Merging Strategies for Sum-Product Networks: From Trees to Graphs

Tahrima Rahman and **Vibhav Gogate**
Department of Computer Science
The University of Texas at Dallas
Richardson, TX 75080, USA.
{tahrima.rahman,vibhav.gogate}@utdallas.edu

Abstract

Learning the structure of sum-product networks (SPNs) – arithmetic circuits over latent and observed variables – has been the subject of much recent research. These networks admit linear time exact inference, and thus help alleviate one of the chief disadvantages of probabilistic graphical models: accurate probabilistic inference algorithms are often computationally expensive. Although, algorithms for inducing their structure from data have come quite far and often outperform algorithms that induce probabilistic graphical models, a key issue with existing approaches is that they induce tree SPNs, a small, inefficient sub-class of SPNs. In this paper, we address this limitation by developing post-processing approaches that induce graph SPNs from tree SPNs by merging similar sub-structures. The key benefits of graph SPNs over tree SPNs include smaller computational complexity which facilitates faster online inference, and better generalization accuracy because of reduced variance, at the cost of slight increase in the learning time. We demonstrate experimentally that our merging techniques significantly improve the accuracy of tree SPNs, achieving state-of-the-art performance on several real world benchmark datasets.

1 INTRODUCTION

Probabilistic graphical models [8, 17] such as Bayesian and Markov networks are routinely used in a wide variety of application domains such as computer vision and natural language understanding for modeling and reasoning about uncertainty. However, exact inference in them – the task of answering queries given a model – is NP-hard in general and computationally intractable for most real-world models. As a result, approximate inference algorithms such as loopy belief propagation and Gibbs sampling are widely used in practice. However, they can often yield highly inaccurate and high variance estimates, leading to poor predictive performance.

One approach to tackle the inaccuracy and unreliability of approximate inference is to learn so-called tractable models from data. Examples of such models include thin junction trees [2], arithmetic circuits (ACs) [7], cutset networks [25], probabilistic sentential decision diagrams [16], AND/OR decision diagrams [9, 21] and sum-product networks [23]. Inference in these models is polynomial (often linear) in the size of the model and therefore the complexity and accuracy of inference is no longer an issue. In other words, once an accurate model is learned from data, predictions are guaranteed to be accurate.

In this paper, we focus on the NP-hard problem of learning both the structure and parameters of sum-product networks (SPNs) from data. At a high level, an SPN is a rooted directed acyclic graph that represents a joint probability distribution over a large number of random variables, both observed and latent. It has two types of internal nodes: sum-nodes which express conditioning or splitting over latent or observed variables and product nodes which represent decomposition of variables into independent components. Leaf nodes represent simple distributions over observed variables (e.g., uniform distribution, univariate distributions, etc.). The key advantage of SPNs and other equivalent representations such as ACs [1] over thin-junction trees is that they can be much compact and never larger than the latter. This is because they take advantage of various fine-grained structural properties such as determinism, context-specific independence, dynamic variable orderings and caching (cf. [7, 9, 4, 13]). For instance, in some cases, they can represent high-treewidth junction trees using only a handful of sum and product nodes [23].

The literature abounds with algorithms for learning the structure of SPNs and ACs from data, starting with the

[1]The equivalence between ACs and SPNs was shown by Rooshenas and Lowd [26]. Thus, algorithms for learning ACs can be used to learn SPNs and vice versa.

work of Lowd and Domingos [19] who proposed to learn ACs over observed variables by using the AC size as a learning (inductive) bias within a Bayesian network structure learning algorithm, and then compiling the induced Bayesian network to an AC. Later Lowd and Rooshenas [20] extended this algorithm to learn a Markov network having small AC size. The latter performs much better in terms of test set log likelihood score than the former because of the increased flexibility afforded by the undirected Markov network structure.

A limitation of the two aforementioned approaches for learning ACs is that they do not use latent variables; it turns out that their accuracy can be greatly improved using latent variables. Unfortunately, the parameter learning problem (a sub-step in structure learning) – the problem of learning the weights or probabilities of a given SPN structure – is much harder in presence of latent variables. In particular, the optimization problem is non-convex, which necessitates the use of algorithms such as gradient descent and expectation maximization that only converge to a local minima. However, since learning is often an offline process, this increase in complexity is often not a big issue.

The first approach for learning the structure of SPNs having both latent and observed variables is due to Gens and Domingos [11]. An issue with this approach is that it learns only directed trees instead of (directed acyclic) graphs and as a result is unable to fully exploit the power and flexibility of SPNs. To address this limitation, Rahman et al. [25], Vergari et al. [28] and Rooshenas and Lowd [26] proposed to learn a graph SPN over observed variables while Dennis and Ventura [10] proposed to learn a graph SPN over latent variables. A drawback of these approaches is that they are unable to learn a graph SPN over both observed and latent variables. In this paper, we address this limitation.

The main idea in our approach is as follows. We first learn a tree SPN over latent and observed nodes using standard algorithms, and then convert the tree SPN to a graph SPN by processing the SPN in a bottom-up fashion, merging two sub-SPNs if the distributions represented by them are similar and defined over the same variables. To convert this idea into a general-purpose algorithm, we have to solve two problems: (1) how to find similar sub-SPNs, and (2) how to merge them into one sub-SPN. Both problems are computationally expensive to solve and therefore we develop approximate algorithms for solving them, which is the main contribution of this paper.

The second contribution of this paper is a thorough experimental evaluation of our proposed merging algorithms on 20 benchmark datasets, all of which were used in several previous studies. Our experiments clearly show that merging always improves the performance of tree SPNs, measured in terms of test-set log-likelihood score and prediction time. We also experimentally compared bagged ensembles of graph SPNs with state-of-the-art approaches such as ensembles of cutset networks [24], sum-product networks with direct and indirect interactions [26], sum-product networks learned via the SVD-based approach[1], arithmetic circuits with Markov networks [20], and mixtures of cutset networks [25] on the same datasets, and found that our new approach yields better test-set log likelihood score on 8 out of the 20 datasets with two ties. This clearly demonstrates the power of our new merging algorithms.

The rest of the paper is organized as follows. In the next section, we present background on SPNs, related work as well as a generic algorithm for learning tree SPNs. Section 3 describes powerful merging approaches for converting an arbitrary tree SPN to a graph SPN. Experimental results are presented in section 4 and we conclude in section 5.

2 BACKGROUND

Any (discrete) probability distribution over a set of variables $\mathbf{V}$ can be expressed using an arithmetic circuit (AC) [7] or a sum-product network (SPN) [23].[2] The key benefit of SPNs over conventional uncertainty representations such as Bayesian and Markov networks is that in SPNs, common probabilistic inference tasks such as maximum-a-posteriori (MAP) and posterior marginal (MAR) estimation can be solved in time and space that scales linearly with the size of the representation. In Bayesian and Markov networks, these tasks are known to be NP-hard in general (cf. [27]). The caveat is that SPNs can be exponentially larger than Bayesian and Markov networks; they are often compiled from the latter by running exact probabilistic inference techniques such as variable elimination [4] and AND/OR search [21], in order to facilitate faster online inference. Formally,

Definition 1. An SPN [23] is recursively defined as follows:

1. A tractable univariate distribution is an SPN;
2. A product of SPNs defined over different variables is an SPN; and
3. A weighted sum of SPNs with the same scope variables is an SPN.

An SPN can be expressed as a rooted directed acyclic graph with univariate distributions as leaves, sums and products as internal nodes, and the edges from a sum node to its

[2] SPNs used in this paper are equivalent to ACs (as well as AND/OR decision diagrams [21]) defined over latent and observed variables. However, in order to be consistent, we will use the term SPNs throughout the paper. We will distinguish between two types of SPNs: SPNs defined over only observed variables and SPNs defined over both observed and latent variables.

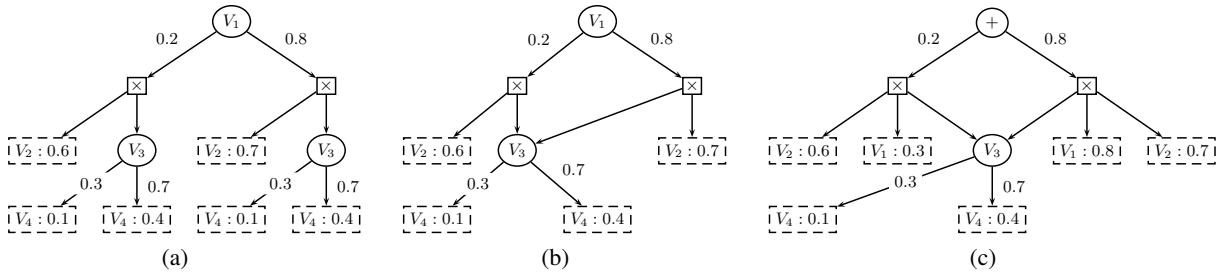


Figure 1: Three example SPNs over variables $\{V_1, V_2, V_3, V_4\}$. We are assuming that all variables are binary and take values from the domain $\{0, 1\}$. Leaf nodes express univariate distributions. For example, the node $V_2 : 0.6$ expresses the probability distribution $P(V_2 = 1) = 0.6$. Sum nodes are labeled either by a variable which denotes conditioning over the variable or by a $+$ sign which denotes that the sum node is latent. All left (right) arcs emanating from a sum node correspond to an assignment of 1 (0) to the labeled variable. Product nodes are labeled by $\times$. (a) Tree SPN (SPN which is a rooted directed acyclic tree) that decomposes according to a tree Markov network $V_4 - V_3 - V_1 - V_2$. (b) Graph SPN that is equivalent to the tree SPN given in (a) obtained by merging identical sub-trees. (c) Graph SPN over latent and observed variables.

children labeled with the corresponding weights. An SPN represents a (normalized) probability distribution when the weights attached to each sum node sum to one. Any unnormalized SPN can be normalized in linear time.

As mentioned earlier, we will distinguish between two types of SPNs: SPNs defined only over observed variables and SPNs defined over both latent and observed variables. The two have different representation powers with the latter being more general and therefore more powerful than the former. For SPNs having only observed variables, each sum node represents a split (conditioning) over a variable and is therefore labeled by the corresponding variable. For SPNs having both latent and observed variables, each sum node can represent either a split over an observed or a latent variable. The splits over the observed variables are represented the usual way while sum nodes that split over the latent variables are labeled by the "+" sign.

Example 2.1. Fig. 1 shows three SPNs over four variables $\{V_1, V_2, V_3, V_4\}$. The two SPNs on the left are defined over only observed variables while the SPN on the right is defined over both latent and observed variables. The graph SPN shown in Fig. 1(b) is obtained from the tree SPN shown in Fig. 1(a) by merging identical sub-SPNs.

2.1 LEARNING SPNs

In this paper, we focus on top-down approaches that directly learn the structure of SPNs from data. Instead of learning Bayesian and Markov networks and then compiling them into SPNs (this is the approach used in [19, 20]), the key advantage of this direct approach is that the size of the SPN can be controlled in a straight-forward manner, which is typically bounded from above by the data size.

Algorithm 1 shows a generic recursive learning algorithm for learning tree SPNs from data, which is loosely based on the algorithm proposed by Gens and Domingos [11]. The algorithm has three steps: base case, decomposition and splitting. In the base case, if the conditions for learning the base model are satisfied, for example, when the size of the training data is small or only one variable remains, then the algorithm learns the corresponding trivial distribution and terminates the recursion. In the decomposition step, the algorithm tries to partition the variables into roughly independent components $\mathbf{V_j} \subseteq \mathbf{V}$ such that $P(\mathbf{V}) = \prod_j P(\mathbf{V_j})$ and recurses on each component, inducing a product node. If neither the base case nor the conditions for the decomposition step are satisfied, then the algorithm partitions the training instances into clusters of multiple instances, inducing a sum node, and recurses on each part.

Algorithm 1: LEARNSPN($\mathbf{T}$,$\mathbf{V}$)

Input: Set of Training Instances $\mathbf{T}$ and set of variables $\mathbf{V}$
Output: An SPN representing a distribution over $\mathbf{V}$
begin
  `// 1. Base Case`
  if *conditions for inducing the base models are satisfied*
   then return LEARNBASEMODEL($\mathbf{T}$,$\mathbf{V}$)
  `// 2. Decomposition Step`
  if $\mathbf{V}$ *can be partitioned into subsets* $\mathbf{V_j}$
   then return $\prod_j$ LEARNSPN($\mathbf{T}$,$\mathbf{V_j}$)
  `// 3. Splitting Step`
  Partition $\mathbf{T}$ into subsets of similar instances $\mathbf{T_i}$
  return $\sum_i \frac{|\mathbf{T_i}|}{|\mathbf{T}|}$ LEARNSPN($\mathbf{T_i}$,$\mathbf{V}$)
end

Several techniques proposed in literature for learning SPNs (and equivalently ACs) can be understood as special cases of Algorithm 1, with the difference between them being the approaches used at the three steps. Table 1 gives examples

Reference	Base Case	Decomposition	Splitting
Gens and Domingos [11]	Univariate distribution	Independence tests	Latent Variables
Gogate et al. [15]	Univariate distribution	Independence assumption	Conjunctive fixed-length features
Cutset networks (CNets)[25]	Tree Markov networks	not used	Observed variables
Ensembles of CNets[24]	Tree Markov networks	not used	Observed and Latent variables
Vergari et al. [28]	Tree Markov networks	Independence tests	Latent Variables
Rooshenas and Lowd [26]	Tractable Arithmetic Circuits	Independence tests	Latent variables

Table 1: Examples of SPN structure learning approaches in the literature that follow the prescription given in Algorithm 1. Base case is the stopping criteria for the recursive algorithm. [11, 15] stop when only one variable remains and induce a univariate distribution; [25, 28, 24] stop when the entropy of the data is small or use a Bayesian criteria, and induce an SPN corresponding to a tree Markov network at the leaves using the Chow-Liu algorithm [5] (this algorithm runs in polynomial time and yields an optimal tree Markov network according to the maximum likelihood criteria). [26] learns an SPN over observed variables in the base case using the algorithm described in [19]. In the decomposition step, [11, 26, 28] use pair-wise variable independence tests (e.g., the G-test) for inducing the product nodes; [15] uses no independence tests and instead assume that each split decomposes the variables into multiple components; while [25, 24] ignore the decomposition step inducing only sum nodes. [11, 26, 28] split only over latent variables, [15, 25] split only over observed variables or their features, while [24] split over both latent and observed variables.

of techniques from the SPN literature that are based on Algorithm 1 and briefly describes how they differ.

Although, the structure learning problem is NP-hard in SPNs having only observed variables as well as in SPNs having both observed and latent variables, the parameter (weight) learning problem is easier in the former than the latter. In particular, parameter learning can be done in closed form when the SPN has only observed variables. On the other hand, the optimization problem is non-convex in the presence of latent variables and one has to use iterative algorithms having high computational complexity such as hard and soft EM to solve the non-convex problem (cf. [23, 22, 24]). Thus, although latent variables help yield a more powerful representation, they often significantly increase the learning time.

3 CONVERTING TREE SPNs TO GRAPH SPNs

A key problem with existing methods for learning SPNs is that they induce tree models, except at the leaves. It is well known in the probabilistic inference literature [9, 7, 6] that tree SPNs can be exponentially larger than graph SPNs, which are obtained from the former by merging identical sub-SPNs (see Fig.1(a) and (b)). Thus, converting tree SPNs to graph SPNs is a good idea because they can significantly improve the time required to make predictions.

From a learning point of view, graph SPNs can potentially improve the generalization performance by addressing the following issue associated with the LEARNSPN algorithm: as the depth of the node increases,[3] the number of training examples available for learning a sub-SPN rooted at the node decreases exponentially. Merging increases the number of examples available at a node, since examples from all directed paths from the root to the node can be combined. This reduces the variance of the parameter estimates while having no effect on their bias. Since the mean-squared error of the model equals bias squared plus the variance, graph SPNs are likely to be more accurate than tree SPNs. The following proposition formalizes this intuition:

Proposition 1. Let S_1, S_2 and $S_{1,2}$ be three (sub-)SPNs having the same structure and defined over the same variables but whose parameters are estimated from training examples T_1, T_2 and $T_{1,2} = T_1 \cup T_2$ respectively. Then assuming that the datasets are generated uniformly at random from a distribution whose structure decomposes according to S_1 (and thus S_2 and $S_{1,2}$), the sample variance of $S_{1,2}$ is smaller than S_1 and S_2.

Proof. The sample variance of S_1, S_2 and $S_{1,2}$ is given by $Var(S_1)/|T_1|$, $Var(S_2)/|T_2|$ and $Var(S_{1,2})/|T_1 \cup T_2|$ respectively where $Var(S_1)$, $Var(S_2)$ and $Var(S_{1,2})$ is the (population) variance of the distributions induced by S_1, S_2 and $S_{1,2}$. Since $Var(S_1) = Var(S_2) = Var(S_{1,2})$ (our assumption), $|T_1 \cup T_2| \geq |T_1|$ and $|T_1 \cup T_2| \geq |T_2|$, the proof follows. □

3.1 OUR APPROACH

The main idea in our approach is to relax the identical sub-SPN requirement and merge *similar* sub-SPNs. We use this relaxation because the sub-SPNs are estimated from data and the likelihood that they will be identical is slim to none. In this context, we develop methods for answering the following two questions: which sub-SPNs to merge and how to merge them.

[3]The depth of a node equals the number of sum nodes from the root to the node.

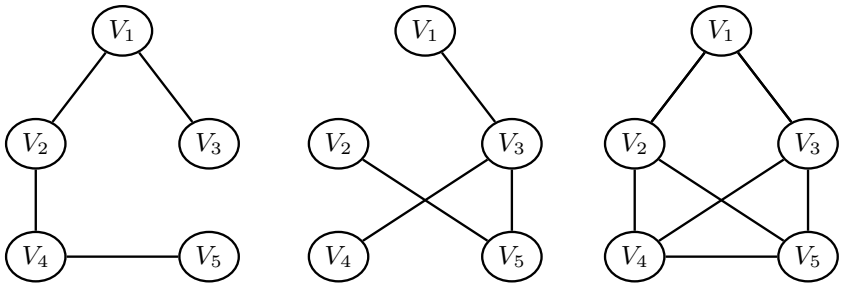


Figure 2: Figure demonstrating why distance computations are hard. (a) and (b): Two tree Markov networks over five variables $\{V_1, \ldots, V_5\}$. The treewidth of these networks is 1 and therefore the complexity of performing inference over them is $O(d^2)$ where d is the number of values in the domain of each variable. (c): Markov network obtained by taking the union of the edges of the tree Markov networks given in (a) and (b). Computing the distance (e.g., KL divergence) between the probability distributions represented by the two Markov networks in (a) and (b) is exponential in the treewidth of the Markov network given in (c). The treewidth of this network is 3 and therefore the complexity of computing the distance is $O(d^4)$, an exponential increase over $O(d^2)$.

One approach for selecting candidate sub-SPNs for merging is to compare the distance between the distributions represented by the two sub-SPNs, given that they are defined over the same variables, and check if the distance is smaller than a threshold. However, computing the distance between two sub-SPNs can be quite hard. For instance, assume that the two sub-SPNs represent Markov networks (MNs) and the junction tree or AND/OR graph search algorithm [9] is used for computing the KL divergence between the probability distributions represented by the two MNs. In this case, the time and space complexity of computation is exponential in the treewidth of the graph obtained by taking a union of the edges of the two MNs. The treewidth of this graph can be quite large (see Fig. 2 for an example). Therefore, we propose to use the following mean-field style approximation [29] of the distance between the two distributions:

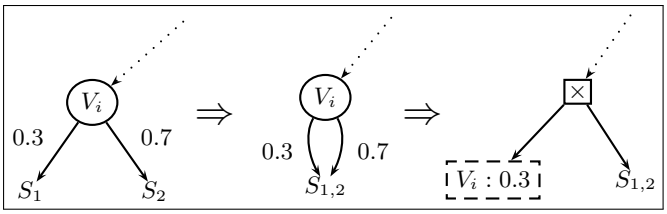
$$D(P||Q) \approx \frac{1}{|\mathbf{V}|} \sum_{V_i \in \mathbf{V}} D(P(V_i)||Q(V_i))$$

where P and Q are two distributions over $\mathbf{V}$ and D is a distance function (e.g., KL divergence, relative error, Hellinger distance, etc.). Since single-variable marginal distributions in each sub-SPN can be computed in time that is linear in the number of nodes of the sub-SPN (and in practice can be pre-computed), our proposed distance method is also linear time.

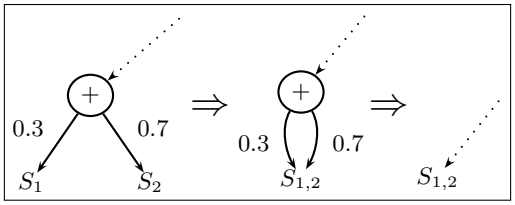
Next, we describe our greedy, bottom-up approach for merging similar sub-SPNs of a given SPN S (see Algorithm 2). The algorithm begins by initializing S' to S and repeats the following steps until convergence. For all sub-SPNs S_i of S' that are defined over exactly i variables, it partitions the sub-SPNs based on their scopes such that all sub-SPNs having the same scope are in the same cell (part) ρ_j of the partition ρ. Then, in each cell ρ_j, ensuring that S' remains a DAG, it merges all sub-SPNs such that the distance between them is bounded by ϵ, a user-defined constant that can be set using a validation set. Another option is to merge two sub-SPNs if the accuracy on the validation set improves, thereby using a greedy strategy (in our experiments, we used both strategies). Note that the for-loop of the algorithm operates in a bottom-up fashion similar to reduced-error pruning in decision trees. The loop starts at the leaves, which are sub-SPNs having just one variable in their scope ($i = 1$), and then proceeds towards the root which includes all variables in its scope ($i = |\mathbf{V}|$). The algorithm is guaranteed to converge in finite number of iterations because at each iteration, the size of the SPN can only decrease or remain the same.

Algorithm 2: MERGE(S,$\mathbf{V}$,ϵ)

Input: SPN S
Output: Merged SPN S'
begin
  $S' = S$
  repeat
    for $i = 1$ *to* $|\mathbf{V}|$ **do**
      $S_i =$ sub-SPNs in S' having exactly i variables in their scope
      $\rho =$ Partition S_i into cells having identical scopes
      for *each cell* ρ_j *of* ρ **do**
        Merge all sub-SPNs in ρ_j such that the distance between them is bounded by ϵ and S' is a DAG
      end
    end
  until *convergence*;
  return S'
end



(a) Merge child nodes of observed sum node



(b) Merge child nodes of latent sum node

Figure 3: Figure demonstrating how to simplify and thus reduce the size of the SPN after merging. As before, sum nodes are labeled either by a variable which denotes conditioning over the variable or by a $+$ sign which denotes that the sum node is latent. All left (right) arcs emanating from a sum node correspond to an assignment of 1 (0) to the labeled variable. Product nodes are labeled by $\times$. $S_{1,2}$ is an SPN obtained by merging SPNs S_1 and S_2. (a): shows how the SPN can be reduced when the two child nodes of an observed sum node are merged. The node $V_i : 0.3$ represents a univariate probability distribution over V_i with $P(V_i = 1) = 0.3$. (b): shows how the SPN can be reduced when the two child nodes of a latent sum node are merged.

3.2 PRACTICAL MERGING STRATEGIES

We complete the description of the algorithm by describing how to merge two similar sub-SPNs S_1 and S_2. A straightforward method is to merge the datasets at the two sub-SPNs and then learn a new graph sub-SPN, say $S_{1,2}$ from the new dataset. An issue with this approach is that since our basic algorithm (see Algorithm 1) learns tree SPNs, we have to call Algorithm 2 again to convert the newly created tree SPN to a graph SPN. This may yield a self-recursive algorithm with infinite loops that may not terminate. To overcome this computational difficulty, we propose to not relearn the structure, but only update the weights. In particular, we use the following approach. We consider two candidate structures for the merged sub-SPN; the first structure is identical to S_1 and the second to S_2. Then, we learn the weights of the two candidate sub-SPNs using the merged dataset and choose the one that yields the maximum improvement in accuracy (log-likelihood score) over the validation set.

There are two types of merging that require special attention. The first type is when the two sub-SPNs are children of the same sum node. In this case, if the sum node corresponds to splitting over an observed variable, we can replace the sum-node by a product node having two children as depicted in Fig. 3(a). On the other hand, if the sum node is a latent node then the sum node can be deleted without changing the underlying distribution. This is depicted in Fig. 3(b). This type of merging is useful because it substantially simplifies the model, allowing us to either prune sub-SPNs (see Fig. 3(b)) or take advantage of problem decomposition (see Fig. 3(a)). This yields better generalization and faster inference.

A second type of merging that requires special attention is when the two sub-SPNs to be merged correspond to tree Markov (or Bayesian) networks over the observed variables. In this case, unlike in the general case, we propose to learn both the structure and parameters of the merged sub-SPN (using the merged dataset). This is because both the structure and parameter learning problem in such SPNs can be solved in polynomial time using the Chow-Liu algorithm [5].

4 EXPERIMENTS

4.1 SETUP

We evaluated the impact of merging SPNs on 20 real world benchmark datasets presented in Table 3. These datasets have been used in numerous previous studies for evaluating the performance of a wide variety of tractable probabilistic graphical model learners (cf. [18, 11, 26, 25, 1, 24]). All datasets are defined over binary variables that take values from the set $\{0, 1\}$. The number of the variables in them range from 16 to 1556 and the number of training instances range from 1600 to 291326. All of our experiments were performed on a quad-core Intel i7 2.7 GHz machines with 16 GB RAM. Each algorithm was given a time bound of 48 hours, after which the algorithm was terminated.

4.2 ALGORITHMS EVALUATED

We implemented two variants of SPNs: SPNs in which sum nodes split over value assignments to a latent variable and SPNs in which sum nodes split over value assignments to a heuristically chosen observed variable. Henceforth we will call the two SPNs `L-SPNs` and `O-SPNs` respectively. We learned tree versions of both SPNs using Algorithm 1. We used tree Markov networks (MNs) as base models in both SPNs; as mentioned earlier tree MNs can be learned in polynomial time using the Chow-Liu algorithm.

To learn sum nodes in `L-SPNs`, following Gens and Domingos [11], we employed hard EM over a naive Bayes mixture model with three random restarts for 15 iterations to split the training instances into two clusters, i.e. we only considered binary splits for latent sum nodes for better reg-

Table 2: Table showing the impact of merging on the average test-set log likelihood, time complexity and prediction time of L−SPNs and O−SPNs (all values rounded to two decimal places). We use the following notation: (1) T-LL: Average test-set log likelihood for the tree SPNs; (2) G-LL: average test-set log likelihood for the graph SPN obtained from the tree SPN by merging similar sub-SPNs; (3) |T|: number of parameters in the tree SPN; (4) |G|: number of parameters in the graph SPN; (5) CR:=Compression Ratio = $\frac{|T|}{|G|}$; (6) T-Time: Tree SPN learning time in seconds and (7) G-Time: Time in seconds required by the merging algorithm (thus the total learning time for graph SPNs is T-time+G-time seconds). In each row, **bold** values indicate the best score for each of the two SPN categories: L−SPN and O−SPN.

Datasets	L-SPN							O-SPN						
	T-LL	G-LL	|T|	|G|	CR	T-time	G-time	T-LL	G-LL	|T|	|G|	CR	T-time	G-time
NLTCS	-6.03	-6.04	5498	3988	1.38	5.37	396.69	-6.04	-6.05	1406	1152	1.22	0.98	4.69
MSNBC	-6.46	-6.46	2780	2440	1.14	109.38	53.49	-6.09	**-6.08**	20032	9478	2.11	6.36	1245.62
KDD	-2.14	-2.14	11516	6670	1.73	199.13	15119.05	-2.22	**-2.19**	34328	16608	2.07	91.38	59.54
Plants	-12.80	**-12.69**	65132	47802	1.36	68.44	17775.76	-13.83	**-13.49**	86530	36960	2.34	9.56	14.12
Audio	-40.11	**-40.02**	12798	10804	1.18	68.30	1995.94	-42.06	-42.06	6142	6142	1.00	10.74	3.90
Jester	-53.12	**-52.97**	12798	10002	1.28	39.09	20.89	-55.38	**-55.36**	6142	4996	1.23	6.51	2.39
Netflix	-56.71	**-56.64**	12798	11604	1.10	62.64	2287.78	-58.64	-58.64	6142	6142	1.00	19.95	2.35
Accidents	-30.09	**-30.01**	14206	13322	1.07	58.23	2089.49	-30.83	-30.83	6846	6846	1.00	14.13	3.90
Retail	-10.88	**-10.87**	3238	2162	1.50	51.15	75.25	-11.02	**-10.95**	6302	3158	2.00	32.69	15.06
Pumsb_star	-24.17	**-24.10**	19558	17604	1.11	66.05	2314.47	-24.42	**-24.34**	20222	18338	1.10	20.6	14.93
DNA	-85.90	**-85.51**	5758	4320	1.33	8.26	11.26	-90.43	**-87.49**	11262	1430	7.88	3.76	9.48
Kosarek	-10.62	-10.62	5318	5318	1.00	219.01	200.11	-11.10	**-10.98**	11902	6712	1.77	79.55	46.66
MSWeb	-9.95	**-9.90**	32926	16484	2.00	490.12	29482.04	-10.07	**-10.06**	15086	12770	1.18	209.54	21.07
Book	-34.80	**-34.76**	15998	11998	1.33	220.56	129.98	-38.60	**-37.44**	31740	11916	2.66	387.75	10.75
EachMovie	-52.07	-52.07	15998	15998	1.00	94.92	91.31	-59.99	**-58.05**	31745	19846	1.60	176.95	6.18
WebKB	-154.86	**-153.55**	26846	20134	1.33	157.89	78.64	-172.08	**-161.17**	53438	10046	5.32	287.01	249.27
Reuters-52	-84.70	**-83.90**	56894	46232	1.23	478.65	1331.38	-90.43	**-87.49**	56638	28334	2.0	485.6	428.4
20NewsGrp.	-154.35	-154.67	58238	43684	1.33	913.81	3457.07	-163.35	**-161.46**	57982	29016	2.0	827.71	705.53
BBC	-256.05	**-253.45**	33854	21160	1.60	98.55	53.93	-272.98	**-260.59**	63242	8454	7.48	163.47	142.89
Ad	-16.77	-16.77	49790	49790	1.00	244.44	155.53	-17.37	**-15.39**	62098	31070	2.00	953.70	832.40

ularization and faster learning as in [28]. To learn sum nodes in O−SPNs, we employed two heuristics proposed in our previous work [25, 24]. The first heuristic selects an observed variable that has the highest information gain. The second heuristic selects an observed variable based on the following mutual information based criteria: given a set of variables $\mathbf{V}$ and training data T, we score each variable $V_i \in \mathbf{V}$ using $Score(V_i) = \sum_{V_j \in \mathbf{V} \setminus V_i} I_T(V_i, V_j)$ where $I_T(V_i, V_j)$ is the mutual information between V_i and V_j according to T and choose a variable having the highest score. Variables having high mutual information score are likely to yield better decompositions, which in turn will likely yield small depth SPNs having high generalization accuracy.

In both L−SPNs and O−SPNs, we learn product nodes using the technique described in Gens and Domingos [11]. We first compute the mutual information graph given data (similar to the Chow-Liu algorithm). This graph is a complete weighted graph over all variables, in which each edge is weighted by the mutual information between the two corresponding variables. Then, we prune weak edges from the graph using a threshold chosen from β: $\{0.001, 0.0015, 0.01, 0.5\}$. Finally, we find connected components of the pruned graph, and recursively learn a sub-SPN over variables and data in each connected component.

We varied the depth h of SPNs from $\{4, 5, 6, 7, 10\}$.[4] We use the following stopping criteria for learning the base model (tree Markov network): stop when the number of samples n at a node is less than 10 or the maximum depth is reached. All parameters in the model were smoothed using 1-Laplace smoothing.

For each possible configuration of h and β we learned both a tree L−SPN and a tree O−SPN. In case of O−SPNs, we also varied the heuristic to choose an observed variable. The best tree SPN in each category was chosen according to the average log-likelihood score achieved on the validation set and provided as the input to the merging algorithm (see Algorithm 2). Then, we applied practical merging and simplification strategies described in section 3.2 on the

[4]Note that the overall depth of the SPN is h plus the depth of the SPN corresponding to the tree Markov network (our base model). Thus, the overall depth can be quite large (> 30 in most cases).

merged SPN and report the test set log-likelihood score of the merged model that achieved the highest log-likelihood score on validation set.[5] We used Manhattan distance to measure the distance between two candidate sub-SPNs and chose a threshold (ϵ) from $\{0.0001, 0.001, 0.01, 0.1\}$ using the validation set. Finally, after each merge we performed the following sanity/model complexity check. If the merged sub-SPN had smaller log-likelihood than a tree Markov network on the validation set, we replaced the merged sub-SPN by the latter.

4.3 IMPACT OF MERGING ON TEST LOG-LIKELIHOOD

Table 2 shows the results of our experiments for evaluating the impact of merging on the accuracy, time complexity and prediction time of L−SPNs and O−SPNs. In terms of learning time, we see that for L−SPNs, merging requires a significant amount of time. This is to be expected because parameter learning is computationally expensive in presence of latent variables. To update the parameters of candidate sub-SPNs, we ran hard EM with the merged dataset for 20 iterations or until convergence. For some L−SPNs (e.g. Plants), merging was a factor of 200 slower than learning tree models. The reason for this anomaly is that the corresponding tree L−SPNs have large number of latent sum nodes. On the other hand, merging is significantly faster in O−SPNs than L−SPNs because the parameters are updated in closed-form, by making only one pass over the data as well as the model.

We measure the prediction time by the number of edges attached to the sum nodes (see columns |T|, |G| and CR in Table 2) since the prediction time is linearly proportional in the number of these weights. We see that in general merging yields reductions in complexity of inference by reducing the size of the network in majority of cases.

In terms of accuracy, we see from Table 2 that merging improves the test set log-likelihood score for the majority of datasets, clearly demonstrating our intuition that it will yield better generalization, primarily because it significantly reduces the variance at the cost of slightly increasing the bias.

4.4 COMPARISON WITH STATE-OF-THE-ART

Finally, we demonstrate that we can achieve state-of-the-art performance using our merging algorithm. For this, following our previous work [24, 28], we learn bagged ensemble of tree SPNs and graph SPNs. It was shown in previous studies that bagged ensembles of tree SPNs (especially latent SPNs) achieves state-of-the-art results. In our evaluation, we wanted to see whether we would be able to match or exceed these results using bagged ensemble of graph SPNs. As a strong baseline, we also compare with five other state-of-the-art tractable model learners: (1) learning sum-product networks with direct and indirect variable interactions (ID-SPN) [26], learning Markov networks using arithmetic circuits (ACMN) [20], learning mixtures of cutset networks (MCNet) [25], learning sum-product networks via SVD based algorithm (SPN-SVD) [1] and learning ensembles of cutset networks (ECNet) [24].

In our experiments, we fixed the number of bags to 40 following [24]. Instead of performing a grid search, we performed random search [3] to create a configuration for the models in the ensemble. Each component model was then weighted according to its likelihood on the training set. To get better accuracy, we treated the bagged ensemble of L−SPNs and O−SPNs as an SPN having one latent sum node as the root and each independent component (bag) as its child sub-SPN. The benefit of this approach is that instead of optimizing the local log-likelihood scores of individual SPNs, while merging, we can directly optimize the global log-likelihood.

Table 3 shows the bagged ensemble scores of L−SPNs and O−SPNs before and after merging as well as the best log-likelihood score obtained to date using the competing approaches mentioned above. Bagged graph SPNs, especially L−SPNs, performed significantly better than the state-of-the-art on all of the high dimensional datasets with very competitive scores on the others. This suggests that merging is especially useful for accurately modeling relationships in high-dimensional data (see also Table 2).

5 CONCLUSION AND FUTURE WORK

In this paper, we presented a novel algorithm for learning graph SPNs from tree SPNs by merging similar sub-SPNs in the tree SPN. Our proposed algorithm for finding and merging similar sub-SPNs is general enough to serve as a template for incorporating suitable functions that measure similarity between sub-SPNs as well as for performing arbitrary mergings. Our experimental evaluation clearly shows that graph SPNs can significantly boost the accuracy and prediction time of tree SPNs by substantially reducing the number of parameters that the learning algorithm needs to induce from data. We also investigated the merit of learning ensembles of graph SPNs, building on our previous work on learning ensembles of tree SPNs, for a variety of high dimensional real world datasets, and comparing them to other state-of-the-art tractable model learners. Our experimental results showed that ensembles of graph SPNs significantly outperformed the state-of-the-art learners, clearly demonstrating the efficacy of our proposed approach.

Future work includes: developing relational merging ap-

[5] Our experiments showed that merging sub-SPNs that are rooted at child nodes of the same sum node (the cases given in Fig. 3(a) and (b)) was often more beneficial as compared to merging sub-SPNs that are child nodes of two different sum nodes.

Table 3: Average test set log-likelihood comparison with state-of-the-art tractable model learners. **Bold** values indicate the winning score for the corresponding dataset. T-LL: Bagged LL of tree SPNs and G-LL: Bagged LL of graph SPNs. Column "Best-LL to date" gives the best log-likelihood score to date for each dataset obtained using the following competing approaches: ID-SPN [26], ACMN [20], MCNet [25], SPN-SVD [1], and ECNet [24].

Datasets	|Var|	|Train|	|Valid|	|Test|	L-SPN		O-SPN		Best-LL to date
					T-LL	G-LL	T-LL	G-LL	
NLTCS	16	16181	2157	3236	-6.01	-6.00	-6.01	-6.00	-6.00
MSNBC	17	291326	38843	58265	-6.45	-6.39	-6.10	-6.10	**-6.04**
KDD	64	180092	19907	34955	-2.13	-2.12	-2.14	-2.13	-2.12
Plants	69	17412	2321	3482	-12.31	-12.03	-12.25	-12.21	**-11.99**
Audio	100	15000	2000	3000	-39.57	**-39.49**	-40.35	-40.31	-39.67
Jester	100	9000	1000	4116	-52.65	-52.47	-53.56	-53.13	**-41.11**
Netflix	100	15000	2000	3000	-55.92	**-55.84**	-56.69	-56.65	-56.13
Accidents	111	12758	1700	2551	-29.41	-29.32	-29.81	-29.82	**-24.87**
Retail	135	22041	2938	4408	-10.85	-10.82	-10.87	-10.85	**-10.60**
Pumsb_star	163	12262	1635	2452	-23.82	-23.67	-23.85	-23.81	**-22.40**
DNA	180	1600	400	1186	-86.63	-80.89	-85.97	-84.79	**-80.03**
Kosarek	190	33375	4450	6675	-10.71	-10.55	-10.85	-10.74	**-10.54**
MSWeb	294	29441	32750	5000	-9.84	-9.78	-9.77	-9.76	**-9.22**
Book	500	8700	1159	1739	-36.49	-34.25	-36.35	-35.89	**-30.18**
EachMovie	500	4524	1002	591	-54.70	**-50.72**	-55.82	-53.07	-51.14
WebKB	839	2803	558	838	-170.27	**-150.04**	-166.65	-152.82	-150.10
Reuters-52	889	6532	1028	1540	-84.32	**-80.66**	-86.00	-82.66	-82.10
20NewsGrp.	910	11293	3764	3764	-151.48	**-150.80**	-158.40	-154.28	-151.47
BBC	1058	1670	225	330	-265.89	**-233.26**	-244.12	-238.61	-236.82
Ad	1556	2461	327	491	-16.33	-14.58	-15.69	**-14.34**	-14.36

proaches that search for similarities in sub-SPNs having different (even disjoint) scopes (cf. [12, 14]); analyzing contexts – assignment to variables on the path from the root – of merged sub-SPNs for finding symmetric contexts; directly inducing graph SPNs from data rather than using post-processing schemes; and extending the approach presented in the paper to hybrid domains having both discrete and continuous variables.

Acknowledgements

This research was funded in part by the DARPA Probabilistic Programming for Advanced Machine Learning Program under AFRL prime contract number FA8750-14-C-0005 and by the NSF award 1528037. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, AFRL, NSF or the US government.

References

[1] T. Adel, D. Balduzzi, and A. Ghodsi. Learning the structure of sum-product networks via an svd-based algorithm. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 32–41, 2015.

[2] F. R. Bach and M. I. Jordan. Thin junction trees. *Advances in Neural Information Processing Systems*, 14:569–576, 2001.

[3] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[4] M. Chavira and A. Darwiche. Compiling Bayesian networks using variable elimination. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2443–2449, 2007.

[5] C. K. Chow and C. N Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.

[6] A. Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.

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

[8] A. Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

[9] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171:73–106, 2007.

[10] A. Dennis and D. Ventura. Greedy structure search for sum-product networks. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 932–938. AAAI Press, 2015.

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

[12] V. Gogate and P. Domingos. Exploiting Logical Structure in Lifted Probabilistic Inference. In *AAAI 2010 Workshop on Statistical Relational Learning*, 2010.

[13] V. Gogate and P. Domingos. Formula-Based Probabilistic Inference. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 210–219, 2010.

[14] V. Gogate and P. Domingos. Probabilistic Theorem Proving. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 256–265. AUAI Press, 2011.

[15] V. Gogate, W. Webb, and P. Domingos. Learning efficient Markov networks. In *Proceedings of the 24th conference on Neural Information Processing Systems*, pages 748–756, 2010.

[16] D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning*, 2014.

[17] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, MA, 2009.

[18] D. Lowd and J. Davis. Learning Markov network structure with decision trees. In *Proceedings of the 10th International Conference on Data Mining*, pages 334–343, 2010.

[19] D. Lowd and P. Domingos. Learning arithmetic circuits. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, Helsinki, Finland, 2008. AUAI Press.

[20] D. Lowd and A. Rooshenas. Learning Markov networks with arithmetic circuits. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2013)*, Scottsdale, AZ, 2013.

[21] R. Mateescu, R. Dechter, and R. Marinescu. AND/OR multi-valued decision diagrams (AOMDDs) for graphical models. *Journal of Artificial Intelligence Research*, pages 465–519, 2008.

[22] M. Meila and M. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.

[23] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 337–346, Barcelona, Spain, 2011. AUAI Press.

[24] T. Rahman and V. Gogate. Learning ensembles of cutset networks. In *AAAI conference on Artificial Intelligence*, pages 3301–3307, 2016.

[25] T. Rahman, P. Kothalkar, and V. Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Proceesings of ECML and PKDD*, pages 630–645, 2014.

[26] A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect interactions. In *Proceedings of the Thirty-First International Conference on Machine Learning*, Beijing, China, 2014. JMLR: W&CP 32.

[27] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82:273–302, 1996.

[28] A. Vergari, N. Di Mauro, and F. Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 343–358. Springer, 2015.

[29] W. Wiegerinck. Variational approximations between mean field theory and the junction tree algorithm. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 626–633, 2000.