

Prometheus : Directly Learning Acyclic Directed Graph Structures for Sum-Product Networks

Priyank Jaini

University of Waterloo, Waterloo AI Institute, Vector Institute, Ontario, Canada

PJAINI@UWATERLOO.CA

Amur Ghose

Indian Institute of Technology, Kanpur, India

AMUR@IITK.AC.IN

Pascal Poupart

University of Waterloo, Waterloo AI Institute, Vector Institute, Ontario, Canada

PPOUPART@UWATERLOO.CA

Abstract

In this paper, we present Prometheus, a graph partitioning based algorithm that creates multiple variable decompositions efficiently for learning Sum-Product Network structures across both continuous and discrete domains. Prometheus proceeds by creating multiple candidate decompositions that are represented compactly with an acyclic directed graph in which common parts of different decompositions are shared. It eliminates the correlation threshold hyperparameter often used in other structure learning techniques, allowing Prometheus to learn structures that are robust in low data regimes. Prometheus outperforms other structure learning techniques in 30 discrete and continuous domains. We also describe a sampling based approximation of Prometheus that scales to high-dimensional domains such as images.

1. Introduction

Sum-Product Networks (SPNs) (Poon and Domingos, 2011; Darwiche, 2003) are a class of deep probabilistic graphical models in which inference is tractable. Unlike most deep neural networks, SPNs have clear semantics in the sense that each node in the network computes a local (unnormalized) probability over the variables in its scope. An SPN, therefore, evaluates a joint distribution over the variables in its leaves. Furthermore, SPNs are equivalent to Bayesian Networks and Markov Networks (Zhao et al., 2015) in which exact (marginal) inference is linear in the size of the network¹. Therefore, an SPN has a dual view: it is both a probabilistic graphical model and a deep neural network.

Unlike most neural networks that can answer only queries with fixed inputs and outputs, SPNs can answer conditional inference queries with varying inputs and outputs by changing the set of variables that are queried (outputs) and conditioned on (inputs). Furthermore, SPNs can be used to generate data by sampling from the joint distributions they encode. This is achieved by a top-down pass through the network. Starting at the root, each child of a product node is traversed. A single child of a sum node is sampled according to the unnormalized distribution encoded by the weights of the sum node and a variable assignment is sampled in each leaf that is reached. This is particularly useful in natural language generation and image completion tasks (Poon and Domingos, 2011; Cheng et al., 2014).

The graph of an SPN is a computational graph, but it also implicitly encodes probabilistic dependencies between features (though this is done differently than traditional probabilistic graphical models). Several structure learning techniques detect correlations/independence and construct a graphical structure accordingly. For instance, decomposable product nodes can be interpreted as computing the product of marginal distributions (encoded by their children) over some features that are locally independent. Hence, some structure learning techniques construct product nodes by greedily partitioning features into subsets of variables that are assumed to be independent. Since it is rarely the case that variables can be partitioned into locally independent subsets, a correlation threshold hyperparameter is often used to identify nearly independent sub-

1. It is common to measure the complexity of probabilistic graphical models with respect to their tree-width, however tree-width is not a practical statistic since finding the tree-width of a graph is NP-hard. Instead, we describe the complexity of inference with respect to the size of the graph (number of nodes, edges and parameters), which is immediately available.

sets. The best setting for this hyperparameter is often problem dependent. This is further complicated by the fact that testing independence based on this hyperparameter does not take into account the rest of the graph and the test is typically done on a subset of the data that might not be representative due to approximations introduced by earlier clustering steps.

We propose a new structure learning algorithm called Prometheus that uses a graph partitioning technique to produce multiple decompositions instead of greedily selecting a single variable decomposition at each step. This eliminates the correlation threshold hyperparameter in Prometheus. The candidate variable decompositions are represented compactly by an acyclic directed graph in which common parts for several decompositions are shared.

The paper makes the following contributions - a new structure learning algorithm, Prometheus, for SPNs that works for both discrete and continuous domains and eliminates the sensitive correlation threshold hyperparameter. Prometheus produces acyclic directed graphs with subtree sharing directly unlike most existing structure learning techniques that produce trees and then share subtrees in a post-processing step (with the exception of (Dennis and Ventura, 2015) who proposed a greedy search algorithm based on (Gens and Domingos, 2013) that directly shares subtrees). We compare Prometheus to existing algorithms like LearnSPN, ID-SPN, oSLRAU, iSPT and DV (Gens and Domingos, 2013; Rooshenas and Lowd, 2014; Hsu et al., 2017; Trapp et al., 2016; Dennis and Ventura, 2015) by performing empirical evaluations on 30 real world datasets. Prometheus improves the state-of-the-art on most datasets. We further demonstrate empirically that capturing correlations efficiently using graph partitioning makes Prometheus robust even in low data regimes. Finally, we propose a scalable approximation of Prometheus based on sampling and demonstrate its efficacy on high dimensional datasets. Prometheus achieves the best reported accuracy by an SPN on MNIST image data.

2. Related Work

Several structure learning algorithms have been proposed for Sum-Product Networks with considerable success (Dennis and Ventura, 2012; Gens and Domingos, 2013; Peharz et al., 2013; Lee et al., 2013; Rooshenas and Lowd, 2014; Adel et al., 2015; Vergari et al., 2015; Rahman and Gogate, 2016; Hsu et al., 2017; Dennis and Ventura, 2015). However, each algorithm lacks certain aspects, which prohibit their straightforward use in real scenarios. In this section, we shed light on each of these previously proposed structure learning algorithms and discuss the advantages of Prometheus over these previous algorithms. Prometheus is easy for practitioners to implement and use.

A popular approach (Dennis and Ventura, 2012; Gens and Domingos, 2013; Rooshenas and Lowd, 2014) for structure learning alternates between instance clustering to construct sum nodes and variable partitioning to construct product nodes in a top down fashion. Alternatively, structures can be learned in a bottom-up fashion by incrementally clustering correlated variables (Peharz et al., 2013; Hsu et al., 2017).

For discrete domains, LearnSPN (Gens and Domingos, 2013) and ID-SPN (Rooshenas and Lowd, 2014) are the two most popular structure learning algorithms. In terms of results on the 20 benchmark datasets, ID-SPN outperforms LearnSPN; although the structure sizes generated by ID-SPN are orders of magnitude larger than LearnSPN. Both, ID-SPN and LearnSPN, follow a similar structure learning paradigm which alternates between variable clustering and instance clustering based on some distance metric. Recently, a variant of LearnSPN (Molina et al., 2018) was also proposed for hybrid domains with non-parametric leaves. However, these algorithms contain a *correlation threshold* hyperparameter that needs to be properly tuned to select a single best variable partition at each iteration. As a result, the SPN structures are sensitive to the setting of this hyperparameter.

Another structure learning algorithm proposed was LearnPSDD (Liang et al., 2017), however it only learns distributions over the observed variables and must be extended with structural EM to obtain hidden variables. Furthermore, LearnPSDD creates a single partition of the variables when creating product nodes, which means that correlations between the subsets of variables are necessarily broken. In contrast, Prometheus learns a distribution over both observed and hidden variables in a single algorithm. Additionally, Prometheus generates multiple partitions of the variables corresponding to multiple product nodes under each sum node, which allows more correlations to be captured. Another algorithm called Merged-L-SPN

(Rahman and Gogate, 2016) merges subtrees in a separate procedure while Prometheus constructs merged subtrees automatically in its structure learning technique. Prometheus also works with continuous data while EM-LearnPSDD and Merged-L-SPN do not.

In (Hsu et al., 2017), the authors proposed an online structure learning algorithm for continuous variables called oSLRAU. While, oSLRAU can indeed work for streaming data, the presence of two hyper-parameters (*correlation threshold* and *maximum variables in leaf nodes*) means that its performance is sensitive to those hyperparameters that cannot be fine tuned unless oSLRAU is ran in an offline fashion. Furthermore, since oSLRAU processes data in mini-batches, information is lost as earlier batches are forgotten unless multiple passes over the data are performed. This impacts the quality of the SPN structures produced by oSLRAU.

Another model called Infinite Sum-Product Trees (iSPT) (Trapp et al., 2016) was proposed recently as a non-parametric Bayesian extension of SPNs. The accompanying structure learning technique generates all possible partitions of the variables as children of sum nodes. Unfortunately, this approach is not scalable since the number of possible partitions scales as the Stirling number of the second kind.

We will demonstrate that Prometheus is applicable without changes to both discrete and continuous domains. In fact, it can also be applied with ease to hybrid domains (datasets with both continuous and discrete variables). Prometheus also eliminates the sensitive correlation threshold hyper-parameter used extensively in previous algorithms for variable partitioning. This allows Prometheus to stay robust in low data regimes since it captures correlations adequately. Additionally, this makes the algorithm easy to use by practitioners since Prometheus does not require hyper-parameter tuning for different application domains. Furthermore, most existing structure learning techniques learn SPNs with a tree structure and typically maintain a correlation matrix of size $O(d^2)$ where d is the number of features. This is problematic for high dimensional data such as images. Here, we also propose a $O(d(\log d)^2)$ approximation of Prometheus for scalability and successfully demonstrate that the scalable approximation of Prometheus gives reasonable performance on high dimensional datasets including the best reported performance by an SPN on the MNIST image dataset.

3. Background

A sum-product network is a probabilistic graphical model that specifies a joint distribution over a set of variables $\mathbf{X} = \{X_1, X_2, \dots, X_d\}$. An SPN is a directed acyclic graph where interior nodes are either sum nodes or product nodes. The leaves are univariate distributions over the variables X_i . Each edge emanating from a sum node is parametrized by a non-negative weight. Every node in an SPN encodes an unnormalized marginal distribution over a subset of the variables \mathbf{X} . Let \mathbf{x} be the assignment for the random variables \mathbf{X} and $P_N(\mathbf{x}|\mathbf{w})$ be the unnormalized probability associated with each node N given an input \mathbf{x} and weights \mathbf{w} . $P_N(\mathbf{x}|\mathbf{w})$ can be calculated recursively as follows:

$$P_N(\mathbf{x}|\mathbf{w}) = \begin{cases} p(\mathbf{X}_j = \mathbf{x}_j) & \text{if } d \text{ is a leaf node over } X_j \\ \prod_{j \in ch(N)} P_j(\mathbf{x}|\mathbf{w}) & \text{if } N \text{ is a product node} \\ \sum_{j \in ch(N)} w_{Nj} P_j(\mathbf{x}|\mathbf{w}) & \text{if } N \text{ is a sum node} \end{cases}$$

where $ch(N)$ are the children of the node N and w_{Nj} is the edge weight for the sum node N and its child node j . An SPN can be used to encode a joint distribution over \mathbf{X} , which is defined by the graphical structure and the weights. The probability of the assignment $\mathbf{X} = \mathbf{x}$ is equal to the value at the root of the SPN with input \mathbf{x} divided by the normalization constant $P_{\text{root}}(\emptyset|\mathbf{w})$ (where \emptyset denotes an empty variable assignment, which implies that all the variables at the leaves are marginalized). That is,

$$P(\mathbf{X} = \mathbf{x}) = \frac{P_{\text{root}}(\mathbf{x}|\mathbf{w})}{P_{\text{root}}(\emptyset|\mathbf{w})} \quad (1)$$

The marginal probability of a partial assignment $\mathbf{Y} = \mathbf{y}$ where $\mathbf{Y} \subseteq \mathbf{X}$ can also be computed using Eq. 1. Furthermore, conditional probabilities can also be computed using an SPN by evaluating two partial assign-

ments:

$$P(\mathbf{Y} = \mathbf{y} | \mathbf{Z} = \mathbf{z}) = \frac{P(\mathbf{Y} = \mathbf{y}, \mathbf{Z} = \mathbf{z})}{P(\mathbf{Z} = \mathbf{z})} \quad (2)$$

$$P(\mathbf{Y} = \mathbf{y} | \mathbf{Z} = \mathbf{z}) = \frac{P_{\text{root}}(\mathbf{y}, \mathbf{z} | \mathbf{w})}{P_{\text{root}}(\mathbf{z} | \mathbf{w})} \quad (3)$$

Therefore, in an SPN, joint, marginal and conditional queries can all be answered by two network evaluations and hence, exact inference takes linear time with respect to the size of the network. Alternatively, Bayesian and Markov networks may take exponential time in the size of the network.

An SPN is said to be *valid* when it represents a distribution. Eq. 1 and Eq. 3 can then be used to answer inference queries correctly (Poon and Domingos, 2011). *Decomposability* and *completeness* are sufficient conditions that ensure validity (Darwiche, 2003; Poon and Domingos, 2011). The *scope* of a node d is the set of all variables that appear in the leaves of the sub-SPN rooted at d . An SPN is said to be *decomposable* when each product node has children with disjoint scopes and it is *complete* when each sum node has children with identical scope.

3.1 Graph Partitioning & Maximum Spanning Tree

Given a graph $G = (V, E)$ with vertex set V and edge set E , the graph partitioning problem is to partition the graph G in to different components having certain desired properties. Graph partitioning problems are NP-hard (Feldmann and Foschini, 2015) and are normally solved using algorithms based on certain heuristics. The Fiduccia-Mattheyses algorithm (Fiduccia and Mattheyses, 1988) and the Kernighan-Lin algorithm (Lin and Kernighan, 1973) are two well-known graph partitioning algorithms for 2-way cuts based on local search strategies. Graph partitioning techniques have been used in various areas (Buluç et al., 2016) including task-scheduling, circuit design and computing. Recently, graph partitioning has received a lot of attention due to its use in clustering methods, an application relevant to this work as well. In data clustering, the objective is to partition the data points in to disjoint sets such that instances in the same cluster are similar and the ones in different clusters are dissimilar. Graph partitioning based clustering algorithms incorporate these properties easily.

In this work, we partition a graph by constructing a Maximum Spanning Tree (MST) and severing the minimal edge in it to create a partition. A maximum spanning tree (MST) is a set of edges of a connected weighted undirected graph connecting all the vertices together without cycles and having the maximum total edge weight. Classic algorithms include Boruvka’s algorithm (Cormen, 2009), Prim’s algorithm invented by Jarnik, Prim and Dijkstra independently (Cormen, 2009) and Kruskal’s algorithm (Kruskal, 1956). All these algorithms run in $\mathcal{O}(E \log V)$ time where E is the number of edges in G and V is the number of vertices. In (Karger et al., 1995), a linear time randomized algorithm for constructing MSTs was proposed.

4. Prometheus

An SPN encodes a joint distribution over the variables in its scope. Therefore, any structure learning algorithm must capture interactions between the variables adequately to encode this distribution. Prometheus captures these variable interactions both directly by utilizing correlation matrices and indirectly via clustering.

Prometheus is a top-down iterative procedure that alternates between creating mixture distributions via sum-nodes and factored distributions via product nodes. Prometheus captures the direct interactions between variables through an affinity matrix by using a similarity measure. While previous approaches created a partition greedily by using a correlation threshold, Prometheus uses graph partitioning to create a sequence of product nodes with various factored distributions. Subsequently, a sum node induces a mixture distribution over these product nodes. Therefore, at a high level, in Prometheus, the sum nodes are created in two stages - indirectly through clustering the data instances and directly by combining multiple product nodes over the same scope of variables, but with different factorizations. We explain this idea in detail in the subsequent section and demonstrate that Prometheus captures correlations well among variables.

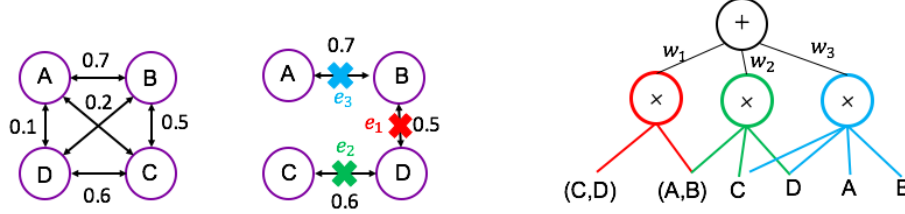


Figure 1: **Schematic for Prometheus.** *Left* : Figure showing the affinity (or correlation matrix) for four variables $\{A, B, C, D\}$. *Middle*: MST for the affinity matrix shown in the first figure. *Right* : The constructed SPN using Prometheus.

4.1 Learning

Consider a d -dimensional dataset \mathcal{D}_n of size n . Let M be the $d \times d$ affinity matrix where $M(i, j)$ is a similarity measure between variables x_i and x_j . For simplicity, we will consider the similarity measure to be the absolute correlation between the variables, i.e.,

$$|\rho_{x_i x_j}| = \left| \frac{\sum_{k=1}^n (x_i^{(k)} - \bar{x}_i)(x_j^{(k)} - \bar{x}_j)}{\sqrt{\sum_{k=1}^n (x_i^{(k)} - \bar{x}_i)^2} \sqrt{\sum_{k=1}^n (x_j^{(k)} - \bar{x}_j)^2}} \right| \quad \text{where } \bar{x}_i = \frac{\sum_{k=1}^n x_i^{(k)}}{n} \text{ and } \bar{x}_j = \frac{\sum_{k=1}^n x_j^{(k)}}{n}$$

M induces a corresponding undirected weighted complete graph G . Each node in G corresponds to one of the d variables, whereas an edge between node i and node j has a weight equal to $M(i, j)$ denoting the correlation between x_i and x_j . Let T be the *maximum spanning tree* over G such that for each cut-set, the highest edge is in T . The edge list in T is sorted in an ascending order say $\{e_{(1)}, \dots, e_{(d)}\}$ according to the weights associated with them.

Creating a product node in an SPN requires disjoint sets \mathcal{A} and \mathcal{B} of variables chosen from amongst the nodes of G such that the edges connecting \mathcal{A} and \mathcal{B} carry low weights. (ID-SPN and LearnSPN use a correlation threshold to choose these disjoint sets \mathcal{A} and \mathcal{B} greedily). This paradigm closely resembles the framework of spectral clustering and graph partitioning. Prometheus finds multiple decompositions instead of a single best decomposition, utilizing a sum-node to induce a mixture over these decompositions.

To create a product node, we begin by removing $e_{(1)}$ – the weakest edge in the maximum spanning tree T – severing the weakest connection between two disjoint sets. This creates a graph with two connected components and a product node P_1 is generated over these two subsets of variables. Next, $e_{(2)}$ is removed, creating three disjoint sets and another product node P_2 is generated over these three disjoint sets. We continue this procedure, removing one edge at a time, and generating a product node over the resulting subsets of variables. At the end, when we remove $e_{(d-1)}$, we recover a fully factored distribution. Therefore, at each level, we obtain a set of decompositions based on graph partitioning. All these decompositions have the same scope and a mixture over these is induced by joining them using a sum-node.

In Fig. 1, we give a schematic for Prometheus on four variables $\{A, B, C, D\}$. The left figure shows the correlations amongst all the variables. The middle figure shows the resultant MST. Prometheus then proceeds as follows : it first severs the edge e_1 which has the lowest weight in the MST. A product node is created with the two partitions - (A, B) and (C, D) - resulting from severing e_1 . This step is denoted by red in the figure. Next, e_2 is severed resulting in three partitions - (A, B) , C , and D - and another product node over these variables is created (denoted by green). Finally, the last edge is cut, resulting in a fully factored product node with variables A, B, C, D (denoted with color blue).

Notice that Prometheus does not use any correlation threshold to determine which correlations can be broken. In the creation of each product node, we simply sever the next weakest edge, which increases the number of subsets of variables in the resulting partition. Since it is not clear which partition is best, we simply combine the product nodes corresponding to each partition into a mixture. While the performance of

other algorithms is sensitive to the choice of the correlation threshold, Prometheus remains robust in low data regimes due to capturing the correlations adequately (see Section 5).

In Prometheus, we view the affinity matrix M as a graph G and create a set of decompositions by eliminating edges one at a time from the maximum spanning tree T of G . The paradigm of creating these decompositions by eliminating edges is similar to graph partitioning. Indeed, our approach mirrors classical graph bi-partitioning algorithms (Fiduccia and Mattheyses, 1988; Lin and Kernighan, 1973). These approaches have been used earlier to solve convex problems with local structures (Friesen and Domingos, 2015). Since, SPNs perform a similar search for local structures in the data through sum nodes, it is natural to consider the idea of graph partitioning for structure learning. However, our approach of graph partitioning is different from classical approaches. In all the classical algorithms, a cut is examined in terms of the sum of the weights of the edges being cut off, whereas we examine cuts in terms of the maximal edge being severed. This is because in an SPN, a product node should ideally be able to split the variables into several subsets. For the max-flow min-cut problem (Papadimitriou and Steiglitz, 1998), the bi-partitioning case is efficiently solvable. However, graph partitioning algorithms face issues when examining a k -way cut, which is not solvable in polynomial time by a deterministic algorithm when k is an input. Therefore, the problem we wish to solve is a k -way min cut problem for cutting edges so that the graph splits into k components and the edges that are cut have the smallest sum (or smallest maximum in our case). We achieve such a k -way partitioning by greedily eliminating minimal edges one at a time in the maximum spanning tree.

Algorithm 1: Prometheus(\mathcal{D}, X)

Input: A dataset \mathcal{D} and variables X
Result: An SPN encoding a distribution over X using \mathcal{D}
if $|X| = 1$ **then**
 induce univariate distribution using \mathcal{D}
end
else
 Partition \mathcal{D} into clusters \mathcal{D}_i and create a childless sum node N // *The clustering method used is K -means, with uniform initial weights ;*
 for each \mathcal{D}_i **do**
 $M_i \leftarrow$ affinity matrix based on \mathcal{D}_i ;
 $T_i \leftarrow$ MST of undirected graph G_i from M_i ;
 while T_i contains edges **do**
 $e_{min} \leftarrow \text{argmin}_{e \in T_i} \text{weight}(e)$ // find weakest edge ;
 $T_i \leftarrow T_i - \{e_{min}\}$ // severe weakest edge;
 create product node P and add P as child of N ;
 $\pi_P \leftarrow$ set of scopes that partition X based on edges severed so far;
 $\mathcal{D}_P \leftarrow \mathcal{D}_i$;
 end
 end
 for each $P \in \text{children}(N)$ **do**
 for each $\text{scope} \in \pi_P$ **do**
 if S_{scope} does not already exist **then**
 $\mathcal{D}_{P, \text{scope}} \leftarrow$ restrict dataset \mathcal{D}_P to variables in scope ;
 $S_{\text{scope}} \leftarrow \text{Prometheus}(\mathcal{D}_{P, \text{scope}}, \text{scope})$;
 end
 add sum node S_{scope} as child of P ;
 end
 end
end

Alg. 1 describes the pseudo-code of Prometheus. The algorithm proceeds by first clustering the samples in the dataset \mathcal{D} into m clusters \mathcal{D}_i , $i \in [m]$. These data clusters have the same scope of variables and a sum node is induced over these clusters to create a mixture. Next, for each cluster i and the associated dataset \mathcal{D}_i , the algorithm, creates a set of product nodes by greedily severing edges in the MST T_i . At each step, the algorithm removes the next weakest edge e_{min} and creates $T_i \leftarrow T_i - \{e_{min}\}$. A product node P is then created with children corresponding to the connected components of T_i . Note that this MST-based method automatically produces a DAG structure with shared subtrees. Each time an edge is severed in the MST, the corresponding partition of variables π_P contains some scopes that are the same as in previous partitions. Instead of creating a new sum node S_{scope} for each scope, we reuse a previously created S_{scope} when possible. Getting a DAG structure with subtree sharing is a nice benefit of using MST based partitioning. Subsequently, a sum node is created over all the product nodes returned from all the clusters. Therefore, Prometheus creates sum-nodes in two steps, one via clustering (capturing interactions indirectly) and another by inducing a sum-node over all decompositions (thereby capturing interactions directly). Prometheus then collapses these two adjacent levels of sum-nodes into one level of sum-nodes. Finally, the algorithm continues by recursing over each child of every product node created in the previous step. If the number of variables in a node decreases to one, the algorithm induces a univariate distribution over it. Furthermore, Prometheus can also incorporate multivariate distributions as leaves (similar to ID-SPN) wherein the algorithm would stop when the number of variables in a node decreases to the specified size of the multivariate leaves.

4.2 Time Complexity Analysis

The running time complexity of the algorithm is linear in the number of data points n , but it is $\mathcal{O}(d^2)$ in the number of dimensions of the data. This is due to the fact that creating the affinity matrix M requires quadratically many operations. Furthermore, since Prometheus directly creates a DAG with subtree sharing, at each layer we construct $\mathcal{O}(d)$ many new children using our graph partitioning method. Therefore, the size of the structure is $\mathcal{O}(d^2)$. In Prometheus, the data is partitioned using K-Means clustering, which has a time complexity of $\mathcal{O}(ndKI)$ where K is the number of clusters and I is the number of iterations. Therefore, the total time complexity of Prometheus is $\mathcal{O}(nd^2 + ndKI)$.

We also propose a sampling approximation of the algorithm. In this variant, instead of constructing a full correlation (or affinity) matrix M , we build a matrix \hat{M} with $d \log d$ many values. For each variable x_i , we sample $\log d$ variables and evaluate the correlation of x_i with these variables. In this manner, we construct an incomplete matrix \hat{M} . All the other entries are set to 0 so that they do not affect the construction of the maximum spanning tree \hat{T} . The complexity for this approximate MST construction is $\mathcal{O}(d(\log d)^2)$ and the total time complexity of Prometheus is $\mathcal{O}(nd(\log d)^2 + ndKI)$. We perform experiments on high dimensional datasets including MNIST using this sampling algorithm and show that it gives reasonable performance (see Section 5).

5. Experiments and Results

We now provide an empirical evaluation of Prometheus. We compare Prometheus' performance to previous structure learning algorithms such as ID-SPN (Rooshenas and Lowd, 2014), LearnSPN (Gens and Domingos, 2013), DV (Dennis and Ventura, 2015), CCCP (Zhao et al., 2016), Merged-L-SPN (Rahman and Gogate, 2016) and EM-LearnPSDD (Liang et al., 2017) on 20 real world discrete benchmark datasets. Furthermore, for continuous domains we compare Prometheus to oSLRAU (Hsu et al., 2017), iSPT (Trapp et al., 2016), SRBMs (Salakhutdinov and Hinton, 2009), oBMM (Jaini et al., 2016) and Gaussian Mixture Models on 10 real world datasets. The experiments show that Prometheus achieves state-of-the-art performance on 19 (out of 30) datasets among pure learning techniques (that are not boosted by ensemble learning). We also perform experiments on the MNIST image data using the sample based approximation of Prometheus and report the best performance by an SPN on it. We also demonstrate that Prometheus is robust in low-data regimes by performing experiments on a subset of the above datasets with low data.

Experimental Setup

We performed experiments for both continuous and discrete domains on real world datasets that have previously been used to compare the performance of SPNs. For each of the 30 real world datasets, we report the average log-likelihood as a performance measure using 10-fold cross validation. The Wilcoxon signed rank test (Wilcoxon, 1950) with p -value < 0.05 is used to determine statistical significance.

For the continuous domains, Table 1 reports the average log likelihood for 7 datasets achieved by SRBMs (Salakhutdinov and Hinton, 2009), oSLRAU (Hsu et al., 2017), GMMs with oBMM (Jaini et al., 2016) for parameter learning and Prometheus in which the parameters are learned exactly as per oSLRAU’s parameter updates. Prometheus outperforms the other algorithms on 4 (out of 7) datasets. The results for SRBMs, oSLRAU and GMMs with oBMM were recorded in (Hsu et al., 2017) for online stream learning only (i.e., single pass through the data). We re-implemented oSLRAU and oBMM, and report offline results (i.e., multiple passes through the data) in Table 1 that are better than the online results. On one dataset, SRBM performed better although Prometheus achieved the best performance by an SPN. SRBMs were trained with 3 hidden layers of binary nodes where the first two layers have the same number of nodes as the dimensionality of the data and the third layer has four times as many nodes. The log-likelihoods of SRBMs are obtained by Gibbs sampling. In Table 2, we provide a comparison between Prometheus, iSPTs (Trapp et al., 2016) and GMMs. The comparison was done on the same datasets used by the authors of iSPT. Prometheus performs better on all three datasets even though iSPT considers exponentially many partitions while creating each product node.

Continuous Datasets				
Data set (Attributes)	SRBMs	oSLRAU	GMMs w oBMM	Prometheus
Abalone (8)	-2.28↓	-0.94↓	-1.17↓	-0.85
CA (22)	-4.95↓	21.19↓	3.42↓	27.82
Quake (4)	-2.38↓	-1.21↑	-3.76↓	-1.50
Sensorless(48)	-26.91↓	60.72↓	8.56↓	62.03
Banknote(4)	-2.76↓	-1.39↑	-4.64↓	-1.96
Flowsize (3)	-0.79↓	15.32↓	5.72↓	18.03
Kinematics(8)	-5.55↑	-11.13↓	-11.2↓	-11.12

Table 1: Average log-likelihoods for SRBMs, oSLRAU, GMMs with oBMM and Prometheus under 10-fold cross validation. ↑(or ↓) indicates that the corresponding algorithm has better (or worse) log-likelihood than Prometheus under the Wilcoxon signed rank test with p -value < 0.05 . Note: some values are positive since the log-likelihood of continuous variables can be positive.

Continuous Datasets			
Data set	iSPT	GMM	Prometheus
Iris	-3.744↓	-3.943↓	-1.06
Old Faithful	-1.700↓	-1.737↓	-1.48
Chemical Diabetes	-2.879↓	-3.022↓	-2.59

Table 2: Average log-likelihoods for iSPT, GMM and Prometheus under 10-fold cross validation. ↑(or ↓) indicates that the corresponding algorithm has better (or worse) log-likelihood than Prometheus under the Wilcoxon signed rank test with p -value < 0.05 .

For the discrete domains, we compare Prometheus to previous pure structure learning algorithms (without any boost from ensemble learning) like LearnSPN (Gens and Domingos, 2013), ID-SPN (Rooshenas and Lowd, 2014), EM-LearnPSDD (Liang et al., 2017), Merged-L-SPN (Rahman and Gogate, 2016) and CCCP (Zhao et al., 2016). We also compare to DV (Dennis and Ventura, 2015), which is the only previous

structure learning algorithm that directly learns a DAG with subtree sharing (like Prometheus) on 20 benchmark datasets. For all the algorithms, we use the same settings as described in the respective manuscripts. We performed extensive hyper-parameter tuning for each of the above algorithms so as to closely match the best reported results given in their respective manuscripts. The results reported for ID-SPN were obtained using the code released by the authors without changes. However, we were not able to reproduce the exact results as reported in the original manuscript. For Prometheus, we initialized the parameters using Learn-SPN like counts and used CCCP for parameter learning. Table 3 reports the average log-likelihoods for each of the algorithm. Prometheus matched or outperformed the other algorithms on 12 datasets and its performance was almost on par with the best performing algorithm on the remaining 8 datasets. Another advantage of Prometheus is its ease of implementation on varying datasets since Prometheus does not require hyper-parameter tuning. While, this simplicity is not reflected in the reported figures, it gives Prometheus a distinct advantage over other algorithms as a practical tool for real world applications. We also report in the last column of Table 3 the best-to-date results obtained by any technique on these 20 datasets (see (Liang et al., 2017) for more details). These results are usually obtained by embedding a pure structure learning technique within an ensemble learning technique such as bagging. While the results obtained by Prometheus are not as good, note that it is possible to use Prometheus within ensemble learning.

Discrete Datasets								
Data set	Learn-SPN	ID-SPN	CCCP	EM-LearnPSDD	Merged-L-SPN	DV	Prometheus	Best-to-date
NLTCS	-6.10 ↓	-6.05 ↓	-6.03 ↓	-6.03 ↓	-6.07 ↓	-6.07 ↓	-6.01	-5.99
MSNBC	-6.11 ↓	-6.05	-6.05	-6.04	-6.11 ↓	-6.06 ↓	-6.04	-6.04
KDD	-2.23 ↓	-2.15 ↓	-2.13	-2.12	-2.13	-2.16 ↓	-2.12	-2.11
Plants	-12.95 ↓	-12.55 ↑	-12.87 ↓	-13.79 ↓	-12.79 ↑	-13.13 ↓	-12.81	-11.99
Audio	-40.51 ↓	-39.82	-40.02 ↓	-41.98 ↓	-40.87 ↓	-40.13 ↓	-39.80	-39.39
Jester	-53.45 ↓	-52.91 ↓	-52.88 ↓	-53.47 ↓	-53.02 ↓	-53.08 ↓	-52.80	-41.11
Netflix	-57.38 ↓	-56.55	-56.78 ↓	-58.41 ↓	-56.52 ↓	-56.81 ↓	-56.47	-55.71
Accidents	-29.07 ↓	-27.23 ↑	-27.70 ↑	-33.64 ↓	-28.72 ↓	-29.01 ↓	-27.91	-24.87
Retail	-11.14 ↓	-10.94 ↓	-10.92 ↓	-10.81 ↑	-10.92 ↓	-10.97 ↓	-10.87	-10.72
Pumsbstar	-24.58 ↓	-22.55	-24.23 ↓	-33.67 ↓	-26.57 ↓	-28.69 ↓	-22.75	-22.40
DNA	-85.24 ↓	-84.69 ↓	-84.92 ↓	-92.67 ↓	-87.63 ↓	-87.76 ↓	-84.45	-80.03
Kosarek	-11.06 ↓	-10.61	-10.88 ↓	-10.81 ↓	-10.74 ↓	-11.00 ↓	-10.59	-10.52
MSWeb	-10.27 ↓	-9.80 ↑	-9.97 ↓	-9.97 ↓	-9.86	-9.97 ↓	-9.86	-9.22
Book	-36.25 ↓	-34.44	-35.01 ↓	-34.97 ↓	-34.67 ↓	-34.91 ↓	-34.40	-30.18
Movie	-52.82 ↓	-51.55 ↓	-52.56 ↓	-58.01 ↓	-52.74 ↓	-53.28 ↓	-51.49	-51.14
WebKB	-158.54 ↓	-153.3 ↑	-157.49 ↓	-161.09 ↓	-154.62 ↑	-157.88 ↓	-155.21	-150.10
Reuters	-85.98 ↓	-84.39	-84.63	-89.61 ↓	-85.34 ↓	-86.38 ↓	-84.59	-80.66
NewsGroup	-156.61 ↓	-151.6 ↑	-153.20 ↑	-161.09 ↓	-155.12 ↓	-153.63 ↑	-154.17	-150.88
BBC	-249.79 ↓	-252.60 ↓	-248.60	-253.19 ↓	-252.43 ↓	-252.13 ↓	-248.5	-233.26
AD	-27.41 ↓	-40.01 ↓	-27.20 ↓	-31.78 ↓	-17.26 ↑	-26.97 ↓	-23.96	-14.36

Table 3: Average log-likelihoods for LearnSPN, ID-SPN, CCCP, LearnPSDD-EM, Merged-L-SPN, DV and Prometheus under 10-fold cross validation. ↑(or ↓) indicates that the corresponding algorithm better (or worse) log-likelihood than Prometheus under the Wilcoxon signed rank test with p -value < 0.05 . The Best-to-date column reports the results for the best performance reported by any model in the literature. These results are usually obtained by bagging.

In Table 4, we demonstrate the robustness of Prometheus in low data regimes. For this, we chose a subset of continuous datasets considered for this manuscript. We train Prometheus on a fraction of the available training data (between 10% and 40% of the total data). We then compare its performance to oSLRAU, GMMs with oBMM and SRBMs which were trained on the complete training data. The results in Table 4 demonstrate that Prometheus is competitive with oSLRAU, GMMs with oBMM and SRBMs even in low-data regimes and achieves high likelihoods with limited data.

% data	Abalone	Sensorless Drive	CA
10	-1.73	53.92	16.85
15	-1.37	56.23	17.22
20	-1.32	56.76	17.53
25	-1.26	57.4	17.63
30	-1.21	58.01	18.67
35	-1.08	61.47 ↑	22.82 ↑
40	-0.98	61.58 ↑	27.27 ↑
90	-0.85 ↑	62.03 ↑	27.82 ↑
oSLRAU	-0.94	60.72	21.19
GMMs with oBMM	-1.17	8.56	3.42
SRBMs	-2.28	-26.91	-4.95

Table 4: Average log-likelihoods for oSLRAU, GMMs with oBMM, SRBMs and Prometheus under 10-fold cross validation. ↑(or ↓) indicates that Prometheus has better (or worse) log-likelihood than the best performing algorithm among oSLRAU, GMMs with oBMM and SRBM under the Wilcoxon signed rank test with p -value < 0.05 . oSLRAU, GMMs with oBMM and SRBMs were trained on the complete dataset while Prometheus was trained on a fraction of the training data as indicated in column one.

MNIST dataset

We now report the performance of Prometheus on the MNIST dataset achieved using both versions of Prometheus proposed in this paper - the normal version which constructs a MST using the complete affinity matrix ($\mathcal{O}(d^2)$) and the scalable version, which uses sampling to construct an incomplete affinity matrix and reduces the complexity to $\mathcal{O}(d(\log d)^2)$.

An additional aim is to demonstrate that an SPN technique designed to be applied directly on raw pixel data of an image gives reasonable performance. The previous best results obtained by SPNs were using a discriminative learning approach (Gens and Domingos, 2012). However, in that paper, the authors utilize CNN like pre-processing as described in (Coates et al., 2011) before training the SPN on the extracted features. Our algorithm, on the other hand, does not utilize CNNs for feature extraction and learns an SPN on raw image pixels. We train a generative SPN for each digit by operating on reduced graphs obtained via sampling. This is achieved by sampling edges between pixels that are located symmetrically about the center of the image, the x-axis and the y-axis. We sample points according to a fixed probability for each pixel from these areas to construct the max-spanning tree.

In Table 5, we report the results for percentage accuracy achieved by different structure learning algorithms for SPNs on the MNIST dataset. We compare the performance of Prometheus with ID-SPN (Rooshenas and Lowd, 2014), SPN-Gens (Gens and Domingos, 2012) a discriminative structure learning algorithm that works on features extracted using CNNs, DSPN-SVD and SPN-SVD (Adel et al., 2015), an SVD based structure learning algorithm with discriminative and generative variants respectively. Furthermore, in (Hartmann, 2014)(SPN-TH), the authors use a similar CNN like pre-processing as proposed in (Gens and Domingos, 2012) before training an SPN over the extracted features and reported an accuracy of 98.34%. Prometheus outperforms all these structure learning algorithms (both generative and discriminative) on MNIST image data by directly training on the raw pixels. This is the best result achieved by an SPN on MNIST data to our knowledge.

DSPN-SVD	SPN-SVD	SPN-TH	SPN-Gens	ID-SPN	Prometheus (scalable)	Prometheus (normal)
97.6%	85%	98.34%	81.8%	84.4%	98.1%	98.37%

Table 5: Percentage accuracy on MNIST dataset.

6. Conclusion

We proposed Prometheus, an MST based partitioning algorithm for directly learning directed acyclic graph structures with subtree sharing for SPNs. We performed extensive comparisons with previous structure learning algorithms on 30 real world datasets from varied domains and achieved state of the art performance on a majority (19) of them. We also proposed an approximate scalable version of Prometheus for high-dimensional domains based on sampling and reported the best accuracy by an SPN on an MNIST dataset by training on raw image pixels. The idea of graph partitioning methods for structure learning helps Prometheus to directly learn DAG structures with subtree sharing and eliminates the sensitive *correlation threshold* hyper-parameter used in previous algorithms, making it robust in low-data regimes.

In the future, it would be interesting to adapt Prometheus to hybrid domains with non-parametric leaf distributions (Molina et al., 2018). Additionally, compression techniques can be incorporated in Prometheus to prune the structures during learning.

References

- T. Adel, D. Balduzzi, and A. Ghodsi. Learning the structure of sum-product networks via an SVD-based algorithm. In *UAI*, pages 32–41, 2015.
- A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.
- W.-C. Cheng, S. Kok, H. V. Pham, H. L. Chieu, and K. M. A. Chai. Language modeling with sum-product networks. In *Annual Conf. of the Int. Speech Communication Association*, 2014.
- A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.
- T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.
- A. Darwiche. A differential approach to inference in Bayesian networks. *JACM*, 50(3):280–305, 2003.
- A. Dennis and D. Ventura. Learning the architecture of sum-product networks using clustering on variables. In *NIPS*, 2012.
- A. W. Dennis and D. Ventura. Greedy structure search for sum-product networks. In *IJCAI*, volume 15, pages 932–938, 2015.
- A. E. Feldmann and L. Foschini. Balanced partitions of trees and applications. *Algorithmica*, 71(2):354–376, 2015.
- C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Papers on Twenty-five years of electronic design automation*, pages 241–247. ACM, 1988.
- A. L. Friesen and P. Domingos. Recursive decomposition for nonconvex optimization. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 253–259, 2015.
- R. Gens and P. Domingos. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pages 3239–3247, 2012.
- R. Gens and P. Domingos. Learning the structure of sum-product networks. In *ICML*, pages 873–880, 2013.
- T. Hartmann. *Friedrich-Wilhelms-Universität Bonn*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2014.

- W. Hsu, A. Kalra, and P. Poupart. Online structure learning for sum-product networks with Gaussian leaves. *arXiv preprint arXiv:1701.05265*, 2017.
- P. Jaini, A. Rashwan, H. Zhao, Y. Liu, E. Banijamali, Z. Chen, and P. Poupart. Online algorithms for sum-product networks with continuous variables. In *Conference on Probabilistic Graphical Models*, pages 228–239, 2016.
- D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995.
- J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.
- S.-W. Lee, M.-O. Heo, and B.-T. Zhang. Online incremental structure learning of sum-product networks. In *NIPS*, pages 220–227, 2013.
- Y. Liang, J. Bekker, and G. Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.
- S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.
- A. Molina, A. Vergari, N. Di Mauro, S. Natarajan, F. Esposito, and K. Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. 2018.
- C. H. Papadimitriou and K. Steiglitz. 6.1 the max-flow, min-cut theorem. *Combinatorial optimization: Algorithms and complexity*, Dover, pages 120–128, 1998.
- R. Peharz, B. C. Geiger, and F. Pernkopf. Greedy part-wise learning of sum-product networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 612–627. Springer, 2013.
- H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.
- T. Rahman and V. Gogate. Merging strategies for sum-product networks: From trees to graphs. In *UAI*, 2016.
- A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *ICML*, pages 710–718, 2014.
- R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In *AISTATS*, pages 448–455, 2009.
- M. Trapp, R. Peharz, M. Skowron, T. Madl, F. Pernkopf, and R. Trappl. Structure inference in sum-product networks using infinite sum-product trees. In *NIPS Workshop on Practical Bayesian Nonparametrics*, 2016.
- A. Vergari, N. Di Mauro, and F. Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *ECML-PKDD*, pages 343–358. 2015.
- F. Wilcoxon. Some rapid approximate statistical procedures. *Annals of the New York Academy of Sciences*, pages 808–814, 1950.
- H. Zhao, M. Melibari, and P. Poupart. On the relationship between sum-product networks and Bayesian networks. In *ICML*, 2015.
- H. Zhao, P. Poupart, and G. J. Gordon. A unified approach for learning the parameters of sum-product networks. In *Advances in Neural Information Processing Systems*, pages 433–441, 2016.