov network structure learning: A randomized feature generation approach. In: Proceedings of the 26th Conference on Artificial Intelligence. AAAI Press (2012)
6. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer (2009)
7. Heckerman, D., Geiger, D., Chickering, D.: Learning bayesian networks: the combination of knowledge and statistical data. Machine learning 20, 197–243 (1995)
8. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
9. Lowd, D., Rooshenas, A.: The Libra Toolkit for Probabilistic Models. CoRR abs/1504.00110 (2015)
10. Lowd, D., Davis, J.: Learning Markov network structure with decision trees. In: Proceedings of the 10th IEEE International Conference on Data Mining. pp. 334–343. IEEE Computer Society Press (2010)
11. Lowd, D., Domingos, P.: Learning arithmetic circuits. CoRR abs/1206.3271 (2012)
12. Lowd, D., Rooshenas, A.: Learning Markov networks with arithmetic circuits. In: Proceedings of the 16th International Conference on Artificial Intelligence and Statistics. JMLR Workshop Proceedings, vol. 31, pp. 406–414 (2013)
13. Meilă, M., Jordan, M.I.: Learning with mixtures of trees. Journal of Machine Learning Research 1, 1–48 (2000)
14. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1988)
15. Poon, H., Domingos, P.: Sum-product network: a new deep architecture. NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning (2011)
16. Quinlan, J.R.: Induction of decision trees. Machine Learning Journal 1(1), 81–106 (1986)
17. Rahman, T., Kothalkar, P., Gogate, V.: Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees. In: Machine Learning and Knowledge Discovery in Databases, LNCS, vol. 8725, pp. 630–645. Springer (2014)
18. Rooshenas, A., Lowd, D.: Learning sum-product networks with direct and indirect variable interactions. In: Proceedings of the 31st International Conference on Machine Learning. pp. 710–718. JMLR Workshop and Conference Proceedings (2014)
19. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence 82(1–2), 273–302 (1996)