State-Based Recurrent SPMNs for Decision-Theoretic Planning under Partial Observability

Layton Hayes¹, Prashant Doshi¹,², Swaraj Pawar² and Hari Teja Tatavarti³
¹ Institute for AI, University of Georgia, Athens GA 30602
² Dept. of Computer Science, University of Georgia, Athens, GA 30602
³ {layton.hayes25, pdoshi, swaraj.pawar, contactme.hariteja} @uga.edu

Abstract
The sum-product network (SPN) has been extended to model sequence data with the recurrent SPN (RSPN), and to decision-making problems with sum-product-max networks (SPMN). In this paper, we build on the concepts introduced by these extensions and present state-based recurrent SPMNs (S-RSPMNs) as a generalization of SPMNs to sequential decision-making problems where the state may not be perfectly observed. As with recurrent SPNs, S-RSPMNs utilize a repeatable template network to model sequences of arbitrary lengths. We present an algorithm for learning compact template structures by identifying unique belief states and the transitions between them through a state matching process that utilizes augmented data. In our knowledge, this is the first data-driven approach that learns graphical models for planning under partial observability, which can be solved efficiently. S-RSPMNs retain the linear solution complexity of SPMNs, and we demonstrate significant improvements in compactness of representation and the run time of structure learning and inference in sequential domains.

1 Introduction
Bayesian networks (BN) and their extensions, influence diagrams (ID) [Howard and Matheson, 1984], are frameworks for modeling probabilistic dependencies within multivariate distributions over various problem classes. These models are traditionally handcrafted and learning over even small networks is generally intractable. Arithmetic circuits (AC) [Huang et al., 2006], which can be compiled from BNs offer better certainty over the complexity of inference by using a computation-oriented tree structure to model the inference problem. Most inference in ACs is linear in the size of the network, a significant improvement over the exponential inference complexity of BNs. Later, sum-product networks (SPN) [Poon and Domingos, 2011] were developed as a way to efficiently learn similar structures (linearly reducible to each other) directly from data. Just as IDs generalize BNs, decision circuits (DC) [Bhattacharjya and Shachter, 2012] and their SPN-based analog, sum-product-max networks (SPMN) [Melibari et al., 2016a], extend the AC and SPN frameworks, respectively, to model probabilistic decision-making domains, allowing for inference and calculation of maximum expected utility over these domains in time linear in the size of the network. Melibari et al. [2016a] shows that SPMNs are efficiently reducible to DCs in time that is linear in the size of the SPMN. Likewise, recurrent SPNs (RSPN) [Melibari et al., 2016b] generalize SPNs in the same way that dynamic BNs extend BNs, providing a framework for learning graphical models for sequential domains. Most inference in RSPNs is linear in the size of the network, but by exploiting the recurrent structure of sequential domains through a repeated template network, RSPNs dramatically reduce the structure size required to model sequential data, thus improving inference speed. However, no dynamic extension of DCs to sequential contexts has been presented.

We present the state-based recurrent SPMN (S-RSPMN) which exploits the recurrent structure in sequential decision-making (planning) problems. S-RSPMNs draw inspiration from the recurrent template that RSPNs add to SPNs to provide an analogous generalization of SPMNs, with a focus on partially-observed contexts. An S-RSPMN template can be viewed as a collection of interlinked SPMN structures, whose links can be followed to repeat the structure and model sequences of arbitrary lengths. We adapt the invariance property of RSPN’s templates to ensure that an S-RSPMN is valid. To guide the learning of the template’s substructures and generate links between them, we utilize a data augmentation and state matching process, which serves to identify distinct belief states within the problem domain and their transitions. The substructures of an S-RSPMN are learned similarly to learning an SPMN. We test the performance of the learning algorithm by learning S-RSPMNs on a testbed of several sequential decision-making domains from OpenAI’s Gym [Brockman et al., 2016] and RDDLSim [Sanner, 2010], demonstrating that they result in nearly optimal policy values for each. We also demonstrate competitive performance of policies learned by S-RSPMNs against a recent neural network approach, batch-constrained Q-learning [Fujimoto et al., 2019b], on the testbed. S-RSPMNs are thus a novel model-based representation for decision-theoretic planning in environments possibly partially observed, and which can be learned directly from offline data.

2 Background on SPMNs
SPNs [Poon and Domingos, 2011] are generative probabilistic graphical models, which can be learned directly from data.
Their simple structure, along with some validity constraints, means that probabilities for given evidence can be correctly computed in time linear in the size of the network for most types of queries (an exception being marginal MAP queries).

An SPN represents a joint probability distribution over its variables \( X_1, X_2, \ldots, X_n \). It is a rooted directed acyclic graph (DAG) whose leaves are the distributions of the random variables and whose internal nodes are sums and products. Each outgoing edge from a sum node has a non-negative weight. The value of a product node is the product of the values of its children while the sum node’s value is the weighted sum of its children’s values. The value of a SPN is the value of its root which can be represented as a network polynomial [Darwiche, 2003]. The SPN is valid if it is sum-complete and decomposable.

The SPN is valid if it is sum-complete and decomposable. Each max node corresponds to a decision variable \( D \), and the variables of an information set \( I_{i-1} \) are observed before decision variable \( D_i \), and the variables of \( I_i \) are observed only after the decision at \( D_i \) is made. This is a partial ordering over the variables because the ordering of the variables in any information set is unspecified.

The partial order is respected while learning the structure of an SPMN in that the variables of information set \( I_{i-1} \) are outside the scope of the decision node corresponding to decision variable \( D_i \) (coming earlier in the structure), whereas all variables \( I_i, I_{i+1}, \ldots \), are within its scope.

MEU calculation in an SPMN is performed by propagating the values of each utility node along side the probabilities. First, the leaf nodes consistent with the given evidence are set to 1 and the rest to zero. Then the structure is evaluated bottom-up, where the operators at each node are applied to the values of the children. In this way, the expected value at each node is determined, with the overall MEU collected at the root. To determine the optimal decision values given the evidence, the network can then be traversed top-down, selecting the decisions corresponding to the child with greatest expected utility at each decision node.

This complexity of the evaluation is thus linear with the number of nodes in the network. However as with SPNs, the number of nodes in the structure may grow exponentially with the number of variables, rendering SPMNs largely intractable in many sequential decision-making domains.

3 State-Based Recurrent SPMNs

A straightforward generalization of RSPNs to decision-making problems thereby yielding recurrent SPMNs exists [Tatavarti et al., 2021], but this targets contexts whose state is perfectly observed. In contrast, we aim to introduce a model for partially-observable environments where the data consists of noisy observations (not values of state variables), which complicates the generalization.

The planning data for learning S-RSPMNs consists of a finite temporal sequence of values of observation, decision, and utility variables. Formally, consider a partially-observable decision-making problem where the state of the environment is informed by \( n \) observation variables, \( \Omega_1, \Omega_2, \ldots, \Omega_m \); decisions by a combination of \( m \) decision variables, \( D_1, D_2, \ldots, D_m \); and a single utility variable \( U \). A candidate data record of at most \( T \) steps is then a sequence of
T tuples of the form \((I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^T\), \((I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^T\), \(\ldots\), \((I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^T\)). Here, \(I_0, I_1, \ldots, I_m\) are information sets, as mentioned in Section 2, but where \(I_{T-1}\), \(1 \leq i \leq m\) consists of values of the observation variables in the information set of \(D_i\). Additionally, \(u\) in each tuple is the value of utility variable \(U\) given the realizations of the hidden state variables and observed decisions in that tuple. Next, we define the template, a key component of S-RSPMNs.

**Definition 6 (S-RSPMN template).** A template network for a slice of \(n\) observation variables at time \(t\) is a DAG with \(k\) roots and \(k + l + n\) leaf nodes, where \(k > 0\), \(l > 0\). The \(n\) leaf nodes hold the distributions over observation variables, \(\Omega_1, \Omega_2, \ldots, \Omega_n\) or hold constant values as utility nodes. The remaining \(k\) and \(l\) leaves consist of two types of distinguished nodes, respectively denoted as \(S_1\) and \(S_2\). Each of the \(k\) roots is a product node which has exactly one \(S_1\) node as a direct child and contains at least one \(S_2\) node in its scope. The interior nodes of the template consist of sum, product, and max nodes and the edges have their usual semantics.

The template network can be viewed as a collection of interlinked SPMN substructures, each of which we refer to as a state estimation branch. Each such branch is rooted at a product node and corresponds to a distinct belief region in the problem with its \(S_1\) node containing a collection of integer labels. Each \(S_2\) leaf node in the template links to the state branch whose \(S_1\) node contains its label. These links represent the transition from one belief region to another at the conclusion of a step of the sequence.

Next, we introduce the top and bottom networks, which allow the S-RSPMN to yield a rooted SPMN when the templates are repeated as many times as the length of the sequence data.

**Definition 7 (Top network).** A top network is a rooted DAG whose root is a sum node with \(k\) leaves as children. Leaves of this network are interface nodes, each of which will be merged with a distinct root of the template. Each edge \((i, j)\) emanating from the root sum node has a non-negative weight \(w_{ij}\). A bijective mapping \(g\) determines which root of the template corresponds to an interface node of the top network.

**Definition 8 (Bottom network).** A bottom network modeling the last time slice of \(n\) observation variables, \((\Omega_1, \Omega_2, \ldots, \Omega_n)^T\), is similar to the template network with \(k\) roots but \(k + n\) leaf nodes. No \(S_2\) nodes are present.

To model sequential decision-making data of length \(T\), we construct an S-RSPMN by learning the top and template networks. Then, \(T - 1\) instances of the template network are stacked on the bottom network, and capped with a top network. To stack instances of the template network, each \(S_2\) node is replaced with the root node of the template’s state estimation branch whose \(S_1\) node contains its value. The top network is then merged with this structure by replacing its interface nodes with the corresponding roots as mapped by \(g\).

The validity of an S-RSPMN over any number of steps can be established by repeating the template, as described above, to the desired number of steps and verifying the conditions given in Defs. 1, 2, 4, and 5 for each node. However, we may define a property to allow establishing the validity without unrolling the network. This property is inspired by the template invariance property of RSPNs [Melibari et al., 2016b].

**Definition 9 (Template soundness).** The template network presented in Def. 6 is sound iff (a) the scope of the structure replacing each \(S_2\) node excludes \(S_1^1, \Omega_1, \ldots, \Omega_n\), and is identical to the scope of each other such structure, (b) all product nodes in the template network are decomposable, (c) all sum nodes in the template are complete, and (d) all max nodes in the template are both max-complete and max-unique.

This property is helpful in proving the validity of a network in that the roots of a sound template satisfy the conditions of soundness when they are used to replace the \(S_2\) nodes of another copy of the same template network modeling the previous step of the sequence. Theorem 3 shows that this property can be used to establish the validity of an S-RSPMN modeling an arbitrary number of steps.

**Theorem 3 (S-RSPMN validity).** If the (a) top network is sum-complete and bottom network is sum- and max-complete, max-unique, and decomposable, (b) scope of each state branch in the bottom network is identical to the scope of every other such branch, and (c) \(S_2\) nodes can be replaced with corresponding state branches in such a way that the template network is sound, then the corresponding S-RSPMN is valid.

The proof, given in the technical appendix included in the supplement, is established by induction on the sequence length and observing that the conditions (a-c) always yield an unrolled SPMN that is valid.

### 4 Learning S-RSPMN from Data

Sequential decision-making problems can be modeled using SPMNs without recurrence by simply blocking the entire sequence of tuples in a record as one data input and applying the LearnSPMN algorithm of Melibari et al. [2016b]. However, as our experiments demonstrate, this approach often results in very large structures whose size is exponential in the number of variables, and also exponential in the number of time steps of a sequence. This proves intractable for longer sequences. Furthermore, the SPMNs once learned are unable to model any newer sequences whose length is greater than the longest sequences in the training data.

Consequently, there is strong motivation for an algorithm that automatically learns the structure of an S-RSPMN template and top networks, such that the structure satisfies the property of soundness given in Def. 9. Overall, our method for learning the S-RSPMN structure iteratively applies LearnSPMN over small numbers of steps to learn multiple SPMN substructures. These are linked together through a matching process to form a recursive structure.

To help illustrate the process, we introduce a simple sequential decision-making problem, the *repeated marbles game*. It starts with a bag containing 2 marbles, one white and the other black, unknown to the agent. The agent can chose to either draw a marble randomly or to reset the problem to the initial state. The problem is also automatically reset to the initial state once both marbles are drawn. Drawing the white marble results in +1 utility, whereas drawing the black marble results in -1 utility, so the optimal play is to draw until the white
marble is found and then reset the problem, resulting in an average of +0.5 utility per 2 steps of the problem. The state is represented by a single variable, \( X \), which corresponds to which marble is left in the bag. The drawn marble is observed perfectly thereby yielding a single observation variable \( \Omega \). A single decision variable \( \text{draw} \) involves choosing one of two actions: draw a marble or reset.

Algorithm 1 gives the main procedure, \textsc{LearnS-RSPMN}, for learning the S-RSPMN template. As we may expect, \textsc{LearnS-RSPMN} takes the set \( D \) of records each containing a sequence of tuples and a partial order over the domain variables \( P \prec \). Additionally, two learning parameters need to be pre-specified: (i) the horizon, \( h \), which is the number of time steps considered in each application of the underlying LearnSPMN. Increasing \( h \) allows us to model longer-term dependencies but doing so will also increase the run time of the algorithm. (ii) The correlation threshold \( \theta_{\text{thresh}} \) is used in the matching process to determine whether a correlation between an existing substructure and variable values is not sufficiently significant thereby necessitating a new substructure.

The algorithm begins by augmenting each data record (lines 1-4). It prepends an additional variable to the partial order of each step of a sequence, labeled as \( S_1 \). The variable takes a state identifier (SID) as its value. Specifically, each augmented tuple in the data record takes the form \((S_1, I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^\tau\). SID for the \( S_1 \) in the initial time-step tuple of each record is 0, signifying the initial region. The \( S_1 \) values for each subsequent step is set based on an evaluation of the network over its preceding step in a process described later.

Next, we learn the first substructure by applying LearnSPMN to the tuples of the first \( h \) time steps of each data record (line 5). This learns a state estimation branch for the initial state of the sequence, SID 0. As \( S_1 \) is first in the partial order and takes only one value in the data, it is always a child of a root product node. We illustrate the state estimation branch with SID 0 for the repeated marbles game in Fig. 1.

![Figure 1: An SPMN learned for the first two time steps of the repeated marbles game. The \( S_1 \) leaf with the SID value of 0 is highlighted in blue.](image)

Algorithm 1: \textsc{LearnS-RSPMN}

\begin{algorithm}[h]
\begin{algorithmic}[1]
\State \textbf{Input:} Data set: \( D = (\tau^0, \tau^1, \ldots, \tau^{T-1})^\epsilon \) where \( \epsilon = 1, 2, \ldots, E \). Partial order over all variables for two steps: \( P \prec \). Horizon: \( h \) \triangleright number of steps used in structure learning and branch matching. Correlation threshold: \( \theta_{\text{thresh}} \).
\State \textbf{Output:} learned S-RSPMN \( \text{SPMN}^0 \).
\State \textbf{Prep} a column for SID, \( S^{\prec} \), to each tuple \( \tau^e \) in each episode \( e \) of \( D \) with NULL value.
\State \textbf{Prep} a new information set containing only SID to each time step of \( P \prec \).
\State Set SID for the initial tuple \( \tau^0 \) of each episode \( e \), \( S^{\prec}_1 \) \leftarrow 0.
\State numSIDs \leftarrow 1. SIDtoBranch \leftarrow new map.
\State \( \text{SPMN}^0 \leftarrow \text{LEARNSPMN}(D^{\prec h-1}, P^{\prec}) \).
\State \( \text{SPMN}^0, \text{numSIDs} \leftarrow \text{REPLACEBRANCHES}(\text{SPMN}^0, P^{\prec}, \text{numSIDs}) \).
\State SDoBranch(0) \leftarrow \text{SPMN}^0.
\State S-RSPMN \leftarrow new root sum node with SPMN as its child \triangleright top network.
\State \textbf{do}
\State \( D, \text{maxDataSID} \leftarrow \text{UPDATEDATACOUNTS}(D, \text{SDoBranch}, \text{numSIDs}) \).
\State matched \leftarrow False.
\State \textbf{for} branch in the children of S-RSPMN \textbf{do}
\State \textbf{if} BRANCHMATCH(D, branch, maxDataSID, h, \( \theta_{\text{thresh}} \)) \textbf{then}
\State \hspace{1em} add maxDataSID to the list of SIDs in the \( S_1 \) node of branch.
\State \hspace{1em} SDoBranch[maxDataSID] \leftarrow branch.
\State \hspace{1em} matched \leftarrow True.
\State \hspace{1em} \textbf{exit} for loop.
\State \textbf{if} not matched \textbf{then}
\State \hspace{1em} newBranchData \leftarrow (\tau^i, \ldots, \tau^{i+h-1} \epsilon) for each tuple \( \tau^i \) and episode \( \epsilon \) in \( D \) where \( S^{\prec}_1 \) \leftarrow \text{maxTuplesSID} \).
\State \hspace{1em} SPMN^i \leftarrow \text{LEARNSPMN}(\text{newBranchData}, P^{\prec})\).
\State \hspace{1em} \text{SPMN}^i, \text{numSIDs} \leftarrow \text{REPLACEBRANCHES}(\text{SPMN}^i, P^{\prec}, \text{numSIDs}) \).
\State S-RSPMN \leftarrow add SPMN as a new child of the root sum node of S-RSPMN.
\State SDoBranch[\text{maxDataSID}] \leftarrow newBranch.
\State \textbf{while} maxDataSID is not NULL.
\State \textbf{return} S-RSPMN.
\end{algorithmic}
\end{algorithm}

While this SPMN models data over multiple time steps, recall that the template focuses on one time step only. We include the additional \( h - 1 \) steps in the learning (rather than focusing on the data at time step 0 only) to model the subsequent transitions, which ensures sufficient branching in the structure as we see in Fig. 1. Taking a step toward creating the template, the additional structure is then pruned away and replaced by \( S_2 \) nodes (line 6). These nodes replace the first occurrence of a variable from the next time step (often the decision variable as in the example). Each new \( S_2 \) is assigned a value that is incremented by 1, starting at 1. Algorithm 2 \textsc{REPLACEBRANCHES} included in the supplement gives the procedure of replacing the branches rooted at such variables with \( S_2 \) nodes. This creates a simpler structure, as we illustrate...
in Fig. 2 for the marbles game.

This first substructure is added as a child to a new sum node which serves as the root of the template network. The resulting structure is sufficient to model only the first state of the problem. In order to model additional states and the transitions between them, we use the newly created $S_2$ nodes to augment the next step tuple of data with SID values. Each $S_2$ node represents a state transition, and its corresponding SID value will serve to tell us to which state estimation branch it connects.

For each sequence, we determine the SID value of its next step by finding the $S_2$ node reached by the current step. Toward this, we obtain the $S_2$ node that yields the maximum likelihood for each state estimation branch for the current time-step tuple. We obtain this by assigning a value of 1 to each $S_2$ node in turn and assigning other $S_2$ nodes a value of 0. Then, calculate the likelihood for the data in the current tuple, say time step 0, and select the $S_2$ node that was assigned 1, which yielded the largest likelihood. This is analogous to selecting the $S_2$ node yielding the most probable explanation (Algorithm 1). This indicates that the belief region represented by the matched state estimation branch covers the belief represented by the data sequence. If no match is found, then the SID may represent a new belief state. In that case, we learn a new SPMN substructure. First, the learning data is comprised of the tuple at time step $t$ that did not match, tuples at step $t$ from other records whose $S_1$ value is identical to the SID, and the tuples of the next $h - 1$ steps in each data record until the last step if it is reached prior to the time step $t$. We apply LearnSPMN to this data subset to obtain a new state estimation branch. As before, we replace the subgraphs whose scopes fall outside the set of variables at time step $t$ with new $S_2$ nodes. The resulting SPMN substructure is added as a new child of the root sum node of the template network.

_weights on the edges correspond to the relative frequency with which belief states in each region are encountered in the data.

The matching process is performed for the next tuple in each data record. If a new SPMN substructure is learned, $S_2$ nodes corresponding to the most probable explanation are found and the matching process is repeated as we move across all the time steps of the data. Figure 3 shows a complete, fully-linked S-RSPMN for the repeated marbles game with three state-estimation branches each corresponding to a distinct belief region – for the states where both marbles are in the bag and the black or white marble only remains in the bag.

SID of most likely $S_2$ nodes based on the last time-step tuple cannot be assigned forward to the next $S_1$ node as there is no further tuple of data. Nevertheless, we must assign them to some $S_1$ node(s) to model the transition in the belief state. Our approach is to add the $S_2$ node’s value to the set of SIDs held in the $S_1$ nodes of all the state-estimation branches in the
template. This is analogous to modeling the transition to a distribution of next belief regions.

5 MEU Evaluation

A brute force approach to MEU evaluation in an S-RSPMN would be to unroll the structure to the desired planning horizon by recursively replacing the $S_2$ nodes with their matching branch – this is the branch whose $S_1$ node contains the $S_2$ node’s SID value. However, we can dramatically speed up the MEU evaluation by exploiting the recurrent properties of the network with dynamic programming. Beginning at the last time step (horizon 1), we prune the template to yield the bottom network (Def. 8). The MEU for each state-estimation branch is then the highest EU at the product nodes that form the roots of the branches in the bottom network, using a bottom-up pass through each branch. We may associate these MEUs with the identifying $S_1$ nodes of the branches.

To calculate the value of a state-estimation branch for two steps of unconditioned evaluation, we iterate through each possible decision path in the branch and add the weighted sum over the $S_2$ nodes reachable in that path to the single step value of the path. The value of an $S_2$ node here is equal to the single-step value of the state-estimation branch to which it is linked. This value was computed as mentioned in the previous paragraph. For each branch, we then take the largest combined value from among its possible decision paths as the MEU value for that branch for a horizon of two. This procedure is repeated, using the branch values for two steps to obtain the values for a horizon of three, and so on, to the specified $T$. By memoizing the MEU values for each branch and time step, we can quickly determine the unconditioned MEU up to any $T$. This effectively reduces the run time complexity from $O(b^T)$ to $O(\log b^T)$, where $b$ is the worst-case branching factor of the network, and the remaining parameters are as defined in Section 2.

6 Experiments

The LEARNS-RSPMN algorithm has been implemented in the SPFlow library [Molina et al., 2019] and is available on GitHub at https://github.com/minimum-LaytonC/SPFlow/tree/rsmpn_rdc_rmeufix under the Apache license. We evaluate its performance on a new testbed of sequential decision-making data sets that adhere to the schema given in Section 3 and where the state is partially observed.

As there are very few existing data sets on simulations of discrete partially observable decision-making domains, we developed a new testbed of eight data sets on decision-making problems, listed in Table 1 and available at https://github.com/minimum-LaytonC/SRSPMNDataset_Generators. In addition to the repeated marble and Tiger problems, we also include simulations of the Tiger problem, a well-known POMDP domain [Kaelbling et al., 1998]. Two of the remaining data sets are simulations of the NChain and Frozen lake domains, which are fully-observable problems (where observation variables are synonymous with state variables) sourced from OpenAI’s Gym [Brockman et al., 2016]. The remaining four data sets (Crossing traffic, Elevators, Skill teaching, and Navigation) are simulations of larger RDDLSim POMDP domains [Saner, 2010], which are partially observable. Each data set is generated by using a random policy which interacts with the environment and collecting the action, observation, reward generated at each step. Each episode is run until either the goal state or some other terminal state is reached, after which some of the domains automatically restart. Each episode generates a data record.

The advantages of S-RSPMNs over SPMNs in sequential domains can be seen clearly in their comparative performance on the toy repeated marbles and Tiger problems. Table 2 shows the learning times and structure sizes over several different sequence lengths $T$ and numbers of episodes. We selected the number of episodes for each sequence length that is sufficient for both models to obtain MEU values that are optimal with some exceptions. The size of the S-RSPMN does not significantly change with increase in sequence length as the template structure learned is sufficient to model any number of steps. Observe that its learning time is effectively linear in the size of the data set. Contrast this with the exponential increase in learning time and structure sizes of the SPMNs with sequence lengths. While we may not see this level of improvement for other problems (many problems will have
Table 3: S-RSPMN and BCQ performances on the Gym and RDDLSim domains. Learning time for BCQ is the time taken to run a million iterations. All other BCQ parameters such as the number of samples and loopback values were set to default. Learning an S-RSPMN requires setting two parameters: horizon \( h \), correlation threshold \( \text{ct}_{\text{thresh}} \). Both S-RSPMN and BCQ models for all domains except Navigation were run for 100 steps (to obtain near-converged values) whereas the Navigation models were evaluated over 10 steps. All models were learned on a PC with Intel Xeon ES-2603, RHEL7, 16GB RAM.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Optimal EU</th>
<th>MEU</th>
<th>Mean total reward</th>
<th>Learn time(s)</th>
<th>#nodes ((h, \text{ct}_{\text{thresh}}))</th>
<th>MEU</th>
<th>Mean total reward</th>
<th>Learn time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NChain</td>
<td>25.8</td>
<td>25.86</td>
<td>24.54 ± 0.26</td>
<td>111</td>
<td>106 ((2, 0.3))</td>
<td>-3.28</td>
<td>-8.025</td>
<td>32,196</td>
</tr>
<tr>
<td>Frozen lake</td>
<td>0.823</td>
<td>0.816</td>
<td>0.742 ± 0</td>
<td>4,081</td>
<td>649 ((3, 0.3))</td>
<td>0.0</td>
<td>0.0</td>
<td>37,878</td>
</tr>
<tr>
<td>Skill teaching</td>
<td>5.953</td>
<td>-6.136</td>
<td>5.198 ± 0.98</td>
<td>17,020</td>
<td>1,447 ((3, 0.26))</td>
<td>-3.29</td>
<td>-18.953 ± 0.85</td>
<td>36,620</td>
</tr>
<tr>
<td>Elevators</td>
<td>-14.44</td>
<td>-1.953</td>
<td>-19.103 ± 0.22</td>
<td>4,941</td>
<td>1,324 ((3, 0.24))</td>
<td>-1.953</td>
<td>-19.882 ± 0</td>
<td>38,548</td>
</tr>
<tr>
<td>Crossing traffic</td>
<td>-3.909</td>
<td>-3.28</td>
<td>-5.635 ± 0.29</td>
<td>2,489</td>
<td>203 ((4, 0.22))</td>
<td>-5.809</td>
<td>-8.025 ± 0</td>
<td>36,059</td>
</tr>
<tr>
<td>Navigation</td>
<td>-7.713</td>
<td>-5.809</td>
<td>-8.025 ± 0</td>
<td>7,519</td>
<td>563 ((3, 0.30))</td>
<td>-5.809</td>
<td>-8.025 ± 0</td>
<td>36,059</td>
</tr>
</tbody>
</table>

a larger space of belief regions that yield distinct behaviors), these simple problems serve to demonstrate the advantages of a recurrent model in terms of both time and compactness of representation. Indeed, we were unable to generate SPMNs for even small time steps for some of the other relatively larger domains with structure learning taking many hours before ultimately failing due to excessive memory requirements.

S-RSPMNs are related in principle to batch (or off-policy) reinforcement learning methods [Lin, 1992; Ernst et al., 2005; Lange et al., 2012]. Both these types of methods seek to derive an optimal policy from a given set of prior experiences though S-RSPMNs learn a model while the latter tend to be model free. Therefore, as a baseline, we compare with a recent off-policy reinforcement learning technique constrained to learn from a batch of data, a discrete implementation of the batch-constrained deep Q-Learning (BCQ) [Fujimoto et al., 2019a; Fujimoto et al., 2019b] applied to the testbed. Table 3 reports on the performances of the learned S-RSPMN and BCQ for the Gym and RDDLSim data sets. The optimal EU for the Gym domains is provided by converged deep Q-networks while the symbolic Perseus solver [Hoey et al., 1999] provides the same for the RDDLSim domains. Clearly, a key question to ask is: how good are the policies learned by S-RSPMN and BCQ? We simulated the policies obtained from the learned models and report the mean ± standard deviation of the total reward across 10 runs of 100 episodes each. Table 3 shows that the mean rewards from S-RSPMNs are close to the optimal EU for all the domains, and the difference between the two is significantly less than or similar to that of the mean rewards from BCQ (the sole exception is Elevators). Simultaneously, learning times for S-RSPMNs are often an order of magnitude less than those of BCQ.

However, the learned S-RSPMNs yielded MEUs that did not always align with the rewards from policy simulation! This is evident for the partially-observed RDDLSim domains and indicates that an accurate model of the environment was not learned in these cases, possibly due to the absence of the state variables from the data. But, the MEUs were indeed computed quickly taking just 0.005s and 0.007s for the NChain and Frozen lake models, respectively.

7 Conclusion

Decision-theoretic planning has relied on accurately specifying the model and (optimal) planning remains intractable for all but the simplest problems. This paper seeks to address both these challenges by infusing data-driven machine learning into planning with a fundamental focus on tractability. Whereas the default approach for bringing efficiency to planning is approximations that trade off quality, our approach learns fitted problem representations that are guaranteed to solve efficiently. Task simulation data, which is seldom directly used for automated planning, can now be collected and used. A concomitant risk to this potentially high payoff is that the size of S-RSPMNs is unbounded. Furthermore, S-RSPMNs offer clear semantics for decision making in contrast to neural nets. A direction of future work is to develop anytime techniques for learning the networks, which offer flexible control over the size of the network while yielding structures with improving likelihoods and MEUs.

Acknowledgements

This research was supported by NSF grant #IIS-1815598. We thank Scott Sanner, Alan Fern, and Murugeswari Issakkimuthu for guidance in making our RDDLSim installation operational and for providing their Python RL interface to RDDL. We acknowledge the anonymous reviewers for their feedback.

References


