Anytime Learning of Sum-Product and Sum-Product-Max Networks

Swaraj Pawar SWARAJ.PAWAR@UGA.EDU

Prashant Doshi PDOSHI@UGA.EDU

THINC Lab, School of Computing, University of Georgia, Athens, GA 30602, USA.

Abstract

Prominent algorithms for learning sum-product networks (SPN) and sum-product-max networks (SPMN) focus on learning models from data that deliver good modeling performance without regard to the size of the learned network. Consequently, the learned networks can get very large, which negatively impacts inference time. In this paper, we introduce anytime algorithms for learning SPNs and SPMNs. These algorithms generate intermediate but provably valid models whose performance progressively improves as more time and computational resources are allocated to the learning. They flexibly trade off good model performance with reduced learning time, offering the benefit that SPNs and SPMNs of small sizes (but with reduced likelihoods) can be learned quickly. We comprehensively evaluate the anytime algorithms on two testbeds and demonstrate that the network performance improves with time and reflects the expected performance profile of an anytime algorithm. We expect these anytime algorithms to become the default learning techniques for SPNs and SPMNs given their clear benefit over classical batch learning techniques.

Keywords: Anytime algorithms; decision making; flexible; probabilistic graphical models.

1. Introduction

Sum-product networks (SPN) (Poon and Domingos, 2011) are probabilistic graphical models with tractable inference that is linear in the size of the network (number of nodes) for most types of inferences. Multiple algorithms have been presented in the past for learning the structure of SPNs, which target improvement in the likelihood of the learned networks. Another model, sum-product-max networks (SPMN) (Melibari et al., 2016) was introduced for tractable decision making by adding decision and utility nodes to generalize the SPNs. However, for complex or sequential domains the structure learning algorithm for SPMNs could fail to learn the networks in a reasonable amount of time (Tatavarti et al., 2021).

Applications that require the generative modeling of data with many variables may settle for learned SPNs of smaller sizes that offer faster inference time. This motivates a need to design algorithms that can learn smaller-sized and possibly less expressive networks in less time, if needed. But, the prominent structure learning algorithms for SPNs and SPMNs (Gens and Domingos, 2013; Melibari et al., 2016; Kalra et al., 2018) are unable to control the size of the learned networks. While some work exists that seeks to reduce network complexity or the learning time (Vergari et al., 2015; Di Mauro et al., 2017), algorithms that flexibly trade off model performance for reduced learning time by learning networks of smaller sizes are preferred.

Anytime or flexible algorithms showcase a progressive increase in the quality of the models as the computation time increases (Zilberstein and Russell, 1995; Zilberstein, 1996). These algorithms return a series of models of increasing accuracy and improve the models as the algorithm progresses. In this paper, we present the first anytime algorithms for learning SPNs and SPMNs by progressively relaxing the crucial operations of clustering and independence testing. These al-

gorithms learn the networks from data and return intermediate valid models of improving quality and increasing complexity until convergence. The model quality is measured using log-likelihood for the SPNs and additionally using average rewards for the SPMNs. These measures are used to generate the performance profiles for the algorithms over known testbeds. We observe an increase in the network size and the computation time as the algorithm progresses towards convergence. We also show that the performance of the models returned by the anytime algorithms moves toward optimality, and eventually often improves on the log-likelihoods returned by the previous structure learning techniques (Gens and Domingos, 2013).

2. Background

We briefly review SPNs and their generalization to decision-making SPMNs in this section.

2.1 Sum-Product Networks

An SPN S (Poon and Domingos, 2011) is a rooted directed acyclic graph that represents a probability distribution over random variables $X_1, \ldots, X_n$. These models consist of sum and product nodes as their internal nodes and univariate distributions over variables as their leaf nodes. Recent enhancements allow for multivariate leaf distributions as well (Rooshenas and Lowd, 2014). The outgoing edges of the sum nodes have non-negative edge weights. Thus, the value of a sum node is the weighted sum of its children's values, while the value of a product node is the product of its children. The value at the root of S is the value of the network polynomial that it represents. An SPN is *valid* iff the normalized network polynomial represents the joint distribution over the variables and it gives correct marginals. Validity thus constrains the structure of the SPN as follows:

Theorem 1 (SPN validity (Poon and Domingos, 2011)) *An SPN is valid if it is sum-complete and decomposable.*

Both completeness and decomposability impose conditions on the *scope* of a node, where the scope is recursively defined as:

Definition 1 (Scope) *The scope of a node is a union of the scopes of its children, where the scope of a leaf node is the set of random variables whose distribution it holds.*

Subsequently, completeness and decomposability are defined as:

Definition 2 (Sum-complete) *An SPN is sum-complete iff all children of a sum node have the same scope.*

Definition 3 (Decomposable) *An SPN is decomposable iff no variable appears in the scope of more than one child of a product node.*

2.2 Sum-Product-Max Networks

SPMNs (Melibari et al., 2016) generalize SPNs to have decision-making capabilities. An SPMN S^+ over decision variables $D_1, ..., D_m$, random variables $X_1, ..., X_n$ and utility functions $U_1, ..., U_k$ has max and utility nodes for the decision variables and utility functions respectively, along with the sum and product nodes. The leaf nodes are either univariate distributions over variables or utility

nodes. The internal nodes consist of sum, product and max nodes. The outgoing edges of a max node are labeled with the decision choices of the corresponding decision variable. The value of the max node is the maximum expected value among its children and the choice on the edge that yields the maximum value is the optimal decision to be taken for that variable. Let σ denote an assignment of a decision choice to each max node in S^+, which we refer to as a policy, and $S^+(\mathbf{x}, \sigma)$ denote the expected utility of a policy σ for some state $\mathbf{x}$ obtained as, $S^+(\mathbf{x}, \sigma) = \mathbf{\Phi}(\mathbf{x}) \cdot U(\mathbf{x}, \sigma)$. Here, $\mathbf{\Phi}(\mathbf{x})$ is the probability distribution over the state $\mathbf{x}$ and $U(\mathbf{x}, \sigma)$ is the sum of all the utility functions. Furthermore, $S^+(\mathbf{x}) = \max_{\sigma} S^+(\mathbf{x}, \sigma)$. Then, an SPMN S^+ is valid iff $S^+(e) = \sum_{\mathbf{x} \sim e} S^+(\mathbf{x})$ for evidence e and $\mathbf{x} \sim e$ denotes states consistent with evidence e. Proposition 1 then follows from this definition of validity.

Proposition 1 *The value of a valid SPMN for evidence e is identical to the maximum expected utility (EU) of that evidence, $S^+(e) = \max_{\sigma} EU(e, \sigma)$, where σ is the policy.*

Proof of the proposition is given in the Appendix included in the supplementary material at http://thinc.cs.uga.edu/files/pdPGM22-appendix.pdf. Melibari et al. (2016) notes that an SPMN is valid if it satisfies Defs. 2, 3, and two new additional properties:

Definition 4 (Max-complete) *An SPMN is max-complete iff each child of a max node has the same scope.*

Definition 5 (Max-uniqueness) *An SPMN is max-unique iff each max node corresponding to a decision variable appears at most once in every path from the root to the leaves.*

A partial order denoted by $P^{\prec}$ gives the order between information sets and the decision variables. More specifically, it is given as $I_0 \prec D_1 \prec I_1 \prec ... \prec D_m \prec I_m$, where the random variables within the information set I_{i-1} are observed prior to the decision associated with variable D_i, $1 \leq i \leq m$, is taken. This is a partial order because variables within each information set may be observed in any order. Learned SPMNs must respect the partial order such that no variables from the information set I_{i-1} should be within the scope of the decision variable D_i, but it should contain the variables from the information sets $I_i, I_{i+1}...$ within its scope.

3. Anytime Learning of SPNs and SPMNs

The popular LEARNSPN and LEARNSPMN algorithms do not control the size of the learned networks. For a given data set, both methods utilize the default of two clusters resulting in two children for a sum node and all variables within the scope are used for splitting for the product nodes. As such, these methods do not regulate their learning times and often learn complex networks containing hundreds of nodes even for simple domains (Vergari et al., 2015).

An *anytime* approach (Zilberstein and Russell, 1995; Zilberstein, 1996) for learning the networks allows flexible control over the size and performance of the learned networks. This approach initially generates smaller and approximate networks, though still valid, in less run time. As more resources are allocated, the size of the networks increases until convergence, which typically correlates with their modeling performance.

3.1 Anytime Structure Learning for SPNs

A new learning method ANYTIMESPN is presented in Algorithm 1, which yields valid SPN structures at each iteration, and these can be flexibly improved with more iterations. It takes as input the dataset D and a list V of the random variables in the domain. At any time, the algorithm may be halted and it outputs a valid SPN; the algorithm yields SPNs with improved likelihoods when allowed more iterations.

Algorithm 1: ANYTIMESPN

Input: D: Dataset, V: Variables
Output: Series of learned valid SPNs
Set $\kappa \leftarrow 2$ ▷ limit on number of clusters
Set $\eta \leftarrow \text{ceil}(\sqrt{|V|})$ ▷ number of variables for independence testing
do
  $S \leftarrow$ **LearnSPN*(**D, V, sum**)**
  Set $\kappa \leftarrow \kappa + 1$ ▷ Increment parameters
  Set $\eta \leftarrow \eta + 1$ until η reaches $|V|$
until log likelihood converges

Algorithm 2: ANYTIMESPMN

Input: D: Dataset, V: Variables, $P^{\prec}$: Partial order
Output: Series of learned valid SPMNs
$\kappa \leftarrow 2$, $\eta \leftarrow \text{ceil}(\sqrt{|V|})$, $d \leftarrow 2$, $d_{max} \leftarrow 1$
do
  $S^{+} \leftarrow$ **LearnSPMN*(**D, V**, 0, null)**
  Set $\kappa \leftarrow k + 1$
  Set $\eta \leftarrow n + 1$ until n reaches $|V|$
  Set $d \leftarrow d + 1$ until d reach $|D|$
  Set $d_{max} \leftarrow d_{max} + 1$
until log likelihood converges

ANYTIMESPN has two parameters: the upper limit κ on the number of clusters to be formed and the number of variables η used for independence testing. [1] Parameter κ controls the branching and the complexity at the sum node, while the parameter η reduces the run time and generates approximate networks in the initial iterations. Parameter κ is initialized to yield at most two clusters while η is set to $\sqrt{|V|}$ initially. At each iteration, a valid usable SPN is generated after which the parameters κ and η are incremented. This continues until the value of η increments to $|V|$ and the log-likelihood of the learned networks converges. The SPNs (in each iteration of ANYTIMESPN) are generated by LEARNSPN* outlined in Algorithm 3. Along with the parameters κ and η, the algorithm takes as input the dataset D, the scope variables V and a current operation indicator $curOp$, which is *sum* in Algorithm 1. The algorithm returns a learned SPN given the parameters.

Whereas similar to LEARNSPN, LEARNSPN* uses RDC independence testing (Molina et al., 2018) to generate the product nodes and its children, this operation differs from LEARNSPN in that only the first η variables in the scope are used for splitting into the independent subsets V_j. We avoid randomly selecting these η variables as it might lead to a sudden drop in the likelihood. *Instead, we use the first η variables using the ordering presented in the data set.* If all η variables are grouped into one subset, another subset is created with an equal number of variables picked from the remaining $|V| - \eta$ variables. The variables not yet assigned are distributed evenly among the subsets. The sub-SPN learned from each subset is assigned as the child of a product node.

Next, we use the X-MEANS clustering algorithm (Pelleg and Moore, 2000) to form the clusters pertaining to the sum nodes and its children. This algorithm differs from the k-means used in LEARNSPN by allowing a variable number of clusters up to the limit of κ clusters. The algorithm aims to initially partition the data set into two clusters. If the clusters are found, in its next iteration it aims to divide each of those clusters again into two. If the new SPN model improves on the previous one using a metric such as the well-known *Bayesian information criterion* (Schwarz, 1978), then the clustering is continued. Otherwise, the algorithm is terminated and the clusters from the previous iteration are returned. This iterative expansion process continues while the number of clusters found

1. These degrees of freedom are not available in LEARNSPN.

Algorithm 3: LEARNSPN*

Input: D: Dataset, V: Variables, $curOp$: current operation
Parameters: κ: maximum cluster limit, η: number of variables for independence testing, ι: minimum number of instances to allow a variable split
Output: learned SPN structure

if $|V| = 1$ **then**
  return univariate distribution over V with Laplacian smoothing
if $|D| < \iota$ **then**
  return a product node with univariate distribution of each variable in V as its child
if $curOp = prod$ **then**
  Partition variables $V[:n]$ into independent subsets V_j
  Distribute the remaining variables $V[n:]$ evenly among the subsets V_j
  return Π_j **LearnSPN*(**D_j, V_j, sum**)**
else
  partition D into max. number of clusters D_j of similar instances such that $j \leq k$
  return $\Sigma_j \frac{|D_j|}{|D|} \times$ **LearnSPN*(**$D_j, V, prod$**)**

is less than κ. The sub-SPN structures learned from each of the clusters form the children of the sum node. If only one variable remains in the scope, then LEARNSPN* returns a (Laplacian) smoothed univariate distribution over that variable to form a leaf node.

Theorem 2 *The SPNs learned at each iteration of the* ANYTIMESPN *algorithm are valid where validity is as defined in Theorem. 1.*

Proofs of all theorems are given in the Appendix included in the supplementary material. A valid SPN can be evaluated using the standard bottom-up traversal to obtain $S(e)$.

3.2 Anytime Structure Learning for SPMNs

Algorithm 2 ANYTIMESPMN gives the procedure for learning SPMNs using the anytime technique. A data set D, the list of variables V (comprised of random, decision, and utility variables) present in the scope of the domain, along with the partial ordering of the random and decision variables $P^{\prec}$ is given as input to the algorithm. It learns a series of SPMNs such that there is improvement in the performance after each iteration.

The algorithm utilizes parameters d and d_{max} in addition to κ and η, which are similar to those utilized by ANYTIMESPN. The parameter d represents the limit on the branching of the decision node. *Each branch of the decision node may correspond to multiple decision choices.* During evaluation, the actual decision for a decision variable is selected randomly from the set of choices assigned to the edge that gives the maximum expected utility. The value of d_{max} gives the maximum possible depth allowed from the point where no decision variables appear in the scope of the branch.

The algorithm first initializes these parameters to their lowest feasible values: at most clusters for the sum nodes, $\eta = \sqrt{|V|}$, two branches for every max node, and $d_{max} = 1$. On learning an SPMN, κ, d, η and d_{max} are incremented. In particular, η is gradually increased until it equals to $|V|$, while d is incremented until it reaches the number of decision choices possible for the decision variables. Parameters κ and d_{max} are increased unbounded. This process stops when the log-likelihoods of the generated networks converge after the values of η and d reach their maximums.

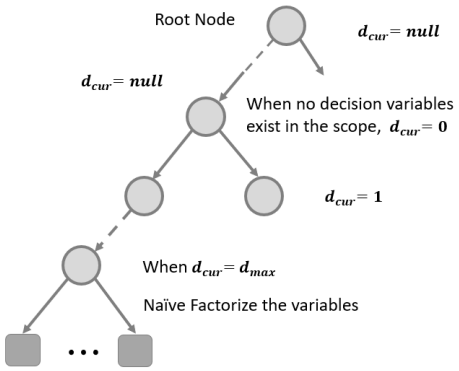

Figure 1: An illustration of depth control using the parameters d_{max} and d_{cur}.

Algorithm 4 LEARNSPMN* is used for flexibly learning the SPMN structures at each iteration. Along with the data set D, scope variables V and the partial order $P^{\prec}$, the method requires the current information set i in $P^{\prec}$ as the input. Another input parameter d_{cur} gives the depth at which the algorithm currently is after the point where no decision variables are found in the scope. Initially i is set to 0 and d_{cur} is $null$.

If the scope V contains one variable only, then Algorithm 4 returns a utility node if the variable in V is a utility, otherwise it returns a smoothed univariate distribution over the single random variable. The information set index i is incremented if there are no variables from $P^{\prec}[i]$ in V. If there are no decision variables in the scope V and parameter d_{cur} is still $null$, the current branch can be truncated from this point onward. At this point the value of d_{cur} is set to 1. Otherwise, d_{cur} is incremented by 1 to indicate increase in the level of depth of the branch. If d_{cur} exceeds the limit given by d_{max}, then the algorithm returns a product node having $|V|$ children created by naïve factorizing the scope variables V. This gives an approximate distribution over the variables once the allowed depth is reached. This depth control procedure is illustrated in Figure 1. After this, all remaining variables from the next information set in $P^{\prec}$ and those that follow are stored in V_R.

If the current information set contains a decision variable, then the decision choices for that variable present in D are uniformly distributed among d groups. For each group v_g, the subset D_{v_g} of D where the decision variable has a value present in the group is formed. A max node having d children is returned, where each child is a sub-SPMN learned from the data subset having scope V_R. On the other hand, if $P^{\prec}[i]$ does not have a decision variable, then variable splitting is performed using the first η variables from V as explained for SPNs. The subsets having variables also present in V_R are merged together. Now, if there is more than one subset, a $product$ node is created having SPMN structures that are learned from the subsets as its children. Otherwise, the data set D is partitioned into a maximum of κ clusters of similar instances using X-MEANS by only considering the variables in the current information set, to form a sum node and its children. SPMNs generated by ANYTIMESPMN exhibit an important property:

Theorem 3 *The SPMNs returned at each iteration of the* ANYTIMESPMN *algorithm are valid where the validity is as defined in Section 2.2.*

As an example, consider the *Export Textiles* domain (Er and Lezki, 2012) having one random variable Economical State (ES), a decision variable Export Decision (ED) and a utility value Profit (Pr). The task is to select a decision for ED that maximizes the utility Pr. The decision

Algorithm 4: LEARNSPMN*

Input: D: Dataset, V: Variables, i: Information set index, d_{cur}: Current depth after decision nodes

Parameters: $P^{\prec}$: Partial order, κ: Maximum cluster limit, d: Maximum decision node branches, η: # variables for independence testing, d_{max}: Maximum depth after decision nodes

Output: learned SPMN

if $|V| = 1$ **then**
  if variable $v \in V$ is utility **then**
    $u \leftarrow$ estimate $Pr(V = True)$ from D
    return utility node with value u
  else
    return univariate distribution over V with Laplacian smoothing
Update $i \leftarrow i + 1$ and $d_{cur} \leftarrow d_{cur} + 1$
if $d_{cur} \geq d_{max}$ **then**
  return a product node with univariate distribution of each variable in V as its child
$V_R \leftarrow P^{\prec}[i+1] \cup P^{\prec}[i+2] \cup P^{\prec}[i+3]$...
if $P^{\prec}[i]$ contains a single decision variable **then**
  $v_{groups} \leftarrow$ distribute decision values in d groups
  for $v_g \in v_{groups}$ **do**
    $D_{v_g} \leftarrow$ subset of D where $P^{\prec}[i] \in v_g$
  return MAX_{v_g} **LearnSPMN*(**$D_{v_g}, V_R, i+1, d_{cur}$**)**
else
  $Z \leftarrow$ Partition variables $V[1:\eta]$ into subsets V_j that are statistically independent
  $D_j \leftarrow$ subset of data having values of variables in V_j
  Distribute the remaining variables $V[\eta:n]$ among the subsets $V_j \in Z$
  Merge together subsets having variables $\in V_R$
  if $|Z| > 1$ **then**
    return Π_j **LearnSPMN*(**$D_j, V_j, i, dcur$**)**
  else
    partition D into clusters D_j of similar instances based on the values in $P[i]$ such that $j \leq \kappa$
    return $\Sigma_j \frac{|D_j|}{|D|} \times$ **LearnSPMN*(**D_j, V, i, d_{cur}**)**

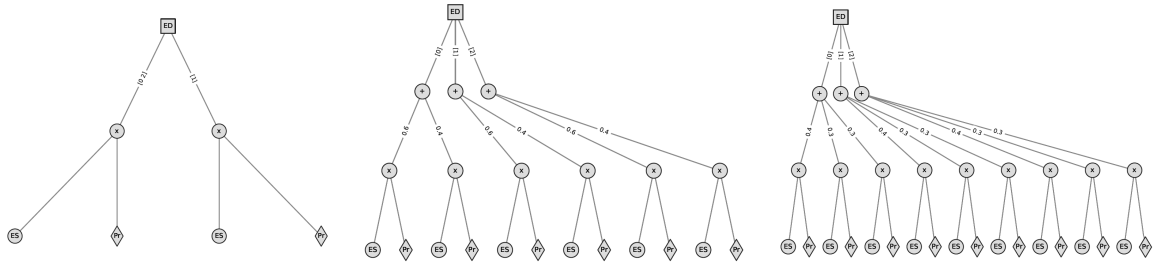

Figure 2: SPMNs from ANYTIMESPMN for *Export Textiles*. (*left*) Initial, (*middle*) an intermediate, (*right*) the final learned network. All SPMNs are valid and present increasingly improved modeling.

variable has three choices and the variable ES is discrete having three values. Some of the SPMNs generated by ANYTIMESPMN for the domain are shown in Fig. 2. Notice that the max node has two branches in Fig. 2 (left) and the depth of the subtrees following the decision node is limited to one. The final network exhibits the optimal MEU. All the SPMNs are valid and the number of nodes and the accuracy of the models increases as expected over the iterations of the algorithm.

Dataset	$\|V\|$	**# Instances**
NLTCS	16	19417
MSNBC	17	349591
KDDCup 2K	65	215047
Jester	100	13116
Audio	100	18000
Netflix	100	18000

(a)

Dataset	$\|X\|$	$\|D\|$	$\|d\|$	Optimal MEU
Elevator	35	6	4	0.5
Navigation	36	5	4	-4.047
Game of Life	36	3	9	10.808
Skill Teaching	72	5	4	-7.181
Crossing Traffic	72	5	4	-4.0

(b)

Table 1: (a) Data sets for evaluating ANYTIMESPN. $|V|$ indicates the number of random variables in the data set, each is binary. (b) Data sets for evaluating ANYTIMESPMN. Here, $|X|$ is the number of random variables, $|D|$ is the number of decision variables, and $|d|$ is the number of decision choices per decision variable. Each data set has 500K instances.

4. Experiments

Both Algorithms 1 and 2 were implemented in the open-source SPFlow library (Molina et al., 2019) and are available on GitHub. [2] We evaluate ANYTIMESPN on a testbed of 6 data sets available as part of the SPFlow library. The number of domain variables and the total number of instances are listed in Table 1 (a). ANYTIMESPMN is tested on data sets generated from the RDDLSim domains (Sanner, 2011) whose specifications are shown in Table 1 (b). The sequential RDDLSim domains were modeled as decision-making domains for a finite number of steps. Each decision-making data instance is generated by simulating a random agent in the environment for the given steps $|D|$. The instances respect the partial order, since I_i is the state observed after an action d_i at the time step i. We record the total reward gained as the utility U.

4.1 Performance Profiles of ANYTIMESPN

We measure the log-likelihoods of the learned SPNs from the ANYTIMESPN algorithm to build the anytime technique's performance profiles. These profiles show the change in the measure as iterations progress (and therefore there is more resource allocation) for an anytime technique. We use 3-fold cross validation for evaluating each of the learned SPNs. We compare the SPNs with that learned by the batch LEARNSPN algorithm. Additionally, we compare with the model learned without any constraints on the parameters κ and η. Consequently, this model is expected to show case the best representation and serves as the upper bound for our results; we denote it as *Upper Limit*. ANYTIMESPN is considered to reach convergence when the standard deviation in the log-likelihood of the past three models is less than 10^{-3}.

The results in Figure 3 for three of the six domains show that the log-likelihoods follow the expected performance profiles of an anytime technique. The models improve at a higher rate in the ini-

2. The implementation of ANYTIMESPN is available at https://github.com/SwarajPawar/SPFlow/tree/anytime_spn and that for ANYTIMESPMN is available at https://github.com/SwarajPawar/SPFlow/tree/anytime_spmn.

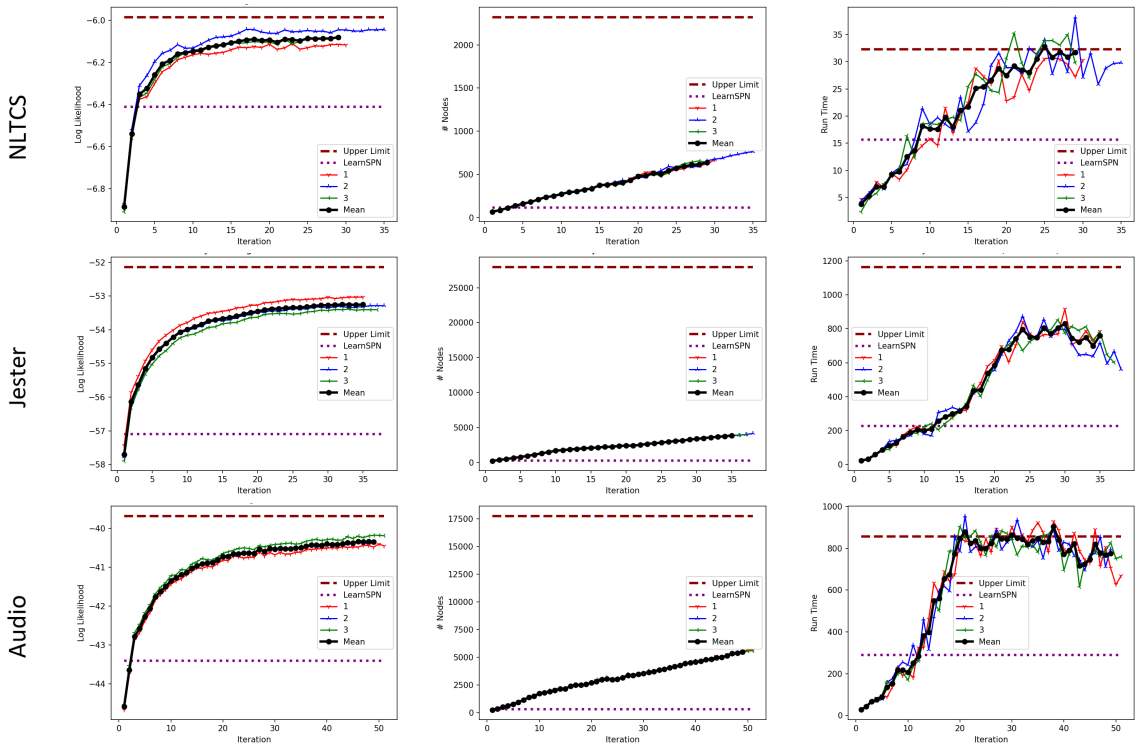

Figure 3: Performance profiles of ANYTIMESPN on 3 of the 6 data sets. Left column shows the log likelihoods, middle column the number of nodes in the SPNs, and the right column presents the run time of each iteration of the algorithm in seconds. Results for the remaining data sets are given in the supplementary file at http://thinc.cs.uga.edu/files/pdPGM22-appendix.pdf. All experiments were run on a Red Hat Linux 8 system with Intel Xeon 12 cores 1.63 GHz each and 16 GB of RAM.

tial iterations, which then starts to flatten out toward convergence yielding a nearly non-decreasing log-likelihood, while the network size increases linearly. *Notice that the anytime algorithm for SPNs is able to learn models whose performance approaches that of the* Upper Limit *model while using significantly less nodes.* Such performance is achieved with smaller network sizes in less or equivalent run times as compared to the *Upper Limit*. Furthermore, the anytime approach learns models that are much better fits as compared to the previous LEARNSPN models in less run times.

4.2 Performance Profile of ANYTIMESPMN

For SPMNs, we measure both the log likelihoods and the MEUs. While the former was evaluated using 3-fold cross-validation, the latter was measured as the total reward from simulating the SPMN decisions in the domains averaged over 25 batches of 20,000 runs each. We compare these results with the previous batch algorithm LEARNSPMN. Additionally, we compare the average rewards obtained with the optimal MEU and the average reward given by a random policy. The optimal MEU is obtained from the policy given by the value iteration solver in RDDLSim.

Figure 4 shows the average reward and log likelihood performance profiles of the ANYTIME-SPMN algorithm, as well as the number of nodes in the learned SPMNs and the learning run times. Observe that both the average rewards and the log likelihoods of the models generally increases given more iterations and is consistent with the increase in the network sizes. A slight exception is

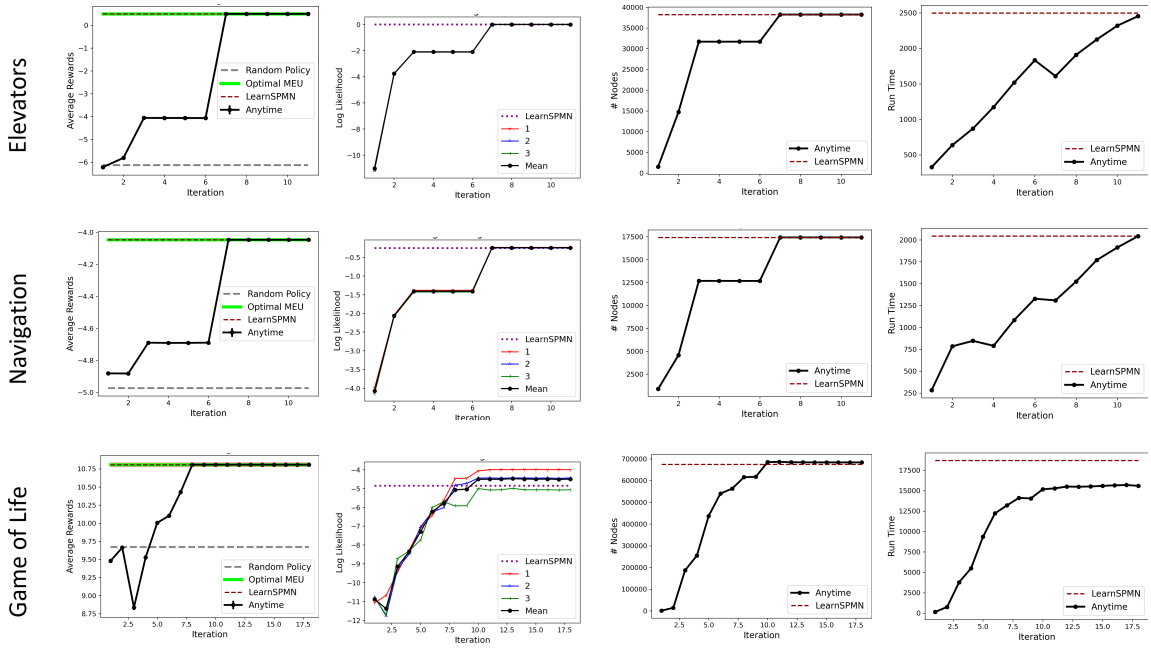

Figure 4: Performance profiles of the ANYTIMESPMN algorithm on three of the five decision-making data sets. Results for additional data sets are shown in the Appendix in the supplementary material. We show, in order from left to right columns, the average rewards (with standard deviations), log likelihoods, number of nodes in the learned SPMNs, and the learning run time in seconds.

the Game of Life domain where the SPMN on the third iteration shows a drop in reward and log likelihood. However, the average rewards reach the optimal values by about the tenth iteration for the domains that are shown indicating that the later SPMNs are yielding the optimal decisions. The learning time for the models (shown in seconds) also generally increases over the iterations (though we note some localized exceptions). Most of the models yielded by the algorithm perform better than a random policy. As such, ANYTIMESPMN produces optimal models in lesser learning times than the batch LEARNSPMN algorithm.

5. Related Work

Multiple methods exist for learning the structure of SPNs such as the cluster architecture algorithm (Dennis and Ventura, 2012), the popular LEARNSPN (Gens and Domingos, 2013) for tree SPNs with leaves as univariate distributions, ID-SPNs (Rooshenas and Lowd, 2014) having leaves as multivariate distributions, a bottom-up greedy approach (Peharz et al., 2013), and SPN-SVD (Adel et al., 2015) based on rank-one submatrices. These algorithms aim to learn SPNs that improve the log-likelihood of the data with no way of regulating the learned network size.

Vergari et al. (2015) modifies LEARNSPN by suggesting binary splits for the sum nodes to allow deeper SPNs. To yield simpler SPNs with fewer edges, it suggests Chow-Liu Trees (Chow and Liu, 1968) instead of naïve factorization. However, reductions in the number of edges were found only for few cases, while additional computation time is required for such trees. The RGVS approach (Di Mauro et al., 2017) randomly picks a subset of variables from the scope for variable splitting.

Unlike our variable splitting, the number of independent subsets is limited to two groups only which may not be practical. It shows reduction in the learning time and in the performance of the model. But, the likelihood may not progressively improve when the size of the subset used for splitting is increased because the remaining variables are assigned randomly to one of the independent subsets.

Melibari et al. (2016) introduces LEARNSPMN for learning the structure of SPMNs from decision-making data with a given partial order. Extensions of SPMNs for sequential domains, recurrent SPMNs (Tatavarti et al., 2021) and state-based recurrent SPMNs (Hayes et al., 2021), use template networks that reduces the network size as compared to SPMNs. While these methods learn models of good quality in sequential domains, the template network size remains unbounded and there is a clear need for a flexible approach that trades off quality for network size.

6. Concluding Remarks

Previously introduced structure learning algorithms for SPNs and SPMNs yielded models having no bound over the network size or the computation time. In this paper, we presented anytime or flexible algorithms for the structure learning of valid SPNs and SPMNs. To achieve this, we replace the existing batch steps with flexible techniques for generating the sum and product nodes and their children of increasing complexity. Our empirical results show that these algorithms successfully trade off the quality of the models for reduced network complexity and learning time. As these methods can generate models that display very good fits and optimal expected utilities and coupled with the flexibility, we expect ANYTIMESPN and ANYTIMESPMN to become the default algorithms for structure learning. For instance, if the available time is less than that required by LEARNSPN, we may instead let ANYTIMESPN run until the given time elapses and use the most recently learned SPN. Recurrent SPMNs (Tatavarti et al., 2021) and state-based recurrent SPMNs (Hayes et al., 2021) have been recently presented to model sequential decision-making data, but the sizes of the recurrent network templates remain unbounded. A future direction for this work would be to extend the anytime technique for learning the template structures for wider applicability.

Acknowledgments

This research was supported in part by NSF grant #1815598. We thank Daniel Redder for his editorial and technical feedback and the anonymous reviewers for their comments.

References

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

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

A. Dennis and D. Ventura. Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

N. Di Mauro, F. Esposito, F. G. Ventola, and A. Vergari. Alternative variable splitting methods to learn sum-product networks. In *AI*IA Advances in Artificial Intelligence*, pages 334–346, 2017.

F. Er and S. Lezki. The usage of influence diagram for decision making in textiles. *Asian Social Science*, 8(11):163–169, 09 2012.

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

L. Hayes, P. Doshi, S. Pawar, and H. T. Tatavarti. State-based recurrent spmns for decision-theoretic planning under partial observability. In *Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2526–2533, 2021.

A. Kalra, A. Rashwan, W.-S. Hsu, P. Poupart, P. Doshi, and G. Trimponias. Online structure learning for feed-forward and recurrent sum-product networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

M. Melibari, P. Poupart, and P. Doshi. Sum-product-max networks for tractable decision making. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1846–1852, 2016.

A. Molina, S. Natarajan, A. Vergari, F. Esposito, N. D. Mauro, and K. Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. In *AAAI Conference on Artificial Intelligence*, 2018.

A. Molina, A. Vergari, K. Stelzner, R. Peharz, P. Subramani, N. D. Mauro, P. Poupart, and K. Kersting. SPFlow: An easy and extensible library for deep probabilistic learning using sum-product networks. *arXiv preprint arXiv:1901.03704*, 2019.

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.

D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *International Conference on Machine Learning*, pages 727–734, 2000.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 2551–2558, 2011.

A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *31st International Conference on Machine Learning (ICML)*, pages 710–718, 2014.

S. Sanner. Relational dynamic influence diagram language (RDDL): Language description, 2011.

G. Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, 1978.

H. T. Tatavarti, P. Doshi, and L. Hayes. Recurrent sum-product-max networks for decision making in perfectly-observed environments. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2021.

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, 2015.

S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17:73–83, 1996.

S. Zilberstein and S. Russell. Approximate reasoning using anytime algorithms. In *Imprecise and Approximate Computation*, pages 43–62. Springer US, 1995.