Sum-Product-Max Networks for Tractable Decision Making

Mazen Melibari [§], **Pascal Poupart** [§], **Prashant Doshi** [‡]
[§] David R. Cheriton School of Computer Science, University of Waterloo, Canada
[‡] Dept. of Computer Science, University of Georgia, Athens, GA 30602, USA
[§] {mmelibar,ppoupart}@uwaterloo.ca, [‡]pdoshi@cs.uga.edu

Abstract

Investigations into probabilistic graphical models for decision making have predominantly centered on influence diagrams (IDs) and decision circuits (DCs) for representation and computation of decision rules that maximize expected utility. Since IDs are typically handcrafted and DCs are compiled from IDs, in this paper we propose an approach to learn the structure and parameters of decision-making problems directly from data. We present a new representation called sum-product-max network (SPMN) that generalizes a sum-product network (SPN) to the class of decision-making problems and whose solution, analogous to DCs, scales linearly in the size of the network. We show that SPMNs may be reduced to DCs linearly and present a first method for learning SPMNs from data. This approach is significant because it facilitates a novel paradigm of tractable decision making driven by data.

1 Introduction

Influence diagram (ID) has been the graphical language of choice for probabilistically modeling decision-making problems [Shachter, 1986; Tatman and Shachter, 1990]. IDs extend the probabilistic inference of Bayesian networks with decision and utility nodes to allow the computation of expected utility and decision rules. IDs offer a general language that can represent factored decision-making problems such as completely- or partially-observable decision problems [Smallwood and Sondik, 1973; Kaelbling *et al.*, 1998]. However, unlike Bayesian networks that have witnessed a rich portfolio of algorithms to automatically learn their structure from data [Tsamardinos *et al.*, 2006; Friedman and Goldszmidt, 1998; Friedman and Koller, 2003], no algorithms exist to the best of our knowledge for learning the structure and parameters of IDs from data.

Recent investigations into new models for tractable probabilistic inference such as arithmetic circuits [Huang *et al.*, 2006] and sum-product networks [Poon and Domingos, 2011] that are suited to learn models from large datasets could help fill this gap. Specifically, several approaches to directly learn a network polynomial that is graphically represented as a network of sum and product nodes from data have been devised [Poon and Domingos, 2011; Adel *et al.*, ; Gens and Domingos, 2013; Lowd and Rooshenas, 2013]. Evaluations of the polynomial provide the joint or conditional distributions as desired and are performed in time that is *linear* in the size of the network. Thus, arithmetic circuits and sum-product networks represent a tractable class of inference models compared to the generally intractable inferencing of Bayesian networks.

Motivated by tractable inference, we generalize sum-product networks to a new class of problems that involve probabilistic decision making, in this paper. To enable this, we introduce two new types of nodes: *max* nodes to represent the maximization operation over different possible values of a decision variable, and *utility* nodes to represent the utility values. We refer to the resulting network as a sum-product-max network (SPMN), whose solution provides a decision rule that maximizes the expected utility in linear time. The semantics of the max node is that its output is the decision that leads to the maximal value among all decisions. Analogously to sum-product networks, we introduce a set of properties that guarantee the validity of the SPMN, such that the solution of an SPMN will correspond to the expected utility obtained from a valid embedded probabilistic model and utility function that are encoded by the network. We also show that a SPMN is reducible to a DC in steps linear in the size of the network.

We present methods to learn the structure and parameters of valid SPMNs from decision-theoretic data. Such data not only consists of instances of the random state variables but also possible decision(s) and the corresponding valuation(s). This is a significant advance because it brings machine learning to decision making, which has so far relied on handcrafted expert models. To evaluate new methods for learning SPMNs in this paper and in the future, we establish an initial testbed of datasets each reflecting a realistic non-sequential decision-making problem.

2 Background

Traditional probabilistic graphical models [Koller and Friedman, 2009] such as Bayesian networks, Markov networks and IDs allow a compact representation of probability distributions and decision-theoretic problems. However, the compactness of the representation does not ensure that inference and decision making can be done tractably (inference is #P-hard and decision making is PSpace-hard).

2.1 Arithmetic Circuit and Sum-Product Network

An Arithmetic Circuit (AC) [Park and Darwiche, 2004] consists of a directed acyclic graph of sums and products for the interior nodes and numerical values for the leaves. ACs were initially proposed as compiled representations of Bayesian and Markov networks that allow fast inference. As most time in inference queries is spent deciding what arithmetic operations to perform rather than actually performing the operations, one can cache the arithmetic operations that should be performed for any query into an AC. When a query is received, it is answered quickly simply by performing a bottom-up pass on the AC. While this speeds up inference tremendously, it doesn't change the complexity of inference (still #P-hard). This is because an exponential blow up in the size of the AC may occur while constructing it from a Bayesian or Markov network.

More recently, Poon et al. [2011] proposed sum-product networks (SPN), which are equivalent to ACs in the sense that ACs and SPNs are reducible to each other in linear time and space. An SPN is also a directed acyclic graph of sums and products with the difference that outgoing edges from sum nodes are labeled with numerical values and the leaves are indicator variables. Instead of compiling SPNs from Bayesian networks, which may also yield an exponential blow up, Poon et al. [2011] proposed to learn SPNs directly from data. This ensures that the resulting model is necessarily tractable for inference. In comparison, learning methods for Bayesian and Markov networks yield tractable networks in terms of space, but not always in terms of inference time and their compilation into ACs could be exponentially large. Nevertheless, SPNs learned from data can be converted into proportionally-sized ACs, and more recently techniques have also been presented to learn ACs directly from data [Lowd and Domingos, 2012].

2.2 Decision Circuits

A DC extends an AC with max nodes for optimized decision making. In other words, a DC is a directed acyclic graph where the interior nodes are sums, products and max operators, while the leaves are numerical values and indicator variables. Bhattacharjya and Shachter [2007] proposed DCs as a representation that ensures exact evaluation and solution of IDs in time linear in the size of the network. However, similar to ACs, DCs are obtained by compiling IDs, which may yield an exponential blow up in their size. More recently, separable value functions and conditional-independence between subproblems in IDs is exploited to produce more compact DCs [Shachter and Bhattacharjya, 2010].

3 Sum-Product-Max Networks

We introduce SPMNs and establish their equivalence with DCs.

3.1 Definition and Solution

SPMNs generalize SPNs [Poon and Domingos, 2011] by introducing two new types of nodes to an SPN: max and utility nodes. We begin by defining an SPMN.

Definition 1 (SPMN) *An SPMN over decision variables $D_1, \ldots, D_m$, random variables $X_1, \ldots, X_n$, and utility functions $U_1, \ldots, U_k$ is a rooted directed acyclic graph. Its leaves are either binary indicators of the random variables or utility nodes that hold constant values. An internal node of an SPMN is either a sum, product or max node. Each max node corresponds to one of the decision variables and each outgoing edge from a max node is labeled with one of the possible values of the corresponding decision variable. Value of a max node i is $\max_{j \in Children(i)} v_j$, where $Children(i)$ is the set of children of i, and v_j is the value of the subgraph rooted at child j. The sum and product nodes are defined as in the SPN.*

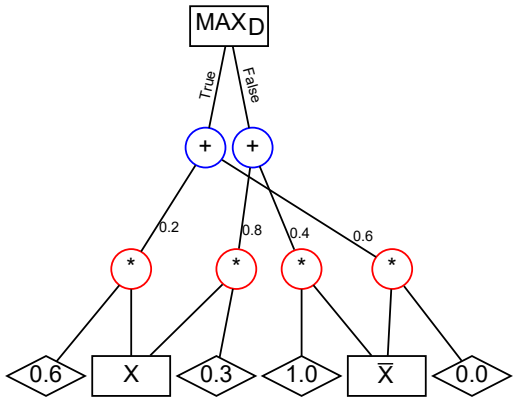

Figure 1: Example SPMN for one decision and one random variable. Notice the rectangular *max* node and the utility nodes (diamonds) in the leaves.

Figure 1 shows a generic example SPMN for a decision-making problem with a single decision D and binary random variable X_1. Indicator nodes $X = T$ and $X = F$ return a 1 and 0 respectively, when the random variable X is true, and vice versa if X is false.

We now turn to recall the concepts of information sets and partial ordering. The information sets $I_0, \ldots, I_m$ are subsets of the random variables such that the random variables in the information set I_{i-1} are observed before the decision associated with variable D_i, $1 \leq i \leq m$, is made. Any information set may be empty and variables in I_m need not be observed before some decision node. An ordering between the information sets may be established as follows: $I_0 \prec D_1 \prec I_1 \prec D_2 \prec \ldots \prec D_m \prec I_m$. This is a partial order, denoted by $\mathcal{P}^{\prec}$, because variables within each information set may be observed in any order.

Next, we define a set of properties to ensure that a SPMN encodes a function that computes the maximum expected utility (MEU) given some partial order between the variables and some utility function U.

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

The scope of a node is the set of all random variables associated with indicators and decision variables associated with max nodes that appear in the SPMN rooted at that node.

Definition 3 (Decomposability of Product Nodes) *An SPMN is decomposable iff no variable appears in more than one child of a product node.*

Definition 4 (Completeness of Max Nodes) *An SPMN is max-complete iff all children of the same max node have the same scope, where the scope is as defined previously.*

Definition 5 (Uniqueness of Max Nodes) *An SPMN is max-unique iff each max node that corresponds to a decision variable D appears at most once in every path from root to leaves.*

Together, these properties allow us to define a valid SPMN.

Definition 6 (Validity) *A SPMN is valid if it is sum-complete, decomposable, max-complete, and max-unique.*

Evaluation An SPMN is evaluated by setting the indicators that are consistent with the evidence to 1 and the rest to 0. Then, we perform a bottom-up pass of the network during which operators at each node are applied to the values of the children. The optimal decision rule is found by tracing back (i.e., top-down) through the network and choosing the edges that maximize the decision nodes.

We may obtain the maximum expected utility of an ID representing a decision problem with a partial order $\mathcal{P}^{\prec}$ and utility function U by using the Sum-Max-Sum rule [Koller and Friedman, 2009], in which we alternate between summing over the variables in an information set and maximizing over the decision variable that requires the information set. Theorem 1 makes a connection between SPMNs and the maximum expected utility as obtained from applying the Sum-Max-Sum rule. We use the notation $S(e)$ to indicate the value of a SPMN when evaluated with evidence e.

Theorem 1 *The value of a valid SPMN S is identical to the maximum expected utility obtained from applying the Sum-Max-Sum rule that utilizes the partial order on the random and decision variables:* $S(e) = \mathrm{MEU}(\mathrm{e} \mid \mathcal{P}^{\prec}, \mathrm{U})$.

Proof of this theorem involves establishing by induction that the bottom-up evaluation of a valid SPMN corresponds exactly to applying an instance of the Sum-Max-Sum rule and is given in [tes, 2016].

3.2 Equivalence of SPMNs and DCs

SPMNs and DCs are syntactically and structurally different, but we establish that they are semantically equivalent. The main difference is that all numerical values in DCs appear at the leaves whereas edges emanating from sum nodes are labeled with weights in SPMNs. We can convert an SPMN into a DC by inserting a product node at the end of each weighted edge and moving the edge weight to a leaf under the newly created product node – this adds two nodes in the corresponding DC for each labeled edge. Hence, SPMNs are more compact than DCs because they contain less nodes, but semantically equivalent. However, the transformation is linear with respect to the number of edges in the SPMN because it involves adding precisely two nodes per labeled edge. In the worst case, the size of the corresponding DC in terms of nodes will be at most thrice the total number of nodes in the SPMN – this increase is proportional.

4 Learning SPMNs

In this section we propose methods to learn the structure and parameters of SPMNs from data. Since these methods generalize existing ones for SPNs, it will be easier to describe how to learn SPMNs, but with the understanding that DCs can be readily obtained from SPMNs as we discussed previously.

4.1 Structure Learning

Our method for learning SPMNs labeled as LearnSPMN generalizes LearnSPN [Gens and Domingos, 2013], which is a recursive top-down learning method for SPNs. This allows automated learning of computational models of decision-making problems from appropriate data. LearnSPMN extends LearnSPN to generate the two new types of nodes introduced in SPMNs: max and utility nodes. Equally important, the generalization also requires modifying a core part of LearnSPN so that the learned structure respects the constraints that are imposed by the partial order $\mathcal{P}^{\prec}$ on variables involved in the decision problem. Algorithm 1 describes the structure-learning method and Fig. 2 visualizes how the algorithm proceeds.

Algorithm 1: LearnSPMN

input : $\mathcal{D}$: instances, $\mathbf{V}$: set of variables, i: infoset index, $\mathcal{P}^{\prec}$: partial order
output:
if $|\mathbf{V}| = 1$ **then**
  if *the variable V in $\mathbf{V}$ is a utility* **then**
    $u \leftarrow estimate\ \Pr(V = True)$ from $\mathcal{D}$;
    return *a utility node with the value u*
  else
    return *smoothed univariate distribution over V*
else
  $rest \leftarrow \mathcal{P}^{\prec}[i+1...]$;
  if $\mathcal{P}^{\prec}[i]$ *is a decision variable* **then**
    for $v \in$ *decision values of* $\mathcal{P}^{\prec}[i]$ **do**
      $\mathcal{D}^{v} \leftarrow$ subset of $\mathcal{D}$ where $\mathcal{P}^{\prec}[i] = v$
    return $\mathrm{MAX}_v\ \mathrm{LearnSPMN}(\mathcal{D}^{\mathrm{v}}, \mathrm{rest}, \mathrm{i}+1, \mathcal{P}^{\prec})$
  else
    Try to partition $\mathbf{V}$ into independent subsets $\mathbf{V}_j$ while keeping $rest$ in one partition;
    if *a partition is found* **then**
      return $\prod_j\ \mathrm{LearnSPMN}(\mathcal{D}, \mathbf{V}_{\mathrm{j}}, \mathrm{i}, \mathcal{P}^{\prec})$
    else
      partition $\mathcal{D}$ into clusters $\mathcal{D}^j$ of similar instances;
      return $\sum_j \frac{|\mathcal{D}^j|}{|\mathcal{D}|} \times \mathrm{LearnSPMN}(\mathcal{D}^{\mathrm{j}}, \mathbf{V}, \mathrm{i}, \mathcal{P}^{\prec})$

LearnSPMN takes as input a dataset $\mathcal{D}$ and a partial order $\mathcal{P}^{\prec}$. Each utility variable in the data is first converted into a binary random variable, say U, independent from other utility variables by using the well-known Cooper transformation [Cooper, 1998]. [1] Specifically, $\Pr(U = true|Parents(U)) = \frac{u-u_{min}}{u_{max}-u_{min}}$ where u_{min} and u_{max} are the minimum and maximum values for that utility variable in the data and $Parents(U)$ is a joint assignment of the variables that U depends on. Next, we duplicate each instance a fixed number of times and replace the utility value of each instance by an i.i.d. sample of *true* or *false* from the corresponding distribution over U. Consequently, utility variables may be treated as traditional random variables in the learning method.

[1] The same Cooper transformation also plays a key role in solving IDs as a probabilistic inference problem.

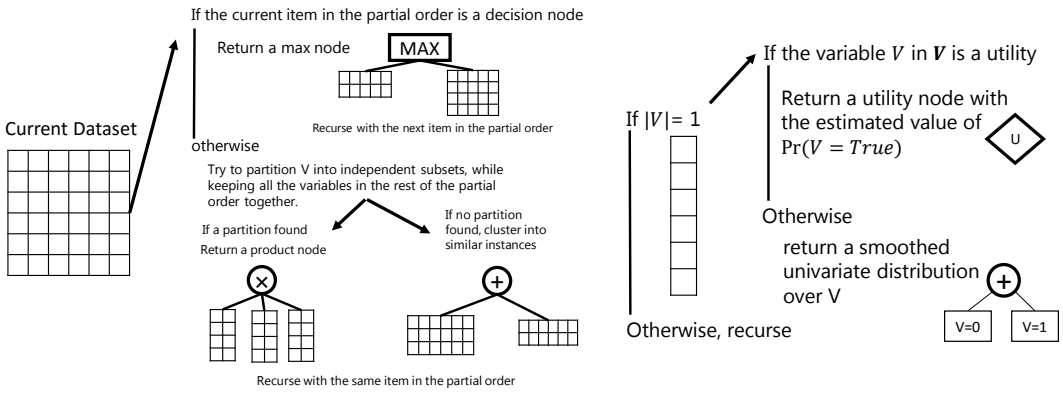

Figure 2: Similar to LearnSPN, LearnSPMN is a recursive algorithm that respects the partial order and extends it to work with max and utility nodes.

Algorithm 1 iterates through the partial order $\mathcal{P}^{\prec}$. For each decision variable D, a corresponding max node is created. For each set $\mathcal{V}$ of random variables in an information set of the partial order, the algorithm constructs an SPN of sum and product nodes by recursively partitioning the random variables in non-correlated subsets and by partitioning the dataset into clusters of similar instances. As in the original LearnSPN, LearnSPMN can be implemented using any suitable method to partition the variables and the instances. For example, a pairwise χ^2 or G-test can be used to find, approximately, a partitioning of the random variables into independent subsets. Clustering algorithms such as EM and K-means can be used to partition the dataset into clusters of similar instances.

Figure 3 shows an example SPMN learned using our generalized structure learning algorithm from decision-making data as described above. The dataset is one of those utilized later in the paper for evaluation.

4.2 Parameter Learning

Let $\mathcal{D}$ be a dataset with $|\mathcal{D}|$ instances, where each instance e_i is a tuple of values of observed random variables denoted as $\mathbf{x}$, values of decision variables denoted as $\mathbf{d}$, and a single utility value u that represents the utility of the joint assignment of values for $\mathbf{x}$ and $\mathbf{d}$; i.e., $e_i = \langle \mathbf{x}, \mathbf{d}, U(\mathbf{x}, \mathbf{d}) = u \rangle$. Algorithm 2 gives an overview of the parameter-learning method. The method is split into two subtasks: (i) Learning the values of the utility nodes, and (ii) learning the embedded probability distribution.

Algorithm 2: SPMN Parameter Learning

input : S: SPMN, $\mathcal{D}$: Dataset
output: SPMN with learned parameters
$S \leftarrow \text{learnUtilityValues}(\text{S}, \mathcal{D})$;
$S \leftarrow \text{SPMN} - \text{EM}(\text{S}, \mathcal{D})$;

Learning the Values of the Utility Nodes

The first subtask is to learn the values of the utility nodes in the SPMN. We start by introducing the notion of *specific-scope*. The *specific-scope* for an indicator node is the value of the random variable that the indicator represents; for all other nodes the *specific-scope* is the union of their childrens' *specific-scopes*. For example, an indicator node $\mathbb{I}_x$ for $X = x$ has the *specific-scope* $\{x\}$, while an indicator node $\mathbb{I}_{\bar{x}}$ for $X = \bar{x}$ has the *specific-scope* $\{\bar{x}\}$. A sum node over $\mathbb{I}_x$ and $\mathbb{I}_{\bar{x}}$ has the *specific-scope* $\{x, \bar{x}\}$.

A product node that has two children, one with *specific-scope* $\{x, \bar{x}\}$ and another one with *specific-scope* $\{y\}$, will have the *specific-scope* $\{x, \bar{x}, y\}$. A simple procedure that performs a bottom-up pass and propagates the *specific-scope* of each node to its parents can be used to define the *specific-scope* of all the sum and product nodes in a SPMN.

Next, for each unique instance e_i in $\mathcal{D}$ we perform a top-down pass where we follow all the nodes whose values in e_i are consistent with their *specific-scopes*. If we reach a utility node, then we increment a counter associated with the value (true or false) of that utility variable in the data. Once all instances are processed, we set each utility node to the ratio of true values (according to the counters) since this denotes the normalized utility based on Cooper's transformation (see Sec. 4.1).

Learning the Embedded Probability Distribution

The second subtask is to learn the parameters of the embedded probability distribution. In particular, we seek to learn the weights on the outgoing edges from the sum nodes. This is done by extending an expectation-maximization (EM) based technique for learning parameters of SPNs [Peharz, 2015] to make it suitable for SPMNs. For each instance e_i in the dataset, we set the indicators to their values in $\mathbf{x}_i$ (the observed values of the random variables in instance e_i). This is followed by computing the expected utility by evaluating the SPMN using a bottom-up pass as described in Section 3. To integrate the decisions $\mathbf{d_i}$, each max node will multiply the value of its children with either 0 or 1 depending on the value of the corresponding decision in the instance. This multiplication is equivalent to augmenting the SPMN with indicators for max nodes. Since our concern is the weights of the sum nodes only in this subtask, all utility nodes may be treated as hidden variables with fixed probability distributions, where summing them out will always result in value 1.

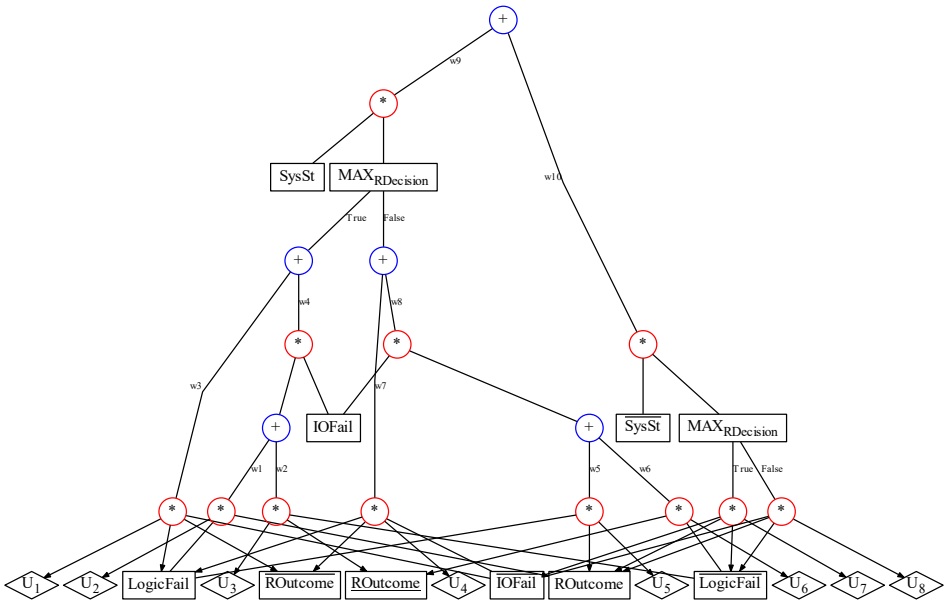

Figure 3: An example SPMN learned from the Computer Diagnostician dataset using LearnSPMN. The partial order used is $\{SysSt\} \prec RDecision \prec \{LogicFail, IOFail, ROutcome\}$. Three different indicators used for ROutcome because it is a ternary random variable

Algorithm 3: SPMN EM Up

input : S: SPMN, e_k: instance
output: SPMN with upward-evaluation values for all nodes
- Set S indicators according to e_k;
for *node i in a bottom-up order of S* **do**
  if *i is a sum node* **then**
    $S_i(k) \leftarrow \sum_{j \in Children(i)} S_j(k)$
  if *i is a product node* **then**
    $S_i(k) \leftarrow \prod_{j \in Children(i)} S_j(k)$
  if *i is a max node* **then**
    $S_i(k) \leftarrow \sum_{j \in Children(i)} \mathbb{I}_{e_k[i]=j} S_j(k)$

Algorithm 4: SPMN EM Down

input : S: SPMN after bottom-up evaluation, e_k: instance
output: SPMN with partial derivatives values for all nodes
for *node i in a top-down order of S* **do**
  if *i is a sum node* **then**
    for $j \in Children(i)$ **do**
      $\frac{\partial S}{\partial S_j} \leftarrow \frac{\partial S}{\partial S_j} + w_{i,j} \frac{\partial S}{\partial S_i}$;
  if *i is a max node* **then**
    for $j \in Children(i)$ **do**
      $\frac{\partial S}{\partial S_j} \leftarrow \frac{\partial S}{\partial S_j} + \mathbb{I}_{e_k[i]=j} \frac{\partial S}{\partial S_i}$;
  if *i is a product node* **then**
    for $j \in Children(i)$ **do**
      $\frac{\partial S}{\partial S_j} \leftarrow \frac{\partial S}{\partial S_j} + \prod_{k \in Children(i)-j} S_k$;

We also perform a top-down pass to compute the gradient of the nodes. The expected counts of each child of a sum node is maintained using a counter for each child. We normalize and assign those values to the edges from the sum nodes at the end of each iteration. This process is repeated until the weights converge. Algorithm 5 gives the algorithm for EM.

Algorithm 5: SPMN-EM

input : S: SPMN, $\mathcal{D}$: Dataset
output: SPMN with learned weights
$S \leftarrow$ randomInitilization(S);
repeat
  for $e_k \in \mathcal{D}$ **do**
    $S \leftarrow$ SPMN EM Up(S, e_k);
    $S \leftarrow$ SPMN EM Down(S, e_k);
    $N_{i,j} \leftarrow 0$ For each child j of sum node i;
    $N_{i,j} \leftarrow N_{i,j} + \frac{1}{S(k)} \frac{\partial S}{\partial i} + S_i(k) \boldsymbol{W}_{i,j}$;
  $\boldsymbol{W}_{i,j} = \frac{N_{i,j}}{\sum_{l \in Chd(N)} N_{i,l}}$
until *convergence*;

5 Experimental Results

We evaluate the LearnSPMN algorithm by applying it to a testbed of 10 data sets whose attributes consist of state and decision variables and corresponding utility values. Three of the datasets were created by simulating a randomly generated directed acyclic graph of nodes whose conditional probability tables and utility tables were populated by values from symmetric Dirichlet distributions. Consequently, these are strictly synthetic data sets with no connection to real-world decision-making problems. The other seven data sets represent real-world decision-making situations in fields spanning different disciplines including health informatics, IT support, and trading. Each of these data set was obtained by simulating an expert system ID. Table 1 gives some descriptive statistics for these data sets such as the number of decision variables in each, the sizes of the data sets, the complexity of solving the underlying expert ID. The real-world datasets and associated metadata are available for download [tes, 2016].

We applied LearnSPMN described in the previous section on each of these datasets. The last column of Table 1 re-

Dataset	#Dec_var	|ID|	|Dataset|	|SPMN|
Random-ID 1	3	116	100K	730
Random-ID-2	5	283	100K	922
Random-ID 3	8	580	100K	2940
Export_textiles	1	10	10K	73
Powerplant_airpollution	2	17	10K	158
HIV_screening	2	46	50K	213
Computer_diagnostician	1	50	50K	186
Test_strep	2	71	200K	205
Lungcancer_staging	3	314	200K	274
Car Evaluation	1	3457	100K	8466

Table 1: Problem, datasets, and learned models statistics. #Dec_var is the number of decisions variables in the problem, |ID| is the total representational size of the influence diagram (total clique size + sepsets), |Dataset| is the size of the dataset, and |SPMN| is the size of the learned SPMN.

ports the size of the SPMN that was learned for each dataset. While the size is usually larger than the total representational complexity of the corresponding ID, the run time complexity of SPMN is *linear* in the size of the network. Furthermore, SPMNs analogous to SPNs tend to have deep structures that are particularly suited to model the hidden variables. On the other hand, the run time complexity of solving the ID may be exponential in the size of the ID.

	MEU			
Data set	ID	SPMN	ID EU	Δ %
Random-ID 1	0.6676	0.6188	0.6676	0
Random-ID 2	0.8159	0.7617	0.8159	0
Random-ID 3	0.9035	0.8832	0.8428	10.30
Export_textiles	0.7068	0.6487	0.7068	0
Powerplant_airpollution	0.7480	0.7281	0.6280	5.39
HIV_screening	0.9497	0.9420	0.9497	0
Computer_diagnostician	0.6740	0.6254	0.6740	0
Test_strep	0.9987	0.9586	0.9987	0
Lungcancer_staging	0.7021	0.6635	0.6957	7.63
Car Evaluation	0.5267	0.4814	0.5267	0

Table 2: Comparison of MEUs of the expert ID (true model) and learned SPMN. The optimal decision rule from the learned SPMN when plugged into the true model yields the EU shown in the fourth column. A discrepancy between the ID's MEU and EU due to the SPMN's decision rule means that the rule from the SPMN does not match the one from the ID. MEU for SPMN is the mean of 10-fold cross-validation. The largest std. error across the folds among all the datasets was 0.00012.

To evaluate the correctness of the learned representation, we exploit the fact that the true model –the expert ID– is also available to us. However, we note that this may not be the case in practice. Subsequently, we solve the SPMN bottom up to compute the MEU and compare it with the MEU obtained from the IDs. We report this comparison in Table 2. Notice that the MEU from the learned SPMN differs from that obtained from the ID. This is expected because the SPMN is learned from a finite set of data that is necessarily an approximate representation of a probabilistic decision-making problem. However, the optimal decision rule may still coincide with that from the ID. Therefore, we enter the decision rule from the SPMN into the ID and report on the obtained EU in the fourth column as well. Notice that it coincides with the MEU from the ID for all but 3 of the datasets. A deeper analysis of the SPMN's decision rule reveals that it differs from the optimal decisions less than or about 10% of the time as reported in the fifth column. We obtained the difference between the two decision rules by executing both on testing datasets and noting the percentage of selected actions that differ.

		MEU time (ms)	
Data set	Learning (s)	SPMN	ID
Random-ID 1	18.20	1.43	39.47
Random-ID-2	22.66	1.92	29.44
Random-ID 3	69.20	4.21	20.76
Export_textiles	1.84	0.21	16.26
Powerplant_airpollu	1.30	0.40	17.44
HIV_screening	8.80	0.57	40.37
Computer_diagnostician	5.69	0.35	17.51
Test_strep	18.93	0.52	16.35
Lungcancer_staging	16.28	0.53	20.70
Car Evaluation	201.87	9.81	27.29

Table 3: Learning time for SPMNs in seconds and a comparison between the MEU computation time of SPMNs and the expert ID in milliseconds.

Finally, we report on the time taken to learn the SPMN and to compute the MEU by both the SPMN and the expert IDs in Table 3. A comparison between the times for the two decision-making representations demonstrates more than an order of magnitude in speed up in computing the MEU by the SPMN given that the two models are available.

6 Concluding Remarks

SPMNs offer a new model for decision making whose solution complexity is linear in the size of the model representation. They generalize SPNs to decision-making problems and are reducible to DCs.We presented an early method to learn SPMNs from non-sequential decision-making data that learns valid networks, which also satisfy any problem-specific partial ordering on the variables. Experiments on a new testbed of decision-making data reveal that the optimal decision rules from the learned SPMNs often coincide with those from the true model – expert system IDs – although the MEU from the learned model differs. Importantly, the time taken to compute the maximum expected utility is more than an order of magnitude less compared to the time taken by IDs. We conclude that SPMN is a viable decision-making model that is significantly more tractable than previous models such as IDs. Importantly, these models can be learned directly from data thereby providing a way to combine machine learning and decision making, which is critically needed for pragmatic applications of automated decision making at a time when large datasets are pervasive. An important avenue for future work is to investigate more efficient structure learning algorithms; here search-and-score templates offer an alternative.

Acknowledgments

This research was supported in part by a grant from Huawei Technologies to Pascal Poupart and Mazen Melibari as well as a grant from ONR to Prashant Doshi with award number N000141310870. The authors acknowledge feedback from participants of the University of Waterloo AI Seminar series. Prashant Doshi performed this research while on a leave of absence at the University of Waterloo, and thanks the University for its support.

References

[Adel *et al.*,] Tameem Adel, David Balduzzi, and Ali Ghodsi. Learning the structure of sum-product networks via an svd-based algorithm.

[Bhattacharjya and Shachter, 2007] Debarun Bhattacharjya and Ross D Shachter. Evaluating influence diagrams with decision circuits. In *Proceedings of the conference on Uncertainty in artificial intelligence*, pages 9–16, 2007.

[Cooper, 1998] G. F Cooper. A method for using belief networks as influence diagrams. 1998.

[Friedman and Goldszmidt, 1998] Nir Friedman and Moises Goldszmidt. Learning bayesian networks with local structure. In *Learning in graphical models*, pages 421–459. Springer, 1998.

[Friedman and Koller, 2003] Nir Friedman and Daphne Koller. Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks. *Machine learning*, 50(1-2):95–125, 2003.

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

[Huang *et al.*, 2006] Jinbo Huang, Mark Chavira, and Adnan Darwiche. Solving map exactly by searching on compiled arithmetic circuits. In *AAAI*, volume 6, pages 3–7, 2006.

[Kaelbling *et al.*, 1998] Leslie Kaelbling, Michael Littman, and Anthony Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[Koller and Friedman, 2009] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[Lowd and Domingos, 2012] Daniel Lowd and Pedro Domingos. Learning arithmetic circuits. *arXiv preprint arXiv:1206.3271*, 2012.

[Lowd and Rooshenas, 2013] Daniel Lowd and Amirmohammad Rooshenas. Learning markov networks with arithmetic circuits. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, pages 406–414, 2013.

[Park and Darwiche, 2004] James D Park and Adnan Darwiche. A differential semantics for jointree algorithms. *Artificial Intelligence*, 156(2):197–216, 2004.

[Peharz, 2015] Robert Peharz. *Foundations of Sum-Product Networks for Probabilistic Modeling*. PhD thesis, Aalborg University, 2015.

[Poon and Domingos, 2011] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Proc. 12th Conf. on Uncertainty in Artificial Intelligence*, pages 2551–2558, 2011.

[Shachter and Bhattacharjya, 2010] Ross Shachter and Debarun Bhattacharjya. Dynamic programming in infuence diagrams with decision circuits. In *Twenty-Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 509–516, 2010.

[Shachter, 1986] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, 1986.

[Smallwood and Sondik, 1973] Richard Smallwood and Edward Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[Tatman and Shachter, 1990] Joseph A. Tatman and Ross D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):365–379, 1990.

[tes, 2016] Evaluation testbed and supplementary file. `https://github.com/decisionSPMN`, 2016. Accessed: April 20, 2016.

[Tsamardinos *et al.*, 2006] Ioannis Tsamardinos, Laura E Brown, and Constantin F Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 65(1):31–78, 2006.