Recurrent Sum-Product-Max Networks for Decision Making in Perfectly-Observed Environments

by

Hari Teja Tatavarti

(Under the Direction of Prashant Doshi)

Abstract

Recent investigations into sum-product-max networks (SPMN) that generalize sum-product networks (SPN) offer a data-driven alternative for decision making, which has predominantly relied on handcrafted models. SPMNs computationally represent a probabilistic decision-making problem whose solution scales linearly in the size of the network. However, SPMNs are not well suited for *sequential* decision making over multiple time steps. In this paper, we present recurrent SPMNs (RSPMN) that learn from and model decision-making data over time. RSPMNs utilize a template network that is unfolded as needed depending on the length of the data sequence. This is significant as RSPMNs not only inherit the benefits of SPMNs in being data driven and mostly tractable, they are also well suited for sequential problems. We establish conditions on the template network, which guarantee that the resulting SPMN is valid, and present a structure learning algorithm to learn a sound template network. We demonstrate that the RSPMNs learned on a testbed of sequential decision-making data sets generate MEUs and policies that are close to the optimal on perfectly-observed domains. They easily improve on a recent batch-constrained reinforcement learning method, which is important because RSPMNs offer a new model-based approach to offline reinforcement learning.

Index words: Tractable Probabilistic Models, Machine Learning, Sequential Decision Making, Batch Reinforcement Learning

Recurrent Sum-Product-Max Networks for Decision Making in Perfectly-Observed Environments

by

Hari Teja Tatavarti

B.Tech., SASTRA University, India, 2015

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

MASTER OF SCIENCE

Athens, Georgia

2020

Recurrent-Sum-Product-Max Networks

for tractable decision making on sequential data

by

Hari Teja Tatavarti

Dedication

I dedicate this to Amma (Mom) and Nanna (Dad)

Acknowledgments

Firstly, I am grateful to my advisor Dr. Prashant Doshi for putting his trust in me. His consistent guidance and discussions during weekly meetings helped me gain a deep understanding of my thesis area in particular and research in general. I express my heartfelt gratitude to Dr. John Miller and Dr. Sheng Li for being in my committee.

I thank Layton Hayes for his insights. Our consistent discussions helped in improving my algorithm. I thank Prasanth Suresh for proof reading my thesis.

I am thankful to the Institute of AI and all its faculty and staff, specifically Dr. Maier, Dr. Rasheed, Dr. Goodie and Ms. Sonya, for being extremely nice and supportive throughout this journey. I express special thanks to all the THINCLab members and my lab mates Aditya, Christian, Ehsan, Muhammed, Nihal, Prasanth and Saurabh for being a fun as well as an intellectually stimulating gang. Thanks a ton to all my friends and family in the US and overseas for being helpful and supportive.

Finally, I express my profound gratitude and love to my mom, for being my backbone during the most difficult of times and, my dad, for his consistent efforts to provide me with good education.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Probabilistic Graphical Models [12] encode complex distributions compactly using graph-based representation based on relationships between the variables. The graph-based representation makes transparent the conditional dependence and independence between the variables which enables a *factored distribution* of the joint distribution over the variables. This transparent structure also allows for efficient *inference* based on observations using the encoded distribution in a graph. They also facilitate *learning* these models with a data-driven approach.

These Probabilistic Graphical Models using some domain knowledge from a human expert can learn distributions from the data. But the inference in such models is often intractable. Since learning uses inference as a sub-routine, learning also becomes intractable.

In general, Bayesian Networks are used for probabilistic reasoning, Influence Diagrams are used for modelling decision problem and Dynamic Bayesian Networks are used for reasoning over sequence data. Dynamic Influence Diagrams (DIDs) are used for sequential decision making. But, data-driven learning is well-studied only in Bayesian Networks and remains a challenge in other graphical representations mentioned above. Moreover, inference is *NP-hard* in all these models.

1.1 Sum-Product Networks and Extensions

Sum-Product-Networks[22] (SPNs) are a class of Graphical Models which can be learned directly from data. These graphical models are appealing because most types of inference can be performed in time that is *linear* in the size of the network. On the other hand, inference in Bayesian networks is generally exponential.

SPNs are based on the concept of network polynomial [2] which is a representation of the probability distribution of a Bayesian Network as a polynomial. Evaluations of the polynomial provide the joint or conditional distributions as desired. But the network polynomial is itself exponential in the number of variables. Arithmetic circuits [10] and sum-product networks (SPN) [23] can compactly represent a network polynomial as a network of sum and product nodes. Although, one limitation of SPNs is that the size of the learned network is not bounded. In [23], the notion of validity of SPN is given which is important for computing the correct probability distribution. An SPN is valid if it performs a marginalisation in the same manner as a network polynomial.

Given the overall benefit of these generative models, recurrent SPNs as a generalization of SPNs are introduced in [16] for modeling sequence data of varying length. In particular, if a recurrent SPN is a valid SPN, inference queries can be answered in linear time, thereby providing a way to perform tractable inference on sequence data. They define and learn a single template network that can be unrolled for length of the sequence. This recurrent template enables learning an SPN over data with varying sequence length.

Sum-product-max networks (SPMN) [15] generalize SPNs by introducing two new types of nodes to an SPN: max and utility nodes. Max nodes correspond to decision variables and utility nodes to the reward function, which allow SPMNs to computationally represent a probabilistic decision-making problem. If the SPMN learned from data is valid by satisfying a set of properties, then it correctly encodes a function that computes the maximum expected utility given the partial order between the variables. As such, valid SPMNs potentially represent a shift in paradigm for decision-making models: from being primarily handcrafted to enabling machine learning from decision-making data.

1.2 Contribution

Motivated by these recent generalizations of the SPN, we present a new graphical model Recurrent Sum-Product-Max Networks. We also present a modified structure learning algorithm LearnSPMN for SPMN. Specific contributions are as follows

- We present Recurrent SPMN (RSPMN) which can be seen as a synthesis of a recurrent SPN and an SPMN: it allows extending the decision-making problem across multiple time steps thereby modeling *sequential* decision-making problems for the first time. They extend the twin benefits of an SPN (tractable inference and directly learned from data) to a new class of problems.

- Given decision-making data consisting of finite temporal sequences of values of state and utility variables, and decisions, we present an effective method for learning an RSPMN of any finite length from this data and evaluating it to obtain the maximum expected utility (MEU) and the corresponding policy. We define a *template network*, which is a key component of the learned model whose repeated application makes the temporal generalization possible.

- We prove that unfolding the learned RSPMN produces a valid SPMN, which, in combination with a result from [15] establishes that its evaluation is equivalent to using the sum-max-sum rule.

- On a testbed of sequential decision-making datasets from simulations in perfectly-observed domains, we demonstrate that the learned RSPMNs generate MEUs that are close to the optimal. RSPMNs offer a model-based approach to offline (batch) reinforcement learning where simulation data has already been collected. Consequently, we also compare the MEUs with those from a recent batch-constrained Q-learning method [7] and report favorable results.

- Additionally, we present a modified LEARNSPMN algorithm, illustrate the differences to original LEARNSPMN algorithm and demonstrate that these modifications improve the log-likelihoods by orders of magnitude on a testbed of decision-making datasets.

1.3 Organisation of Thesis

Rest of the thesis is organised as follows.

- In the "Background" chapter, Chapter 2, we introduce several important foundational concepts that are required for comprehending this thesis well. Sections 2.1 and 2.2 introduce several well established notions like network polynomial and validity in the SPN literature. We also discuss structure learning and evaluation of SPNs in these sections. Focus of these sections is to define and clearly state these concepts and provide an intuitive understanding using simple examples.

- In Section 2.3, we introduce SPMNs and the notion of validity of an SPMN. Focus of this Section is to show the original structure learning algorithm for SPMN and explain the *modifications* we made to improve the learning algorithm. We also make the often confusing differences and similarities in notions of validity of SPN and SPMN clear in this section. In [15], an evaluation of SPMN is given. But the evaluation of SPMN to compute the MEU is not discussed in depth. In Section 2.3, we describe in detail the evaluation of SPMN for computing the MEU.

- In the chapter "Recurrent SPMNs", Chapter 3, we present the theoretical framework of

RSPMNs. In this chapter, we present notion of validity of RSPMN. We define template and top networks that constitute RSPMN and define some constraints on them which make the RSPMN valid.

- In the chapter "Learning and Evaluation of RSPMN", Chapter 4, we present a complete structure learning algorithm LearnRSPMN for RSPMN, aided by an illustrative example and pseudo-code. We also describe the evaluation of RSPMN to compute the MEU and how to determine the best decisions from the network based on MEU.

- In chapter "Experiments", Chapter 5, we report the experimental results of modified LearnSPMN algorithm (Section 5.1 and LearnRSPMN(Section 5.2 on various metrics and discuss these results.

- We conclude in Chapter 6 by summarising RSPMNs and providing limitations and possible future directions to build on this research.

Chapter 2

Background

In this chapter, we define and explain several foundational concepts and terms that we build on in later parts of this thesis. We begin by introducing Network Polynomial in Section 2.1. Then, we introduce the notion of validity of an SPN and define some properties that ensure the validity of an SPN in Section 2.2. In Section 2.2, we also provide the structure learning algorithm LearnSPN and evaluation of SPN. We then present SPMN on similar lines to SPN in Section 2.3 while making the distinctions clear between notions of validity between SPN and SPMN. Next, we present a new structure learning algorithm and evaluation for SPMN that was built on top of the original SPMN algorithm and SPMN evaluation. We explain the modifications made by us in Section 2.3 and present results of the modified algorithm in the experiments chapter. In Section 2.4, we introduce some related work describing recurrent SPNs, decision circuits and batch reinforcement learning

2.1 Network Polynomial

A network polynomial presented in [3] is a representation of the probability distribution of a Bayesian network as a polynomial. It is showed in [3] that several kinds of probabilistic inferences can be made immediately from the values and partial derivatives of network polynomial. Formally,

Definition 2.1.1. *A network polynomial is defined as*

$$f = \Sigma_{\mathbf{x}} \Pi_{xu \sim \mathbf{x}} \lambda_x \theta_{x|\mathbf{u}}$$

for a Bayesian network N with variables $\mathbf{X}$ and the parents of variable X being $\mathbf{U}$.

For each network variable X there are a set of evidence variables λ_x and for each network family $\mathbf{XU}$ there are a set of parameters $\theta_{x|\mathbf{u}}$. The outer sum in definition 2.1.1 ranges over all instantiations $\mathbf{x}$ of the network variables. For each instantiation x, the inner product ranges over all in- stantiations of families $\mathbf{xu}$ that are compatible with x. For example, consider the Bayesian network (BN) in Figure 2.1 with conditional probability tables as given in Table 2.1

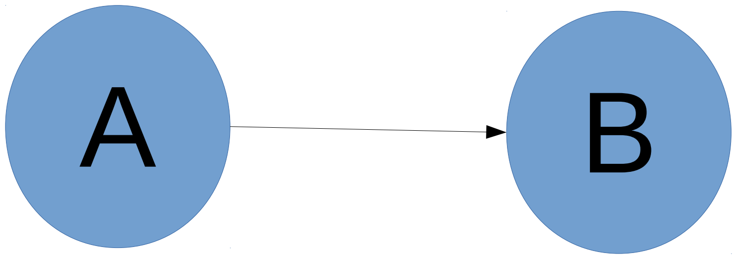

Figure 2.1: Bayesian Network with A as parent of B [3]

A	θ_A
true	$\theta_a = 0.3$
false	$\theta_{\tilde{a}} = 0.7$

A	B	$\theta_{B\|A}$
true	true	$\theta_{b\|a} = 0.1$
true	false	$\theta_{\tilde{b}\|a} = 0.9$
false	true	$\theta_{b\|\tilde{a}} = 0.8$
false	false	$\theta_{\tilde{b}\|\tilde{a}} = 0.2$

Table 2.1: Conditional Probabilities [3]

The network polynomial for the BN given in Figure 2.1 would be

$$f = \lambda_a \lambda_b \theta_a \theta_{b|a} + \lambda_a \lambda_{\tilde{b}} \theta_a \theta_{\tilde{b}|a} + \lambda_{\tilde{a}} \lambda_b \theta_{\tilde{a}} \theta_{b|\tilde{a}} + \lambda_{\tilde{a}} \lambda_{\tilde{b}} \theta_{\tilde{a}} \theta_{\tilde{b}|\tilde{a}} \tag{2.1}$$

Definition 2.1.2. *[3] The value of network polynomial f at evidence $\mathbf{e}$, denoted by $f(\mathbf{e})$, is the result of replacing each evidence indicator λ_x in f with 1 if x is con- sistent with $\mathbf{e}$, and with 0 otherwise.*

In the network polynomial in equation 2.1, if the evidence $\mathbf{e}$ is ab, then $f(ab)$ is computed by assigning $\lambda_a = \lambda_b = 1$ and $\lambda_{\tilde{a}} = \lambda_{\tilde{b}} = 0$. Therefore, $f(ab) = \theta_a\theta_{b|a}$, which is the probability $P(ab)$ of the BN in Figure 2.1.

Theorem 2.1.1. *[3] Let N be a Bayesian network representing probability distribution P and having polynomial f . For any evidence (instantiation of variables) $\mathbf{e}$, we have $f(\mathbf{e}) = P(\mathbf{e})$.*

A large number of probabilistic inference queries can be answered by using the values of network polynomial and its partial derivatives. We present some probabilistic semantics of network polynomial here that are sufficient to comprehend the later parts of this thesis. We encourage the reader to refer [3] for a more detailed analysis.

Theorem 2.1.2. *[3] Let N be a Bayesian network representing probability distribution P and having polynomial f . For every variable $\mathbf{X}$ and evidence $\mathbf{e}$, we have*

$$\frac{\partial f}{\partial \lambda_x}(\mathbf{e}) = P(x, \mathbf{e} - \mathbf{X})$$

Here, $\mathbf{e} - \mathbf{X}$ denotes the instantiation of subset of variables in $\mathbf{e}$ after removing those instantiations that correspond to variables in $\mathbf{X}$. For example, if $\mathbf{e} = a\tilde{b}c\tilde{d}$ and $\mathbf{X} = AD$ then $\mathbf{e} - \mathbf{X} = \tilde{b}c$

To analyse the equation in theorem 2.1.2 for the BN given in Figure 2.1, let us suppose

$\mathbf{e} = \tilde{b}$ and $x = a$ which implies $\mathbf{X} = A$. Therefore,

$$\begin{aligned}
&\lambda_x = \lambda_a \\
&\frac{\partial f}{\partial \lambda_a} = \lambda_b \theta_a \theta_{b|a} + \lambda_a \lambda_{\tilde{b}} \theta_a \theta_{\tilde{b}|a} \\
&\frac{\partial f}{\partial \lambda_a}(\mathbf{e}) = \frac{\partial f}{\partial \lambda_a}(\tilde{b}) = \theta_a \theta_{\tilde{b}|a}
\end{aligned}$$

which is equal to $P(x, \mathbf{e} - \mathbf{X}) = P(a, \tilde{b} - \mathbf{A}) = P(a, \tilde{b})$. This means that if we differentiate a network polynomial f with an indicator value corresponding to a variable X and compute the result at an evidence $\mathbf{e}$, then we get join probability distribution of the instantiation x and the instantiation of remaining variables in $\mathbf{e}$ except instantiations corresponding to $\mathbf{X}$.

While we have seen probabilistic semantics of first order partial derivative of network polynomial f with respect to indicator variables λ_x works, one could also differentiate f partially w.r.t network parameters θ.

Theorem 2.1.3. *[3] Let N be a Bayesian network representing probability distribution P and having polynomial f . For every family* $\mathbf{XU}$ *in the network and for every evidence* $\mathbf{e}$*, we have*

$$\theta_{x|\mathbf{u}} \frac{\partial f}{\partial \theta_{x|\mathbf{u}}}(\mathbf{e}) = P(x, \mathbf{u}, \mathbf{e})$$

The equation from Theorem 2.1.3 can be analysed as done above on the network polynomial f for the BN in Figure 2.1. Since $f(\mathbf{e}) = P(\mathbf{e})$, The sematics of second order partial derivative are showed in [3].

Network polynomial itself can have exponentially large number of terms, but it can be expressed compactly as a graph. Specifically, in [3], it is shown that the network polynomial of exponential size can sometimes be expressed as an arithmetic circuit of linear size. Informally, an arithmetic circuit(**AC**) is a rooted directed acyclic graph with leaf nodes holding numerical values of variables and other nodes being arithmetic $(+, -, *, /)$ operators. Size of the arithmetic circuit is the number of edges it contains. The results from the probabilistic semantics of partial derivatives of f are important because some of these partial derivatives can be obtained in polynomial time using arithmetic circuits. Hence, the probabilistic inference in arithmetic circuits encoding a network polynomial is tractable.

Sum-Product Networks (SPNs) presented in the next section can be considered as a restricted **AC** containing weighted sum and product operators as internal nodes. In SPNs, the probabilistic semantics are defined more directly compared to **ACs** and SPNs can be learned efficiently from data.

2.2 Sum-Product Networks

Sum-Product Networks build on the ideas of network polynomial [3]. The network polynomial is generalised to unnormalised probability distrubutions in [22]. The unnormalised network polynomial can be given as $\Sigma_{\mathbf{x}}\Phi(\mathbf{x})\Pi(\mathbf{x})$, where $\Phi(\mathbf{x}) >= 0$ is an unnormalized probability distribution and $\Pi(\mathbf{x})$ is the product of the indicators that have value 1 in state $\mathbf{x}$. The probability distribution P is related to Φ as $P(\mathbf{x}) = \frac{\Phi(\mathbf{x})}{Z}$, where Z is the partition function. Partition function is a normalising factor which can be obtained from a network polynomial by setting all the indicator variables to 1. For example, consider an unnormalised network polynomial for BN given in Figure 2.1.

$$f_u = \lambda_a\lambda_b\Phi(ab) + \lambda_a\lambda_{\tilde{b}}\Phi(a\tilde{b}) + \lambda_{\tilde{a}}\lambda_b\Phi(\tilde{a}b) + \lambda_{\tilde{a}}\lambda_{\tilde{b}}\Phi(\tilde{a}\tilde{b}) \tag{2.2}$$

The partition function Z would be equal to f_u after setting $\lambda_a = \lambda_b = \lambda_{\tilde{a}} = \lambda_{\tilde{b}} = 1$ The probability distribution $\theta_a\theta_{b|a} = \frac{\Phi(ab)}{Z}$.

As we have seen in Section 2.1, while it is possible to make inference using a network polynomial, it grows exponentially in the number of variables. But a Sum-Product Network may be able to represent and evaluate it in polynomial space and time. An SPN is defined as follows:

Definition 2.2.1. *[22] A sum-product network (SPN) over variables $x_1 \ldots x_d$ is a rooted directed acyclic graph whose leaves are the indicators $x_1 \ldots x_d$ and $x_1 \ldots \overline{x}_d$ and whose internal nodes are sums and products. Each edge (i, j) emanating from a sum node i has a non-negative weight w_{ij}. The value of a product node is the product P of the values of its children. The value of a sum node is $\Sigma_{j \epsilon Ch(i)} w_{ij} v_j$, where $Ch(i)$ are the children of i and v_j is the value of node j. The value of an SPN is the value of its root.*

A sample SPN for the BN shown in Figure 2.1 is shown in Figure 2.2 for illustration purposes[1]. The value of an SPN can be represented as a function of indicator variables as $S(x_1, x_2 \ldots x_n, \tilde{x_1}, \tilde{x_2} \ldots \tilde{x_n})$. When all indicator variables are set to 1, it is represented as

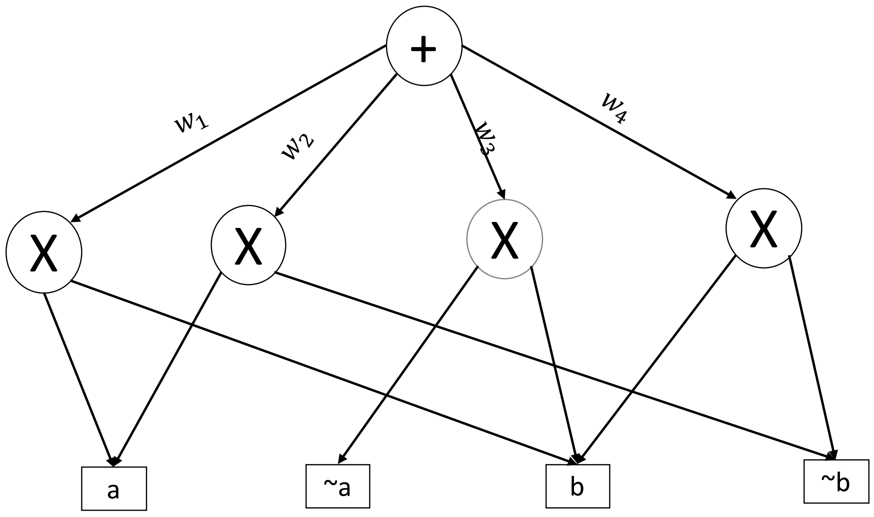

Figure 2.2: Bayesian Network with A as parent of B [3]

$S(*)$ which is the partition function Z.

[1]The SPN in the Figure is one representation for this network polynomial. There could be other SPN structures that represent this network polynomial

Consider the SPN represented in Figure 2.2. This SPN can be represented as

$$S(a, b, \tilde{a}, \tilde{b}) = w_1 ab + w_2 a\tilde{b} + w_3 \tilde{a}b + w_4 \tilde{a}\tilde{b} \tag{2.3}$$

When a partial evidence $\mathbf{e} = a$, $S(\mathbf{e}) = S(1, 1, 0, 1)$. When a complete state i.e., it contains instantiations for all the variables, is $\mathbf{x} = a\tilde{b}$, $S(\mathbf{x}) = S(1, 0, 0, 1)$. $S(\mathbf{x})$ gives the unnormalised probability distribution at a state $\mathbf{x}$, for all $\mathbf{x} \in \mathbf{X}$. Therefore, the unnormalised probability distribution for a partial evidence $\mathbf{e}$ is $\Phi_S(\mathbf{e}) = \Sigma_{\mathbf{x} \sim \mathbf{e}} S(\mathbf{x})$.

Definition 2.2.2. *A sum-product network S is valid iff $S(\mathbf{e}) = \Phi_S(\mathbf{e})$ for all evidence $\mathbf{e}$.*

This means that an SPN is valid if it computes the correct probability of evidence. Since $\Phi_S(\mathbf{e}) = \Sigma_{\mathbf{x} \sim \mathbf{e}} S(\mathbf{x})$, an SPN is valid if $S(\mathbf{e}) = \Sigma_{\mathbf{x} \sim \mathbf{e}} S(\mathbf{x})$ In the example, when $\mathbf{e} = a$, the complete states that are consistent with $\mathbf{e} = a$ are $\mathbf{x_1} = ab$, $\mathbf{x_2} = a\tilde{b}$. Therefore, for this SPN to be valid $S(1, 1, 0, 1)$ must be equal to $S(1, 1, 0, 0) + S(1, 0, 0, 1)$.

The expansion of an SPN is given as a polynomial $\Sigma_k s_k \Pi_k(\dots)$, where $\Pi_k(\dots)$ is a monomial over indicator variables and $s_k >= 0$. Let us consider an SPN over two variables X_1, X_2 with value at root as

$$S(x_1, x_2, \tilde{x_1}, \tilde{x_2}) = (w_1 x_1 + w_2 \tilde{x_1})(w_3 x_2 + w_4 \tilde{x_2}) \tag{2.4}$$

This SPN has a sum node $+_1$ that connects indicator variables $x_1, \tilde{x_1}$ with weights w_1, w_2 and another sum node $+_2$ that connects $x_2, \tilde{x_2}$ with weights w_3, w_4. These two sum nodes are connected to a root product node P_0. The expansion of this SPN would be

$$S(x_1, x_2, \tilde{x_1}, \tilde{x_2}) = w_1 w_3 x_1 x_2 + w_1 w_4 x_1 \tilde{x_2} + w_2 w_3 \tilde{x_1} x_2 + w_2 w_4 \tilde{x_1} \tilde{x_2} \tag{2.5}$$

An SPN is valid if its expansion is its network polynomial which also means that each monomial in the expansion is non-zero in exactly one state and each state has exactly one monomial that is non-zero in it. Hence, each monomial and state has one to one correspondence with each other. We show this by verifying it on the networks represented in Equations 2.3 and 2.4. For a complete proof refer [22]. Let's say former SPN is S_A and the latter is S_X. As we can see from the equations, both networks satisfy the condition that monomials and states have one to one correspondence. Therefore, both these networks must be valid. Let us verify it on a partial evidence $\mathbf{e} = a$ for S_a and $\mathbf{e} = \tilde{x_2}$ for S_x. $S_A(\mathbf{e}) = S_A(1, 1, 0, 1)$ and $S_X(\mathbf{e}) = S_X(1, 0, 1, 1)$. For the SPNs to be valid $S(\mathbf{e}) = \Sigma_{\mathbf{x} \sim \mathbf{e}} S(\mathbf{x})$. Hence, $S_A(1, 1, 0, 1)$ must be equal to $S_A(1, 1, 0, 0) + S_A(1, 0, 0, 1)$ and $S_X(1, 0, 1, 1)$ must be equal to $S_X(1, 0, 0, 1) + S_X(0, 0, 1, 1)$. We can verify from Equations 2.3 and 2.5 that this is indeed the case.

Now that we understand that an SPN would be valid if its expansion is its network polynomial, we would like to know how can we guarantee, if possible, that an expansion of an SPN always leads to its network polynomial. To this extent, constraints on the network of an SPN are discussed in [22], which guarantee that the resultant expansion gives a network polynomial . This makes the SPN valid. We present these properties as follows,

Definition 2.2.3 (scope)**.** *Scope of a node is the union of scopes of its children; scope of a leaf node is the set of random variable that the indicator variable corresponds to.*

Note that the leaf of node of an SPN can also hold a tractable distribution over random variables instead of being an indicator variable. Then, the scope of a leaf node is the set of random variables whose distribution the leaf node holds.

Definition 2.2.4 (Sum-complete)**.** *A sum-product network is complete iff all children of the same sum node have the same scope.*

Definition 2.2.5 (Consistent)**.** *A sum-product network is consistent iff no variable appears negated in one child of a product node and non-negated in another.*

Definition 2.2.6 (Decomposable)**.** *A sum-product network is decomposable iff no variable appears in more than one child of a product node.*

Theorem 2.2.1. *A sum-product network is valid if it is complete and consistent/decomposable.*

When an SPN is complete and consistent/decomposable, the expansion of SPN always leads to a polynomial with each of its monomials having one to one correspondence with the states. This makes the SPN valid. Although completeness and consistency/decomposability are sufficient conditions for the network to be valid, they are not necessary. For example, the SPN represented by $w_1 x_1 x_2 \tilde{x_2} + w_1 x_1$ is valid as $S(\mathbf{e}) = \Phi_S(\mathbf{e})$ for all evidence $\mathbf{e}$. But it is both incomplete and inconsistent. Completeness and consistency are necessary for the stronger condition that each sub-SPN of an SPN is valid.

Evaluation: Let us suppose that we have an SPN over a set of random variables $\mathbf{X}$. Let $scope(n)$ represent the scope of a node n of SPN. S_n gives the sub-SPN rooted at the node n. $\mathbf{x} = val(\mathbf{X})$ is a vector of values assigned to all random variables in $\mathbf{X}$. ϕ_n represents a tractable unnormalised probability distribution held by a leaf node n over $scope(n)$. $\mathbf{x}_{|scope(n)} = val(scope(n))_{scope(n) \sim \mathbf{X}}$.

Let an evidence for variables $\mathbf{E}$ is given where $\mathbf{E} \subseteq \mathbf{X}$ and $\mathbf{e} = val(\mathbf{E})$. To evaluate an SPN, the random variables in leaf nodes are assigned values consistent with the evidence and corresponding probability is computed. Rest of the random variables that are not consistent with evidence are marginalised by setting their probability to 1. The values from leaf nodes are passed up by applying the operators at each node as shown below,

$$\mathcal{S}_n(\mathbf{e}_{|scope(n)}) = \begin{cases} \begin{cases} \phi_n(scope(n) = \mathbf{e}_{|scope(n)}), & \text{if } scope(n) \sim \mathbf{E} \\ 1, & \text{otherwise} \end{cases}, & \text{if n is leaf node} \\ \Pi_{c \in ch(n)} \mathcal{S}_c(\mathbf{e}_{|scope(c)}), & \text{if n is product node} \\ \Sigma_{c \in ch(n)} w_{nc} \mathcal{S}_c(\mathbf{e}_{|scope(c)}), & \text{if n is sum node} \end{cases} \tag{2.6}$$

The value at the root node of SPN, $\mathcal{S}_{root}(\mathbf{e})$ gives the joint probability $P(\mathbf{E} = \mathbf{e})$. When $\mathbf{E} = \mathbf{X}$, it corresponds to complete evidence inference and when $\mathbf{E} \subset \mathbf{X}$, it corresponds to marginal inference. Notice that both these inferences are linear in size of the network. Consequently, conditional probability distributions are tractable as well because,

$$P(\mathbf{Q}|\mathbf{E}) = \frac{P(\mathbf{Q}, \mathbf{E})}{P(\mathbf{E})}$$

Another type of inference that is linear in the size of the network is MPE inference. Formally, MPE is given as

$$\begin{aligned} \mathbf{q}{*} = \arg\max_{\mathbf{q} \in \mathbf{Q}} P(\mathbf{E} = \mathbf{e}, \mathbf{q}) \text{ where,} \\ \mathbf{E}, \mathbf{Q} \subset \mathbf{X}, \\ \mathbf{E} \cap \mathbf{Q} = \emptyset \text{ and,} \\ \mathbf{E} \cup \mathbf{Q} = \mathbf{X} \end{aligned}$$

Given an evidence , it is the most likely value of all the remaining variables. To perform an approximate MPE inference, the SPN is evaluated bottom-up as shown in Equation 2.6. To get the most likely values of remaining variables, a top-down pass is done by,

- passing through all the out-going(child) branches of product node and

- passing through through the out-going(child) branch with maximum likelihood at sum node.

After reaching the leaf node, the most likely value of the random variable, given evidence $\mathbf{e}$, that the leaf node corresponds to is determined as,

$$q_n = \arg\max \phi_n(scope(n))$$

Note that this is different from MAP inference which gives the most likely value of some remaining variables. This means $\mathbf{E} \cup \mathbf{Q}$ need not be equal to $\mathbf{X}$ for MAP inference which is not linear in size of the network in SPNs.

Structure Learning: We have learnt that a valid SPN is able to represent a network polynomial and the learned SPN is useful in making tractable probabilistic inferences. This leads to the next question, how can we create these SPN structures. One way would be to use the domain knowledge and handcraft these SPNs and learn the parameters (weights) using the data. While this is possible, it may not be scalable to large complex domains. Better yet, can we learn these valid SPN structures from data? In [8], a generic structure learning algorithm (LearnSPN) has been presented for learning a valid SPN. We show a pictorial representation of the algorithm in Figure 2.3.

LearnSPN proceeds by recursively splitting the data into groups of independent variables and clusters of instances as shown in the Figure 2.3. LearnSPN takes the dataset D as input and performs an independence test on all the variables V in D. If the independence test is passed, it returns a product node with each out-going edge corresponding to the group of correlated variables which are independent with respect to the rest of the variables. If the independence test fails, it performs clustering on the instances. It returns a sum node with each out-going edge corresponding to the cluster of instances grouped together. The weight on each out-going edge is equal to the $\frac{numberOfInstancesClusteredTogether}{totalNumberOfInstances}$. This process is

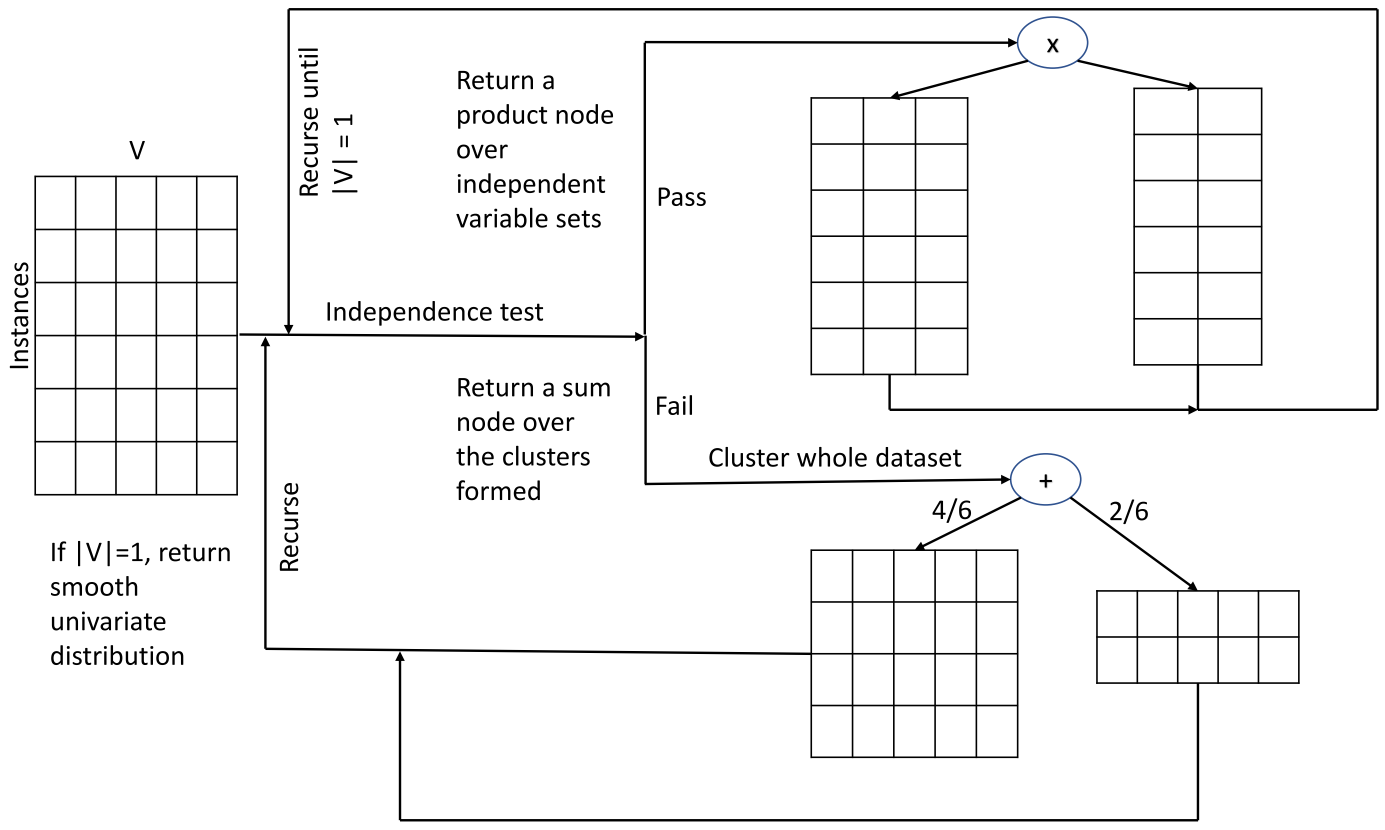
V
Instances
Recurse until |V| = 1
Return a product node over independent variable sets
x
Pass
Independence test
Fail
Return a sum node over the clusters formed
Cluster whole dataset
+
4/6
2/6
Recurse
If |V|=1, return smooth univariate distribution

Figure 2.3: Structure learning algorithm for SPN

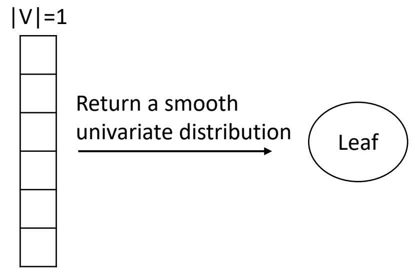

Figure 2.4: Leaf node of SPN

repeated recursively until the number of variables V become 1. At this stage, a smoothed out univariate distribution over V is returned. Additionally, this process can also be stopped after reaching a minimum number of instances for clustering and returning a multivariate distribution over the remaining variables. Several structure learning algorithms for SPNs have been proposed based on this generic LEARNSPN algorithm [19].

2.3 Sum-Product-Max Networks

SPMNs [15] generalise SPNs by introducing two new types of nodes to SPNs. Max nodes that represent decision variables and Utility nodes to represent utility functions. SPMN is defined as follows:

Definition 2.3.1. *[15] An SPMN over decision variables $D_1 \dots D_m$, random variables $X_1 \dots X_n$, and utility functions $U_1 \dots U_k$ 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 $max_{j \epsilon Children(i)} v_j$, 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.*

Recall the concepts of information sets and partial ordering in influence diagrams [12]. Information sets $I_0, \dots, I_m$ 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. An ordering between the information sets may be established as follows: $I_0 \prec D_1 \prec I_1 \prec D_2, \dots, \prec D_m \prec I_m$. This is a partial order, denoted by $P^{\prec}$, because variables within each information set may be observed in any order. Any information set may be empty and variables in I_m need not be observed before some decision node.

In [15], Validity of SPMN is defined in terms of the properties on the nodes of SPMN. It is proved in [15] that these properties are sufficient to ensure that evaluation of the SPMN computes MEU values identical to those arrived at by an application of the Sum-Max-Sum rule over the same evidence and the partial order of the variables We present these properties and definitions below [15]:

Definition 2.3.2 (Sum-complete)**.** *An SPN is complete iff all children of the same sum node have the same scope.*

Definition 2.3.3 (Decomposable)**.** *An SPN is decomposable iff no variable appears in more than one child of a product node.*

Definition 2.3.4. *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 2.3.5. *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.*

Definition 2.3.6. *An SPMN is valid if it is sum-complete, decomposable, max-complete, and max-unique.*

Theorem 2.3.1. *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:* $S(e) = MEU(e|P, U)$.

Notice that the notion of validity in an SPN is tied to correctly computing the probability distribution, while it corresponds to computing correct MEU in accordance with the sum-max-sum rule in an SPMN.

Evaluation: To evaluate an SPMN, values are assigned to the random variables that are consistent with the evidence. 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 node.

2.3.1 Structure Learning: LearnSPMN

A learning algorithm (LearnSPMN) is presented in [15] to learn a valid SPMN. This algorithm works by splitting the data recursively using decision values on decision variables and clustering and independence testing on random variables and utility variable based on the partial order of the variables.

LearnSPMN takes a data set and partial order $P^{\prec}$ as input. It iterates through the partial order $P^{\prec}$. If the current item in the partial order $P^{\prec}[i]$ is a decision variable D, a corresponding max node is returned by splitting the data over each decision. It then iterates to the next item in $P^{\prec}$ and recursively calls LearnSPMN on the split data as shown in Figure 2.5.

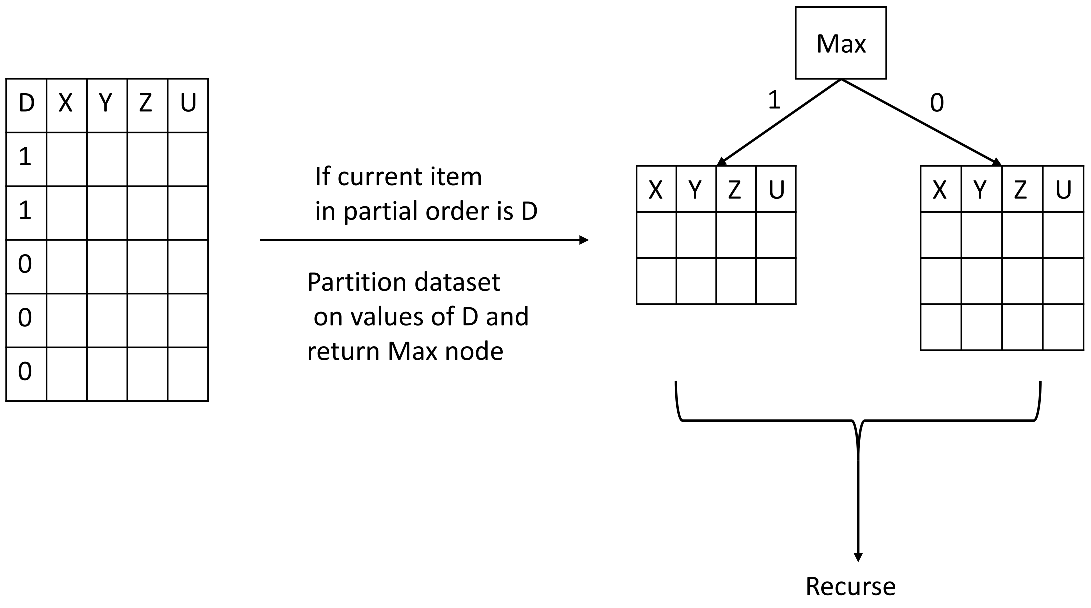
D X Y Z U
1
1
0
0
0
If current item in partial order is D
Partition dataset on values of D and return Max node
Max
1
0
X Y Z U
X Y Z U
Recurse

Figure 2.5: LearnSPMN if the current item in partial order $P^{\prec}$ is a decision variable D. Next step is shown in Figure 2.6

If the current item in the partial order $P^{\prec}[i]$ is an information set $\mathbf{I}$, then an independence test is performed on variables in the current information set $\mathbf{I}$. This test tries to partition variables $\mathbf{V_I}$ in $\mathbf{I}$ into independent subsets while keeping the rest of the variables $\mathbf{V_R} = \{P^{\prec}[i] \cup P^{\prec}[i+1] \cup \ldots]\}$ as one set. If a partition is found in V_I, a product node is returned.

One of the out-going edge corresponds to set of variables in rest of the partial order $\mathbf{V_R}$ and other out-going edges correspond to the sets of variables in $\mathbf{V_I}$ as shown in Figure 2.6

If a partition is not found during independence test, clustering is performed over instances. Unlike independence testing, clustering takes all variables in the data set into account. Similar to clustering in LearnSPN, a sum node is returned with corresponding weights after clustering the instances as shown in Figure 2.6.

This process is recursively repeated until the number of variables $|V| = 1$. If $|V| = 1$, a smoothed out univariate distribution is returned. If V is a utility variable, the smoothed out univaraiate distribution over utility values is returned.

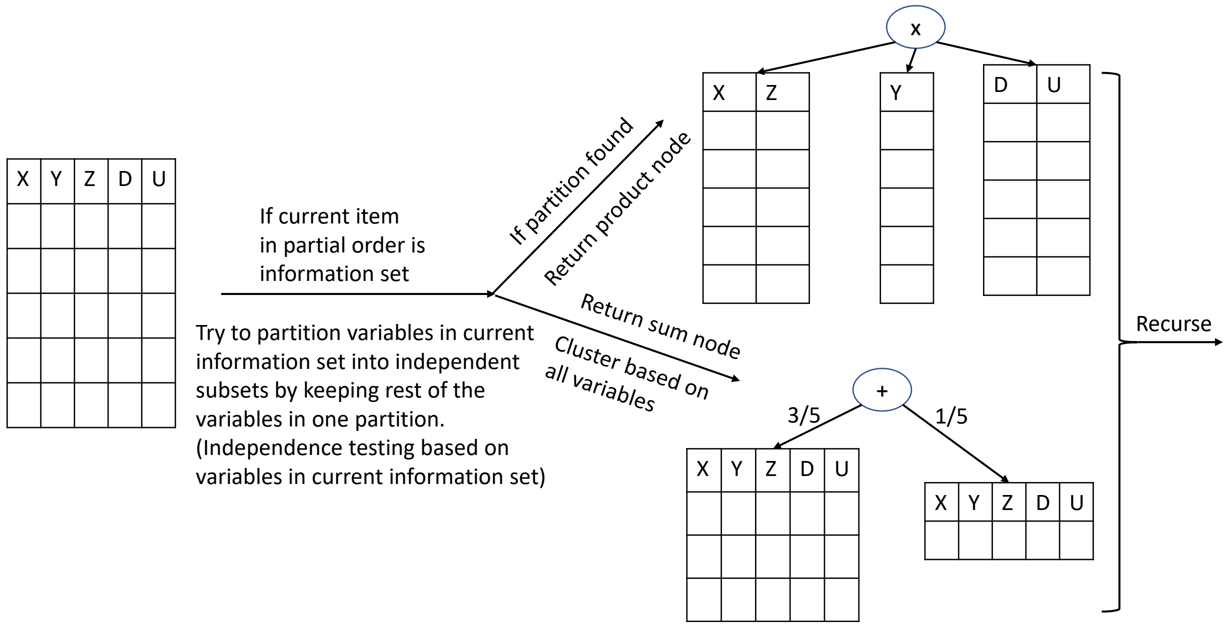
X Y Z D U
If current item in partial order is information set
Try to partition variables in current information set into independent subsets by keeping rest of the variables in one partition. (Independence testing based on variables in current information set)
If partition found
Return product node
Return sum node
Cluster based on all variables
x
X Z
Y
D U
+
3/5
1/5
X Y Z D U
X Y Z D U
Recurse

Figure 2.6: LearnSPMN if the current item in partial order $P^{\prec}$ is an information set $\mathbf{I} = \{X, Y, Z\}$, $\mathbf{V_R} = \{D, U\}$.

2.3.2 Modified LearnSPMN algorithm

While the original LearnSPMN creates valid SPMNs, it fails to acknowledge correlations between the variables in current information set $\mathbf{V_I}$ and the variables in the rest of the

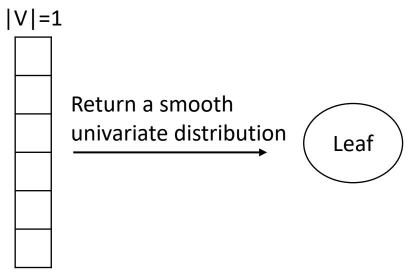

Figure 2.7: Leaf node of SPMN. If V is a utility variable, the smoothed out univaraiate distribution over utility values is returned.

partial order $\mathbf{V_R}$ while making the split on independence testing. An illustration of the same is shown in Figure 2.8.

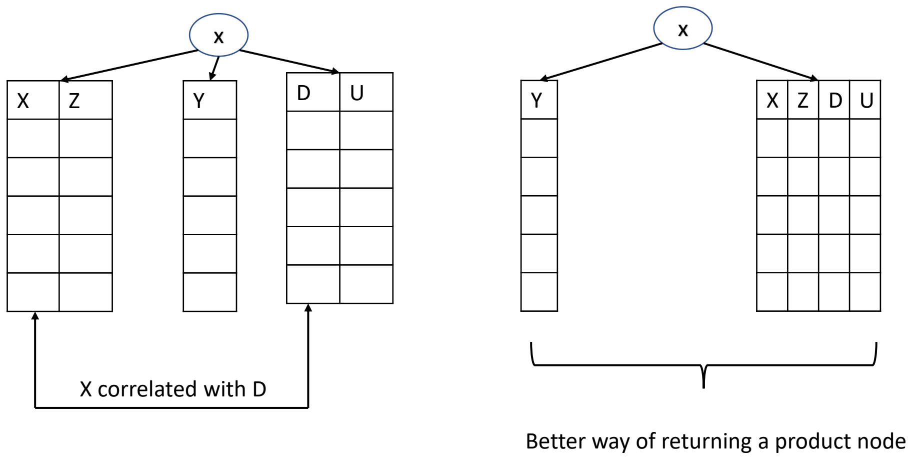

Figure 2.8: Left figure shows the product node returned when $XZ \perp\!\!\!\perp Y$ using original LearnSPMN even though X is correlated with D. Right figure shows a better way of returning a product node in this scenario

Since the independence test is done among the variables $\mathbf{V_I}$ within the current information set $\mathbf{I}$, the correlations between variables in $\mathbf{V_I}$ with those in $\mathbf{V_R}$ is missed. Therefore, we propose a change to the independence test in the original LearnSPMN algorithm.

In the new approach, we test the independence between all the variables $\mathbf{V}$ in the data set. This returns a set of independent subsets of variables $\mathbf{S}$. If $\mathbf{V_R} \cap \mathbf{S_j} = \emptyset$ then $\mathbf{S_j}$ is a set of correlated variables independent of all the other variables in $\mathbf{V}$. This corresponds to variable Y in Figure 2.8.

All the subsets such that $\mathbf{S_j} \cap \mathbf{V_R}$ is not $\emptyset$ are identified. This means that these subsets have some correlation with variables in $\mathbf{V_R}$. A union of all such subsets $\mathbf{V_{rem}}$ is formed while respecting the partial order $P^{\prec}$. This corresponds to the set of variables $\{X, Z, D, U\}$ in Figure 2.8. A product node is returned with one of the out-going edge corresponding to $\mathbf{V_{rem}}$ and other out-going edges corresponding to the independent subsets $\mathbf{S_j}$ such that $\mathbf{V_R} \cap \mathbf{S_j} = \emptyset$. This difference in independence testing between the original LearnSPMN and the modified LearnSPMN is shown in the Figure 2.9.

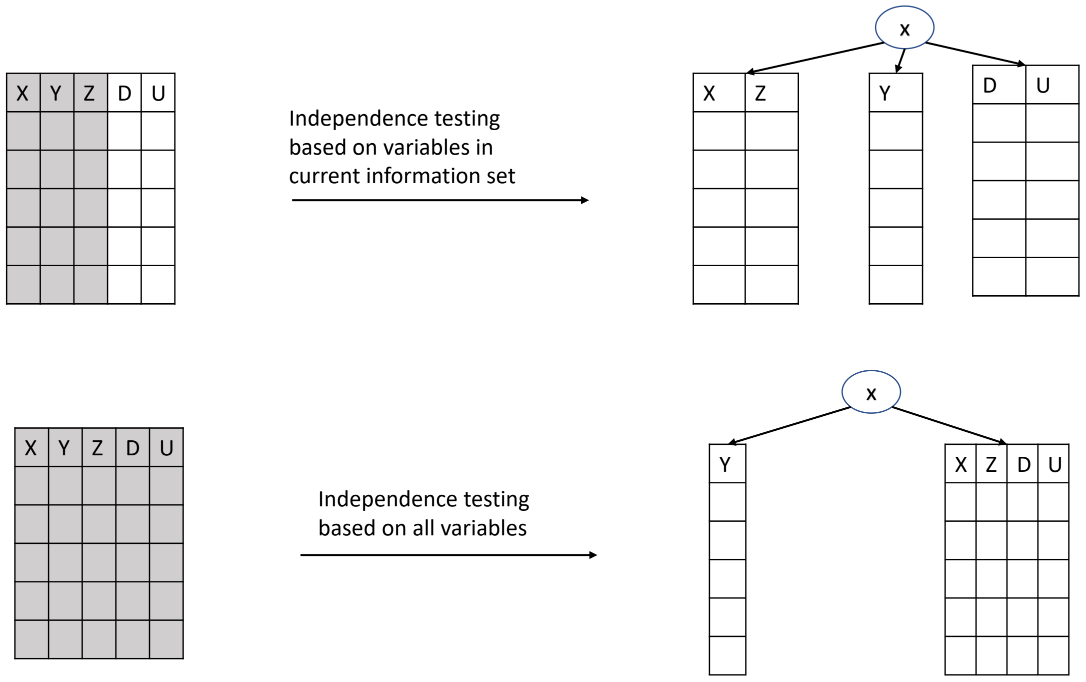
X Y Z D U
Independence testing based on variables in current information set
x
X Z
Y
D U
X Y Z D U
Independence testing based on all variables
x
Y
X Z D U

Figure 2.9: Top figure shows the indpendence testing in original LearnSPMN while the bottom figure shows it on modified LearnSPMN

We show in the experimental section that this change to independence testing improves the log-likelihood on various data sets by several orders of magnitude. We also observed that the structures were more comprehensible with the change. Clustering of instances in the original LearnSPMN is done by taking all the variables in the data set into account. While this makes sense, we found that changing the clustering by confining the variables to current information set **I** improved the performance of the algorithm. We found from experiments that changing this not only improved the log-likelihoods, but also created sensible structures when compared to clustering by taking all variables **V** into account. A possible intuitive explanation for this is that the variables $\mathbf{V_I}$ in the current information set **I** in the partial order $P^{\prec}$ are observed before any variables $\mathbf{V_R}$ of next items in the partial order. Due to this, clustering over all variables **V** might violate this partial order. Nonetheless, it is interesting to note that, considering correlations between variables in $\mathbf{V_I}$ and $\mathbf{V_R}$ works better for independence testing.

To summarise, LearnSPMN is modified to do independence testing on all variables and clustering within the partial order. This is essentially reversing the clustering and independence testing on the variables and leaving the rest of the algorithm in place. The pseudo-code for the complete modified LearnSPMN is shown in Algorithm 1. From here on, when we refer LearnSPMN, we refer to the modified LearnSPMN algorithm.

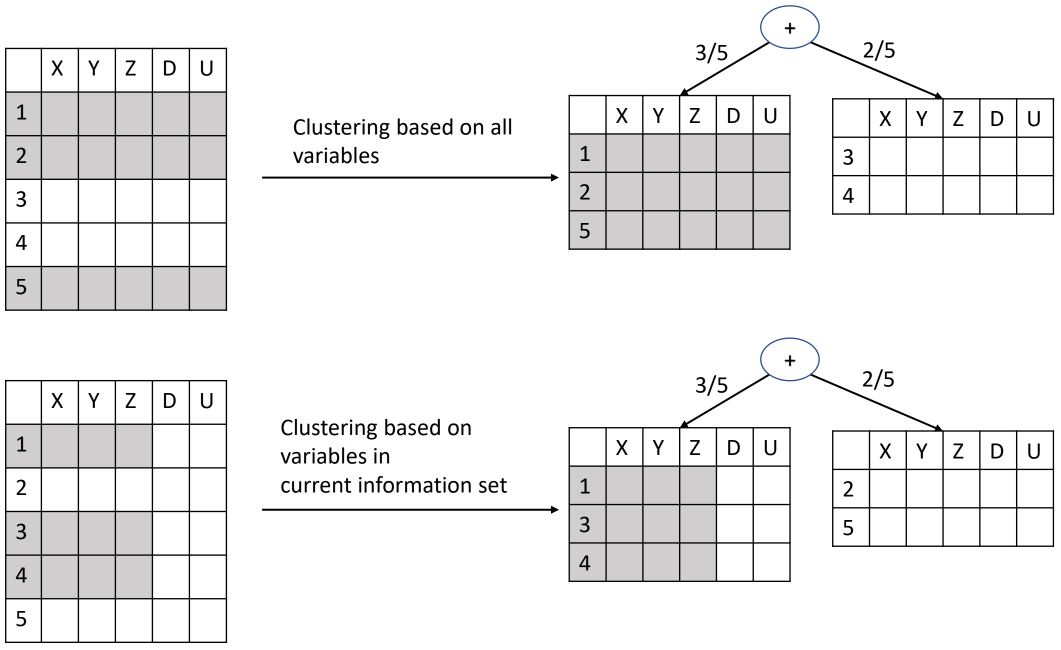
X Y Z D U
1
2
3
4
5
Clustering based on all variables
+
3/5
2/5
Clustering based on variables in current information set

Figure 2.10: Top figure shows the clustering in original LearnSPMN while the bottom figure shows it on modified LearnSPMN.

Algorithm 1: LearnSPMN

```
input: $D$: instances, $V$: set of variables, $i$: infoset index, $P^{\prec}$: partial order
output: SPMN
1  if $|\mathbf{V}| = 1$ then
2      return smoothed univariate distribution over $V$
3  else
4      if $\mathbf{V} \cap P[i] = \emptyset$ then
5          $i \leftarrow i + 1$
6  $\mathbf{V_R} \leftarrow$ P[i + 1] $\cup$ P[i + 2] $\cup$ P[i + 3] $\cup \ldots$
7  if $P[i]$ is a decision variable then
8      for $v \in$ decision values of $P[i]$ do
9          $D^v \leftarrow$ subset of $D$ where $P[i] = v$
10     return $\mathrm{MAX}_v$ LearnSPMN$(D, \mathbf{V_R}, i + 1, P^{\prec})$
11 else
12     Try to partition $\mathbf{V}$ into a set of independent subsets $\mathbf{S}$
13     $\mathbf{V_{rem}} \leftarrow \emptyset$
14     for each subset $\mathbf{S_j} \in \mathbf{S}$ do
15         if $\mathbf{S_j} \cap \mathbf{V_R}$ is not $\emptyset$ then
16             $\mathbf{V_{rem}} \leftarrow \mathbf{S_j} \cup \mathbf{V_R}$
17             remove $\mathbf{S_j}$ from $\mathbf{S}$
18     $\mathbf{V_{rem}} \cup \mathbf{S}$
19     if length of $\mathbf{S} > 1$ then
20         return $\Pi_j$LearnSPMN$(D^j, \mathbf{S_j}, i, P^{\prec})$
21     else
22         partition $D$ into clusters $D^j$ of similar
             instances based on the values in $P[i]$
23         return $\Sigma_j \frac{|D_j|}{|D|} \times$ LearnSPMN$(D^j, \mathbf{V}, i, P^{\prec})$
```

Evaluation: In [15], a brief evaluation for SPMN is presented. Although, it is sufficient for evaluating the likelihood, it does not clearly show the evaluation of the SPMN for computing the MEU. Here, we present a detailed explanation on evaluating the SPMN to compute the MEU.

To compute the MEU, in a bottom-up pass, we propagate the likelihoods and utility values from the leaf nodes. In addition to evaluation shown in Equation 2.6, in SPMN, evaluation is done at a max node by taking the likelihood from the max child branch. Let a max node have a finite set of decision values $K = \{d_1, d_2, \dots d_n\}$, where $1, 2, \dots n$ belong to each child branch of max node. If a decision value $d_g \in K$ is given for a decision variable D_k corresponding to a max node, the likelihood of the child branch that corresponds to d_g is taken. Let an indicator function to indicate d_g is given by,

$$\mathbf{1}(d) = \begin{cases} 1, & \text{if } d = d_g \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

Evaluation of an SPMN at each node is shown below,

$$\mathcal{S}_n(\mathbf{e}_{|scope(n)}) = \begin{cases} \begin{cases} \phi_n(scope(n) = \mathbf{e}_{|scope(n)}), & \text{if } scope(n) \sim \mathbf{E} \\ 1, & \text{otherwise} \end{cases}, & \text{if n is a leaf node} \\ \Pi_{c \in ch(n)} \mathcal{S}_c(\mathbf{e}_{|scope(c)}), & \text{if n is a product node} \\ \Sigma_{c \in ch(n)} w_{nc} \mathcal{S}_c(\mathbf{e}_{|scope(c)}), & \text{if n is a sum node} \\ \begin{cases} \Sigma_{c \in ch(n)} \mathbf{1}(d_c) \mathcal{S}_c(\mathbf{e}_{|scope(c)}), & \text{if } d_g \text{ is given} \\ \max_{c \in ch(n)} \mathcal{S}_c(\mathbf{e}_{|scope(c)}), & \text{otherwise} \end{cases}, & \text{if n is a decision node} \end{cases} \tag{2.8}$$

To compute the MEU, the likelihoods along with utility values from leaf utility nodes are used. At a utility leaf node the utility value is computed as the expected value $E_{\phi_n}(U)$ based on the distribution the leaf utility node holds. At a product node the utility values are added. At a sum node, the utility values coming from the child nodes are multiplied by normalised likelihood and are added. At a max node, the maximum utility value coming from the child branches is taken or the utility value of child branch g is taken if d_g is given.

This is formally shown below,

$$MEU_n(\mathbf{e}_{|scope(n)}) = \begin{cases} E_{\phi_n}(scope(n)), & \text{if n is a leaf utility node and if not } \mathbf{e}_{|scope(n)} \sim \mathbf{e} \\ \mathbf{e}_{|scope(n)}, & \text{if n is a leaf utility node and if } \mathbf{e}_{|scope(n)} \sim \mathbf{e} \\ 0, & \text{if n is a leaf random variable node} \end{cases} \tag{2.9}$$

$$MEU_n(\mathbf{e}_{|scope(n)}) = \begin{cases} Equation\ 2.9, & \text{if n is a leaf node} \\ \Sigma_{c\in ch(n)} MEU_c(\mathbf{e}_{|scope(c)}), & \text{if n is a product node} \\ \Sigma_{c\in ch(n)} w_{nc} \times \frac{\mathcal{S}_c(\mathbf{e}_{|scope(c)})}{\mathcal{S}_n(\mathbf{e}_{|scope(n)})} \times MEU_c(\mathbf{e}_{|scope(c)}), & \text{if n is a sum node} \\ \begin{cases} \Sigma_{c\in ch(n)} \mathbf{1}(d_c) MEU_c(\mathbf{e}_{|scope(c)}), & \text{if } d_g \text{ is given} \\ \max_{c\in ch(n)} MEU_c(\mathbf{e}_{|scope(c)}), & \text{otherwise} \end{cases}, & \text{if n is a decision node} \end{cases} \tag{2.10}$$

In general, to compute the MEU, all leaf nodes are marginalised by setting their likelihoods to 1 and the MEU is computed as shown in Equation 2.10. The MEU value at the root MEU_{root} gives the maximum expected utility.

To obtain best decision at any decision node D_i, given some observations of variables in information sets preceding D_i, the leaf random variable nodes are assigned values consistent with the observation corresponding likelihoods and MEUs are computed as shown in Equation 2.8 and 2.10. To get the best decision, a top down pass is done from the root node by

- passing through all the out-going(child) branches of product node and
- choosing the out-going(child) branch with maximum likelihood at sum node

- choosing the child branch g if d_g is given or choosing the branch with maximum likelihood otherwise.

The best decision for a decision variable D_i is the decision chosen at its corresponding max node.

2.4 Related work

Dynamic SPNs a.k.a recurrent SPNs that generalize SPNs to sequence data is presented in [16]. A template network is defined that can be repeated as many times as needed. A valid SPN is produced by capping the repeated templates with a top and a bottom network. This enables learning an SPN and performing probabilistic inference over data with varying sequence lengths. An invariance property for the template if met yields recurrent SPNs that are valid. More recently, an online structure learning algorithm has been presented [11] for these SPNs. RSPMNs can be viewed as a synergistic integration of some of the temporal concepts of recurrent SPNs with SPMNs; thereby generalizing the model to decision making. Furthermore, the handcrafted template structure in recurrent SPNs is mostly fixed (with just the number of interface nodes allowed to change) while LearnRSPMN generates the entire template from data.

SPNs are related to other graphical models for probabilistic inference such as arithmetic circuits[20] and AND/OR graphs[4]. Bhattacharjya and Shachter[1] proposed decision circuits as a representation that ensures exact evaluation and solution of influence diagrams in time linear in the size of the network. A decision circuit extends an arithmetic circuit with max nodes for optimized decision making, which is analogous to how SPMNs extend SPNs. However, decision circuits are obtained by compiling IDs. Previous work has shown that SPMNs are efficiently reducible to decision circuits in time that is linear in the size of the SPMN[14]. However, no dynamic extension of decision circuits has been presented nor any algorithms to learn decision circuits directly from data to the best of our knowledge.

Batch Reinforcement learning is a sub-field of reinforcement learning that deals with data-driven approach to RL. In contrast to online reinforcement learning, batch (also labeled as offline) reinforcement learning seeks to derive an optimal policy from a given set of prior experiences. This set is analogous to our simulations, and may either be fixed or allowed to grow. Batch RL is particularly useful when it is risky, expensive or not feasible to interact with the environment to learn an optimal policy. This also becomes important in several real world scenarios where some sub-optimal policy already exists and we would like to improve on this by using the experience collected from this sub-optimal policy.

While offline reinforcement learning is not as well studied as its online counterpart, the general approach is to modify online techniques for use in batch contexts. Prominent methods, such as experience replay[13] and fitted Q-iteration [5], are model-free and utilize the Q-update rule synchronously over all data until convergence. Recently, methods based on deep neural networks such as BCQ [7] have appeared. In contrast, RSPMNs offer a *model-based approach* to learning the policy from data, which is provably tractable in the size of the network. Additionally, we established RSPMN's favorable performance in comparison to BCQ.

Chapter 3

Recurrent SPMNs

In this chapter, we introduce the theoretical framework of RSPMN. In Section 3.1, we begin by introducing RSPMN and its advantages over SPMN. In Section 3.2, we provide the data schema required by the RSPMN algorithm. In Section 3.3, we formally define the template and top networks and provide some properties that ensure the validity of RSPMN. In Section 3.4, we present a theorem for the validity of an RSPMN and provide an inductive proof for it.

3.1 RSPMN

Popular frameworks such as a Markov decision process (MDP) and languages such as dynamic influence diagrams [25] model long-term decision making as a temporal sequence of decision-making steps.A dynamic influence diagram unfolds an influence diagram with temporal links as many times as the number of steps in the extended problem thereby generating a much larger influence diagram that models the complete sequence. Likewise, a dynamic part of SPMN may be modeled using each of the time steps which can be unrolled to form a larger SPMN that models the entire sequence.

We take this perspective to modeling sequential decision making and introduce a *recurrent SPMN* (RSPMN), which unfolds a *template network* as many times as the number of time steps in each sequence of data. While the template network is not rooted at a single node and is not a valid SPMN, we obtain these by learning an additional component: a top network that caps the unfolded templates, which, in conjunction with some properties on the structure of the template then yields a valid SPMN.

An alternative approach to the recurrent SPMN is to directly learn the SPMN from the sequence data using the LearnSPMN algorithm [14]. However, this poses two main challenges. First, an increase in the sequence length often leads to an exponential blow up of the size of the network and subsequently in evaluation time as we demonstrate later in our experiments. Second, the LearnSPMN algorithm requires a fixed number of variables in each data record. Hence, it may not be used when the sequence length varies between records as there may not always be an efficient way to either fill in the missing time steps for shorter sequences or eliminate extra sequences from the longer ones. One could break the data sequences into fixed-length segments corresponding to each time step. An SPMN can be learned from this data set of segments. However, it is not clear how the resulting SPMNs can be linked to each other to obtain a valid SPMN that accurately models the sequential data.

We begin by describing which domain attributes should be present in the data to allow learning RSPMNs followed by formal definitions of the components of the RSPMN.

3.2 Data Schema

Useful data for learning RSPMNs consists of a finite temporal sequence of values of state and utility variables, and decisions that are actions. More formally, consider a decision-making problem where the (fully observed) state of the environment is characterized by

n variables, X_1, X_2, …, X_n; decisions by a combination of m decision variables, D_1, D_2, …, D_m; and a single utility variable U. The partial order of the variables given by $P^{\prec}$ is $I_0 \prec D_0 \prec I_1 \prec D_1 \cdots \prec I_p$, where $I_0, I_1 \ldots I_p$ are information sets containing subsets of random variables and $D_0, D_1 \ldots D_p$ are decision variables. When the variables follow some $P^{\prec}$, the random variables in the information set I_{i-1} are observed before the decision associated with variable D_i, $1 <= i <= m$, is made.

A candidate data record of at most T steps is then a sequence of T tuples of the form $\langle(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^0$, $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^1$, …, $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^{T-1})\rangle$. Recall from Section 2.3, $I_0, I_1, \ldots, I_m$ are information sets where I_{i-1}, $1 \leq i \leq m$ consists of values of the state 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 state variables and decisions in that tuple.

3.3 Definitions

A RSPMN models sequences of decision-making data of varying lengths using a fixed set of parameters by unfolding a template network. In the context of a dynamic influence diagram, our template corresponds to an influence diagram with temporal links between nodes that are repeated in each time slice.

Definition 3.3.1 (Template network)**.** *A template network is a directed acyclic graph with r root nodes and at least $n+1$ leaf nodes where n is the number of state variables and there is one utility function. The root nodes form a set of interface nodes Ir. The leaf nodes in the network hold the distributions over the random state variables $X_1, X_2, \ldots, X_n$, hold constant values as utility nodes, or are latent interface nodes. The root interface nodes and interior nodes can either be sum, product, or max nodes. Let L denote the set of leaf latent interface nodes. Each latent node in L is related to a root interface node in Ir of the template network* *through a bijective mapping f such that $f : L \rightarrow Ir$.*

The bijective mappings can be seen as time delay edges that link latent interface nodes at time step t with root interface nodes at $t+1$, thereby enabling recurrence of the template. The scope of a latent node of a template network in time step t is related to the scope of a root interface node of the template network at time step $t+1$. More formally, for any pair of latent nodes $l_i^t, l_j^t \in L$, let $f(l_i^t) = ir_i^{t+1}, f(l_j^t) = ir_j^{t+1}$, where $ir_i^{t+1}, ir_j^{t+1} \in Ir$, then

$$scope(l_i^t) = scope(ir_i^{t+1}) \tag{3.1}$$

Intuitively, the leaf latent interface nodes can be viewed as summarizing the latent information coming from the subsequent template network. They pass information between templates of different steps. In other words, they pass up the information in a bottom-up evaluation of the RSPMN and pass down the information in a top-down pass. As such, the root and leaf latent nodes play a key role in linking the template networks during unfolding.

Toward ensuring that the unfolded network is a valid SPMN with a single root, we define another special network as given below.

Definition 3.3.2 (Top network)**.** *A top network is a rooted directed acyclic graph consisting of sum and product nodes, and whose leaves are the latent interface nodes. Edges from a sum node are weighted as in a SPN.*

A generic top and template networks are shown in Figures 3.2 and 3.1

Furthermore, the bottom-most template network – corresponding to the final time step T of the sequential decision making – has its leaf interface nodes removed. Parents of these interface nodes that are sum or product nodes with no other children are also pruned. We may also achieve this while evaluation by setting the values of all these interface nodes as 1 (thus summing them out) and any utility values to pass set to 0.

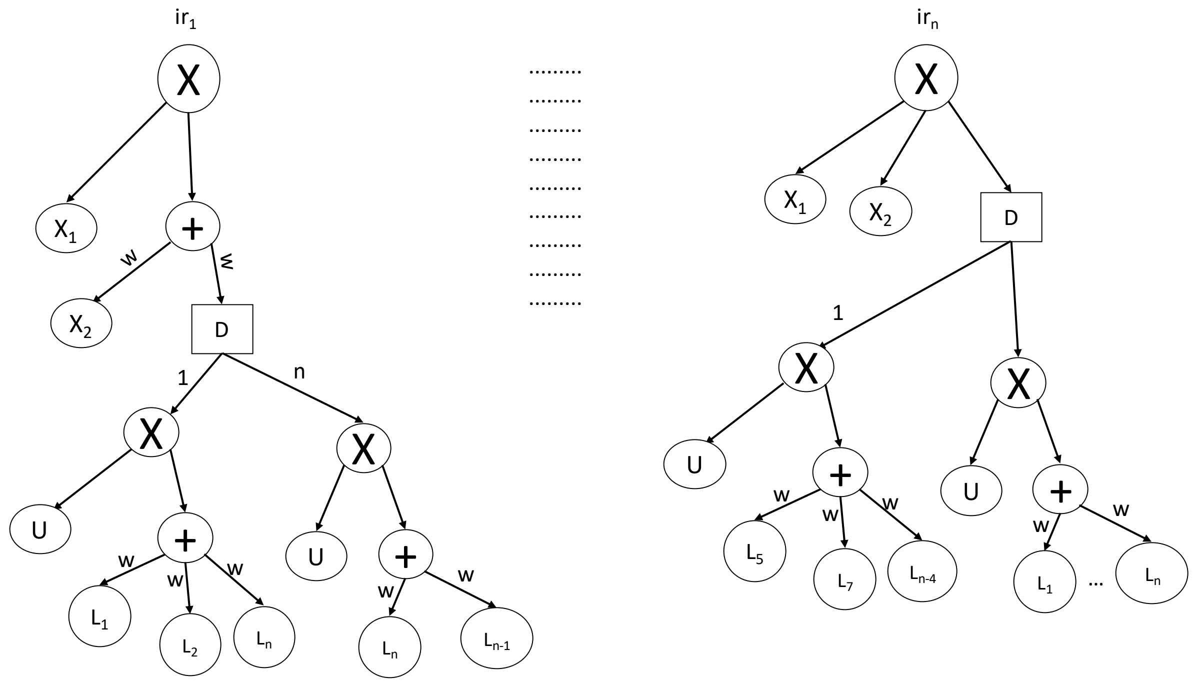

Figure 3.1: Generic template network with n root nodes

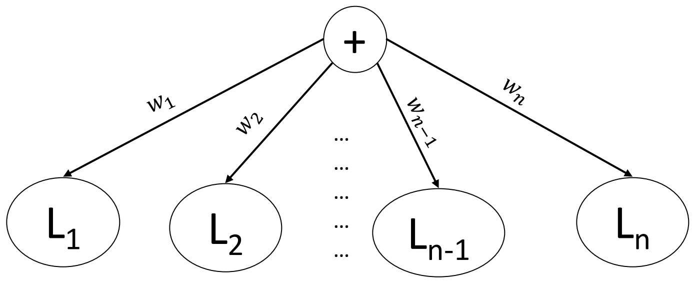

Figure 3.2: Generic top network with n leaf latent nodes

Next, we seek to ensure that the SPMN formed after interfacing the top network and repeated templates is valid. One way to check for validity is to ensure that all the sum nodes in the unfolded SPMN are complete, the product nodes are decomposable, and max nodes are complete and unique as in defined in Section 2.3. However, can we define constraints on the top and template networks that will ensure validity of the unfolded SPMN? If so, we may establish the validity without checking the full network, which may grow to be quite large. To establish this, we first introduce a *soundness* property for the template network.

Definition 3.3.3 (Soundness of the template)**.** *A template network is sound iff*

- *All sum nodes in the template are sum-complete as defined in Def. 2.3.2;*
- *All product nodes in the template are decomposable as defined in Def. 2.3.3;*
- *All max nodes in the template are max-unique and max-complete as defined in Defs. 2.3.4 and 2.3.5;*
- *The scope of all the root interface nodes in Ir is the same, i.e.,*

$$scope(ir_i) = scope(ir_j) \quad \forall\ ir_i, ir_j \in Ir;$$

- *And, the scopes of the leaf latent interface nodes in L are related to scope of the mapped root interface nodes in Ir as*

$$(scope(ir_i) = scope(ir_j)) \Rightarrow (scope(l_i) = scope(l_j)) \tag{3.2}$$

Next, Theorem 3.4.1 establishes that a sound template network combined with a valid top network generates a valid SPMN on unfolding the RSPMN.

3.4 Validity

Theorem 3.4.1 (Validty of RSPMN)**.** *If (a) in the top network, all sum nodes are complete and product nodes are decomposable, i.e., top network is valid, and (b) the template network is sound as defined in Def. 3.3.3, then the SPMN formed by interfacing the top network and the template network unfolded an arbitrary number of times as needed is valid.*

Proof. We sketch a proof by induction. By assumption,

- In the top network, all sum nodes are complete and product nodes are decomposable i.e. top network is valid
- The template network is sound, i.e., all sum nodes are complete, product nodes are decomposable, max nodes are complete and unique.

Base case: We prove that the SPMN formed by interfacing a top network and a single template network is valid. The relation between scopes of leaf latent interface nodes in the top network with the root interface nodes Ir of template network can be inferred from bijective mapping f as,

$$\begin{gathered} scope(ir_i) = scope(ir_j) \Rightarrow scope(l_i^{top}) = scope(l_j^{top}), \qquad (3.3) \\ (l_i^{top}, l_j^{top}) \in L, (ir_i, ir_j) \in Ir, f(l_i) \rightarrow ir_i, f(l_j) \rightarrow ir_j \end{gathered}$$

Since template network is sound,

$$scope(ir_i) = scope(ir_j), \forall (ir_i, ir_j) \in Ir \qquad (3.4)$$

From 3.3 and 3.4, the scopes of leaf latent interface nodes of top network become,

$$scope(l_i^{top}) = scope(l_j^{top}), \forall (l_i, l_j) \in L \qquad (3.5)$$

This means that all the leaf latent interface nodes in top network have same scope. Under this condition and assumption, all the sum nodes of the top network are complete and product nodes are decomposable.

Next, the template network is sound. This means the scopes of all leaf latent interface nodes of template network are same because,

$$scope(ir_i) = scope(ir_j), \forall (ir_i, ir_j) \in Ir \tag{3.6}$$

From bijective mapping $f(L) \rightarrow Ir$, we can infer

$$\begin{aligned} &scope(ir_i) = scope(ir_j) \Rightarrow scope(l_i) = scope(l_j) \\ &(ir_i, ir_j) \in Ir, (l_i, l_j) \in L \end{aligned} \tag{3.7}$$

From 3.6 and 3.7, the scopes of leaf latent interface nodes of template network become,

$$scope(l_i) = scope(l_j), \forall (l_i, l_j) \in L \tag{3.8}$$

Under this condition and soundness of template, all sum nodes of template are complete, product nodes are decomposable and max nodes are complete and unique.

Now, when the top network is interfaced with a single template network, the scopes of leaf latent interface nodes of top network change based on relation with root interface nodes Ir at $t = 0$ as below,

$$\begin{aligned} &scope(ir_i^0) = scope(ir_j^0) \Rightarrow scope(l_i^{top}) = scope(l_j^{top}), \\ &(l_i^{top}, l_j^{top}) \in L, (ir_i^{t+1}, ir_j^{t+1}) \in Ir, f(L_i) \rightarrow ir_i, f(L_j) \rightarrow ir_j \end{aligned} \tag{3.9}$$

From 3.6 we have,

$$scope(ir_i^0) = scope(ir_j^0), \forall (ir_i^0, ir_j^0) \in Ir \tag{3.10}$$

From 3.9 and 3.10,

$$scope(l_i^{top}) = scope(l_j^{top}), \forall (l_i^{top}, l_j^{top}) \in L \tag{3.11}$$

The condition from 3.11 is equivalent to the condition from 3.5. This means the scopes of leaf latent interface nodes of top network have not changed after interfacing with the template network. So, all the sum nodes of top network are complete and product nodes are decomposable even after interfacing with template network. Since no scope is changed in template network after interfacing, all sum nodes are complete, product nodes are decomposable and max nodes are complete and unique in the template network. Therefore the SPMN formed after interfacing top network with single template network is valid.

Induction hypothesis: Let us assume that the SPMN formed after interfacing a top network and the template repeated t times is valid, i.e., all sum nodes are complete, product nodes are decomposable and max nodes are complete and unique. Let this SPMN be R

Inductive step: We now prove that an SPMN formed by interfacing one more template network (template network repeated $(t+1)$ times in total) with R is a valid SPMN.

Since the template network is sound, as we have shown in 3.6, 3.7 and 3.8, we can show that for template at $t+1$,

$$scope(ir_i^{t+1}) = scope(ir_j^{t+1}), \forall (ir_i^{t+1}, ir_j^{t+1}) \in Ir \tag{3.12}$$

$$scope(l_i^{t+1}) = scope(l_j^{t+1}), \forall (l_i^{t+1}, l_j^{t+1}) \in L \tag{3.13}$$

and for template at t,

$$scope(ir_i^t) = scope(ir_j^t), \forall (ir_i^t, ir_j^t) \in Ir \tag{3.14}$$

$$scope(l_i^t) = scope(l_j^t), \forall (l_i^t, l_j^t) \in L \tag{3.15}$$

When the template at t is interfaced with the template at $t+1$, the scopes of leaf latent interface nodes of template at t relate to root interface nodes of template at $t+1$ as follows,

$$scope(ir_i^{t+1}) = scope(ir_j^{t+1}) \Rightarrow scope(l_i^t) = scope(l_j^t), \tag{3.16}$$
$$(l_i^t, l_j^t) \in L, (ir_i^{t+1}, ir_j^{t+1}) \in Ir, f(L_i) \rightarrow ir_i, f(L_j) \rightarrow ir_j$$

From 3.12 and 3.16 we have,

$$scope(l_i^t) = scope(l_j^t), \forall (l_i^t, l_j^t) \in L \tag{3.17}$$

The condition from 3.17 is equivalent to the condition from 3.15. This means the scopes of leaf latent interface nodes of template network at t have not changed after interfacing with the template network at $t+1$. From inductive hypothesis, SPMN R is valid. Since there is no change in scopes of any of the nodes in R after interfacing with template at $t+1$, all sum nodes are complete, product nodes are decomposable and max nodes are complete and unique in R. Since no scope is changed in template network at $t+1$ after interfacing, all sum nodes are complete, product nodes are decomposable and max nodes are complete and unique in the template network at $t+1$. So, all sum nodes are complete, product nodes are decomposable and max nodes are complete and unique in the SPMN formed after interfacing R with template network at $t+1$. Therefore, the SPMN formed after interfacing R with one more template network (template repeated $t+1$ times in total) is valid.

$\square$

Chapter 4

Learning and Evaluation of RSPMN

In this chapter, we present the complete algorithm for structure learning of RSPMN and its evaluation to compute the MEU and the best decisions based on MEU. In Section 4.1, we present the four main parts of the LEARNRSPMN algorithm and a description of the domain that we use for illustration. In Sections 4.2 to 4.5, we describe each step of the algorithm in detail aided by pseudo-code and an illustration. In Section 4.6, we describe the evaluation of RSPMN to compute the MEU. We also show how to determine the best decisions from the network based on the MEU values.

4.1 Structure Learning

Illustration: We use a simple 2X2 grid world to provide illustrations of the application of various steps of the RSPMN structure learning algorithm. The data set is generated from an agent following a random policy for navigating in the 2X2 grid world shown in Fig 4.1. Each cell in the grid is represented by two binary state variables X, Y, which represent the x, y coordinates of the cell, respectively. Here, the top-left cell is $(0, 0)$ and $(1, 1)$ indexes the bottom-right cell. The agent can decide to either move in one of the four cardinal directions or perform a No-op, which is represented using a single decision variable, A. Let $(0, 0)$ be the start state, $(1, 0)$ a penalizing state with a reward of -10, and $(1, 1)$ the goal state with a reward of 10. All transitions are deterministic and cost -1. Reward is represented by the utility variable U. Recall from Section 3.2 that the data schema is $\langle(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^0$, $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^1$, …, $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^{T-1}\rangle$.

For this domain, the information set I_0 consists of variables X, Y, decision variable D_1 is A and the utility values are represented by u. There are no further state or decision variables. So the data schema for the 2X2 grid world is $\langle(x, y, a, u)^0$, $(x, y, a, u)^1$, …, $(x, y, a, u)^{T-1}\rangle$.

A data set of 10K records over 4 time steps T with schema as given is generated by a randomly-acting agent in the domain.

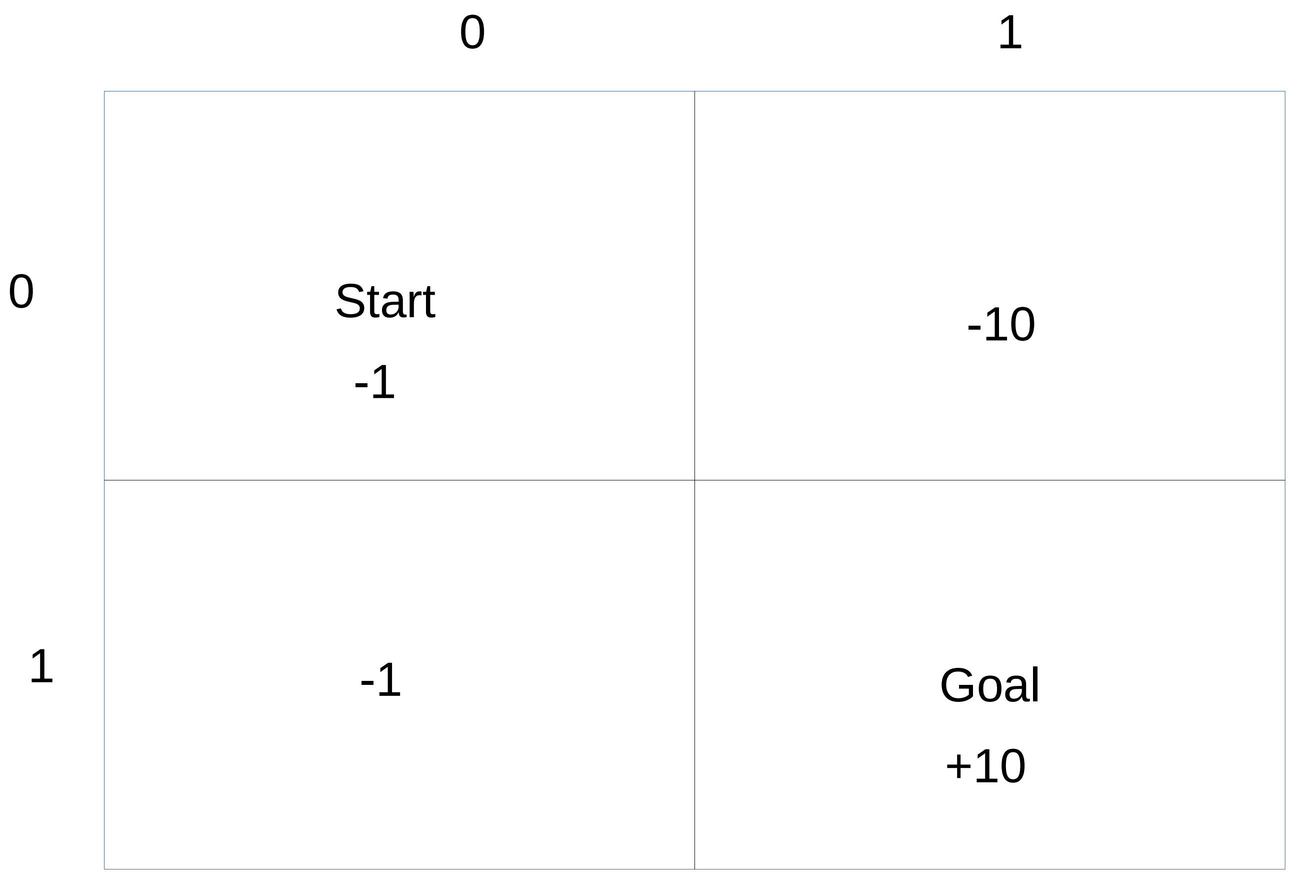

Figure 4.1: 2X2 Grid world used in the illustration of each step of RSPMN strcuture learning

Algorithm: We present a complete algorithm for learning the structures of a valid top network and a sound template network from data whose schema is outlined in Section 3.2 . Each data record of length T is a capture of an episode during which a decision-maker interacts with the environment for T steps (observing the state, acting, and obtaining reward). Let there be E such episodes. For convenience, we denote a tuple $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^t$ as τ^t. Consequently, the data set has E records each consisting of T tuples $\langle \tau^0, \tau^1, \ldots, \tau^T \rangle_e$ where $e = 1, 2, \ldots, E$.

Recall from Section 3.2 that the partial order is given by $P^{\prec}$ is $I_0 \prec D_0 \prec I_1 \prec D_1 \cdots \prec I_p$. Let us also note for the sake of completeness that all variables related to time step $t+1$ assume their values after time step t. This is reflected in the expanded $P^{\prec}$, which now specifies the partial order not only among variables of a single time step but also includes variables of the next time step. This is sufficient because the partial order among variables of two time steps does not change over time.

```
Algorithm 2: LearnRSPMN
    Input: Dataset $\langle \tau^0, \tau^1, \ldots, \tau^T \rangle_e$ where $e = 1, 2, \ldots, E$; Partial Order $P^{\prec}$
    Output: top network, template network
1   $\mathcal{S}^{t=2} \leftarrow$ Run LearnSPMN over wrapped 2-time step data $\langle \tau^0, \tau^1 \rangle_e$   $e = 1, 2, \ldots, T \times E$
2   Create top network and set of root interface nodes $Ir$ from $\mathcal{S}^{t=2}$
3   Create initial template network from $Ir$ and $\mathcal{S}^{t=2}$
4   Revise initial template using sequence data to obtain the final template
```

Algorithm 2 presents the four main steps involved in learning the RSPMN from data. We refer to this algorithm as LearnRSPMN. We describe each of these steps below in more detail, give their algorithms, and also illustrate their applications on a simple 2×2 grid problem.

4.2 Learn an SPMN from 2-time step data

Sequential decision making environments can often be modeled as Markovian. For a Markov domain, the transition from a state $\langle x_0, x_1, \ldots x_n \rangle^t$ in time step t to next state $\langle x_0, x_1, \ldots x_n \rangle^{t+1}$ in time step $t+1$ is dependent only on the state $\langle x_0, x_1, \ldots x_n \rangle^t$ and actions $\langle a_0, a_1, \ldots a_n \rangle^t$ in the previous time step t. Therefore, state transition probabilities and utility functions can be sufficiently learned from data spanning two time steps. Consequently, the first step of LearnRSPMN is to use Melibari et al.'s LearnSPMN algorithm to learn a valid SPMN, $\mathcal{S}^{t=2}$, from 2-time step data. Subsequently, $\mathcal{S}^{t=2}$ serves as a basis for obtaining the template network.

Algorithm 3: SPMN from 2-time step data

```
  input: Dataset: $\langle \tau^0, \tau^1, \ldots, \tau^T \rangle_e$ where $e = 1, 2, \ldots, E$, Partial Order: $P^{\prec}$
  output: SPMN from 2-time step data: $\mathcal{S}^{t=2}$
1 $w^0 \leftarrow$ Empty, $w^1 \leftarrow$ Empty
2 for $e$ in $1, 2, \ldots, E$ do
3     for $t$ in $0, 1, \ldots T-1$ do
4         $w^0 \leftarrow w^0.add(\tau^t)$
5         $w^1 \leftarrow w^1.add(\tau^{t+1})$
6 $W \leftarrow \langle w^0, w^1 \rangle$
7 $\mathcal{S}^{t=1} \leftarrow$ LearnSPMN($W$, $P^{\prec}$)
```

However, which two time steps of the data record should we utilize? One might think that it may be sufficient to limit utilizing tuples of the first two steps in each data record $\langle \tau^0, \tau^1 \rangle$, or to tuples of any particular two consecutive steps $\langle \tau^{t'}, \tau^{t'+1} \rangle$. But, an agent often starts the episode at the same start state and is often at the same intermediate state in a subsequent time step. As such, data in the first two time steps, or for that matter, any fixed pair of time steps, is seldom fully representative of the transition probabilities. Consequently, we consider each consecutive pair of tuples $\langle \tau^t, \tau^{t+1} \rangle$ $t = 0, 1, \ldots, T-2$ in each data record and wrap it to create a data set with $T \times E$ rows spanning two time steps as shown Algorithm 3. Note that in order to observe the goal state in the tuple τ^t, padding values can be added into τ^{t+1} where any transition from goal state to next state is goal state with a zero utility value.

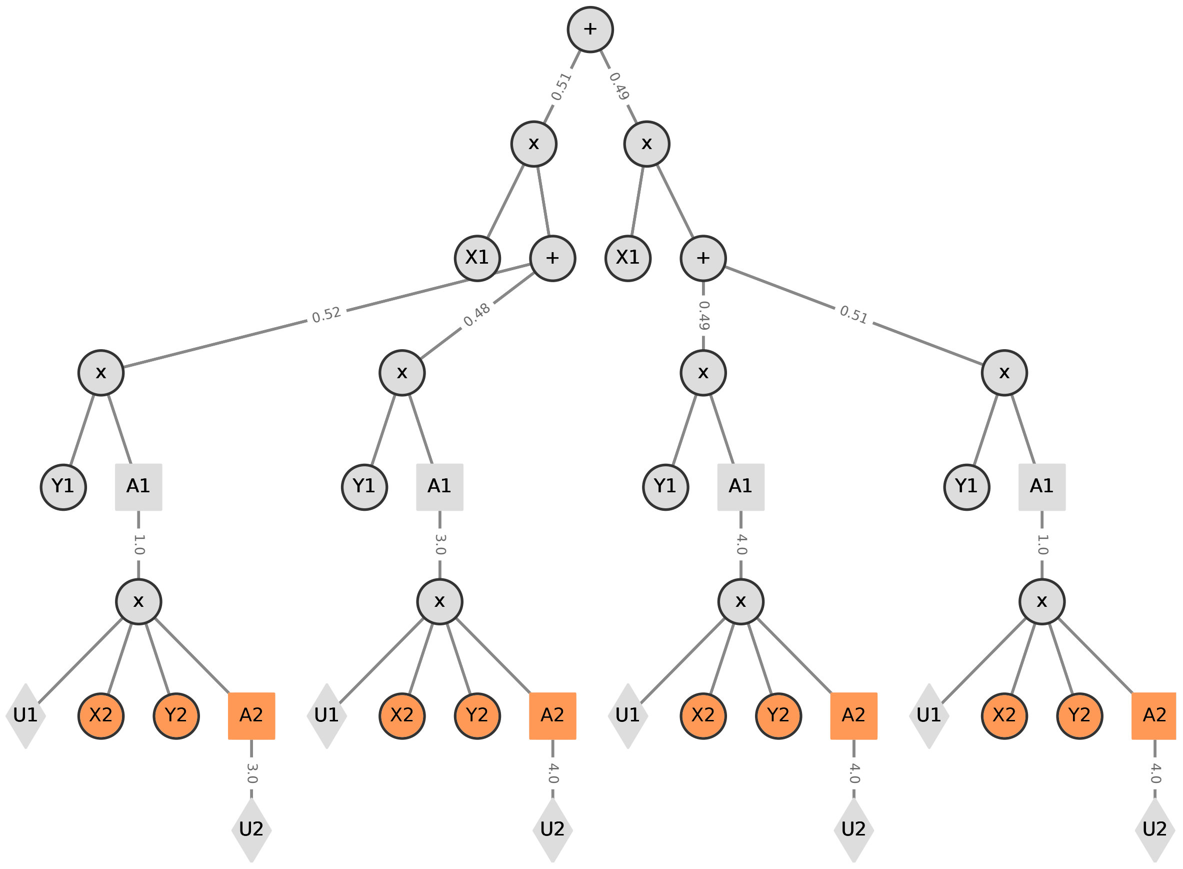

Figure 4.2: SPMN $\mathcal{S}^{t=2}$ learned on wrapped 2-time step data for the example grid problem. Xt, Yt, At, and Ut represent the corresponding variables for step t, where $t \in \{1, 2\}$.

Note that $P^{\prec}$ is focused on the partial order among the variables of two consecutive time steps, and need not change for use in LEARNSPMN. Then, LEARNSPMN is run over the wrapped data set with $P^{\prec}$ as the partial order. We show the learned 2-time step SPMN for the example grid problem in Fig. 4.2.

4.3 Obtain top network and Ir nodes

To obtain the nodes in Ir, we extract a 1-time step network $\mathcal{S}^{t=1}$ from $\mathcal{S}^{t=2}$. We point out that it is necessary to use a 2-time step SPMN to obtain $\mathcal{S}^{t=1}$. To realize this, let a state-action pair $\langle s_0, a_0\rangle$ transition to state s_1 while $\langle s_2, a_0\rangle$ transition to s_3. If $\mathcal{S}^{t=1}$ is learned from data of a single time step, the correlations between variables of different steps is obviously is not ascertained. Due to this, both state values s_0 and s_2 may get included in a single substructure. However, the 2-time step data makes it clear that they effect differing transitions, which could be identified during independence testing, thereby modeling them separately by creating different clusters for each of these states in the first step as they result in a transition to the next step. Importantly, this identifies the behaviorally distinct variables of the domain, which in turn helps in identifying distinct states. This helps create an interface node for each of the states in the template network.

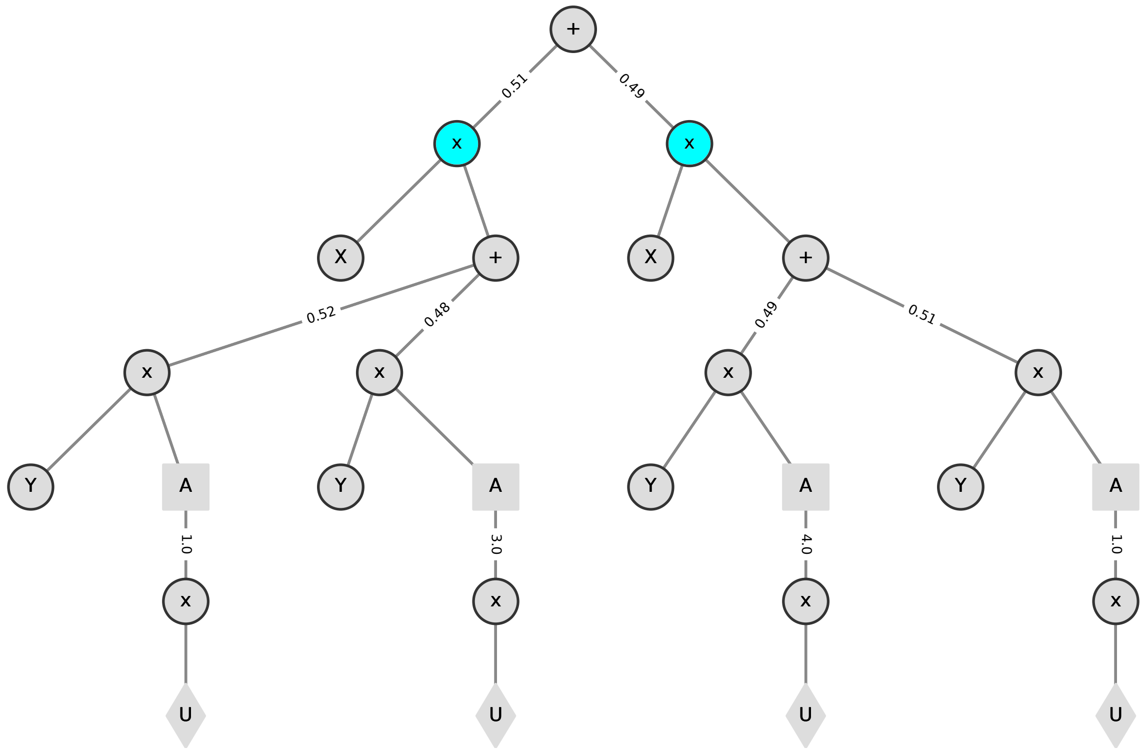

Figure 4.3: $\mathcal{S}^{t=1}$ obtained from the $\mathcal{S}^{t=2}$ of Fig. 4.2 with the orange nodes removed.

As shown in Algorithm 4, to obtain $\mathcal{S}^{t=1}$, we simply remove all those sum nodes whose immediate children have scopes that consist of subset of variables in the next time step $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^1$. If there are no such sum nodes, but instead variables of the next time step are directly linked to product nodes (as in Fig. 4.2 where the orange nodes are children of the product nodes), then we remove the nodes corresponding to these children. This results in $\mathcal{S}^{t=1}$ with no nodes whose scopes lie in variables of the next time step, yet appropriately modeling its impact on the first time step. Figure 4.3 illustrates $\mathcal{S}^{t=1}$ for the grid problem.

Algorithm 4: 1 step SPMN from 2 step SPMN

```
  input: Two step SPMN: $\mathcal{S}^{t=2}$
  output: Top Network $\mathcal{S}_O$, Set of root interface nodes $Ir$
1 $Queue \leftarrow \mathcal{S}^{t=2}.root$
2 while $Queue$ is not $\emptyset$ do
3     $node \leftarrow Queue.dequeue$
4     if $node$ is product then
5         for each child $c$ in the set of node's children $C_n$ do
6             if $c$ does not have any variable of $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^1$ in its scope then
7                 Remove $c$ from $C_n$
8 One step network $\mathcal{S}^{t=1} \leftarrow$ remaining $\mathcal{S}^{t=2}$
```

Now, we may obtain the nodes in Ir using $\mathcal{S}^{t=1}$ (shown in Algorithms 5 and 6). Starting from the top-most product nodes (the blue nodes in Fig. 4.3), the root interface nodes are obtained by identifying all the distinct state distributions from these product nodes. This is done by recursively traversing all the branches of the product node until we find a product node without any sum node as a child.

This corresponds to the four product nodes in Fig. 4.3 below the blue colored product nodes. Each of these product nodes as well as leaf nodes of all the parent product nodes on path to this product node are added to a corresponding set. A product node is created for *each* of these sets and the elements of the set are added as children of the product

Algorithm 5: Create top network and set of root interface nodes

```
   input: Two step SPMN: $\mathcal{S}^{t=1}$
   output: Top Network $\mathcal{S}_O$, Set of root interface nodes $Ir$
1  set of nodes seen $E \leftarrow \emptyset$ ; root interface nodes $Ir \leftarrow \emptyset$
2  $Queue \leftarrow \mathcal{S}^{t=1}.root$
3  while Queue is not $\emptyset$ do
4      $node \leftarrow Queue.dequeue$
5      if (node is product and all $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^0$ are in node.scope) then
6          topProdChildren $C \leftarrow \emptyset$
           /* c.scope must be a proper subset */
7          if (each $c.scope \subset (I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^1, \forall c \in node.children$) then
8              $C \leftarrow node.children$
9              $node.children \leftarrow \emptyset$
10         if C is not $\emptyset$ then
11             Create Product node $P$ ; $P.children \leftarrow C$
12             $R \leftarrow IrChildren(P)$
13             latent interface nodes $L \leftarrow \emptyset$
14             for each irchildren set $ir_C$ in $R$ do
15                 Create interface root product node $ir$
16                 $ir.children \leftarrow ir_C$
17                 Create latent interface node $l$ with a bijective mapping $f(l) \rightarrow ir$
18                 $L \leftarrow L \cup l$ ; $Ir \leftarrow Ir \cup ir$
19             Create Sum node $S_L$ with $S_L.children \leftarrow L$
20             $node.children \leftarrow node.children \cup S_L$
21         else
22             for each child $c$ in the set of node's children $C_n$ do
23                 if $c$ is not in $E$ then
24                     Add $c$ to $E$
25                     Enqueue $c$ into $Queue$
26 Top Network $\mathcal{S}_O \leftarrow$ remaining $\mathcal{S}^{t=1}$
27 Set of root interface nodes $Ir$
```

node as shown in Fig. 4.5. Each of these product nodes is a root interface node. Each of these interface nodes holds an SPMN that corresponds to a state distribution (there are four

Algorithm 6: make sets of Ir children

```
    input: Product node $P$
    output: set of interface root node children sets
1   Function $IrChildren(P)$:
2       if (any $c$ is $Sum, \forall c \in P.children$) then
3           for each $c$ in $P.children$ do
4               stateVars $X \leftarrow \emptyset$
5               if $c$ is $Sum$ then
                    /* a set of sets                                */
6                   set of $IrChildren$ sets $R \leftarrow \emptyset$
7                   for each $c_p$ in $c.children$ do
8                       $irChildren$ set $ir_C \leftarrow IrChildren(c_p)$
9                       $R \leftarrow R \cup ir_C$
10              else
11                  $X \leftarrow X \cup c$
12          for each $ir_C$ in $R$ do
13              $ir_C \leftarrow ir_C \cup X$
14          return $R$
15      else
16          return $\{\{P\}\}$
```

interface nodes for the four states in illustration). The interface nodes, for example, can help learn the probability of transitioning from one state s^t on taking a^t to some other state s^{t+1} in the next time step. *Observe that union of the scopes of the children of each interface node in Ir is identical. This makes the scopes of all the interface nodes Ir identical.*

The top network is then simply a sum node with as many children as the nodes in Ir. The weights on these edges are equal and correspond to a uniform distribution. Each of these children is a latent interface node with a bijective relationship to a root interface node. We show the top network in Fig. 4.4. The weights on the root sum nodes can be learned using the wrapped two step data obtained in Section 4.2.

Alternatively, the top network can be created by removing the children of the top blue

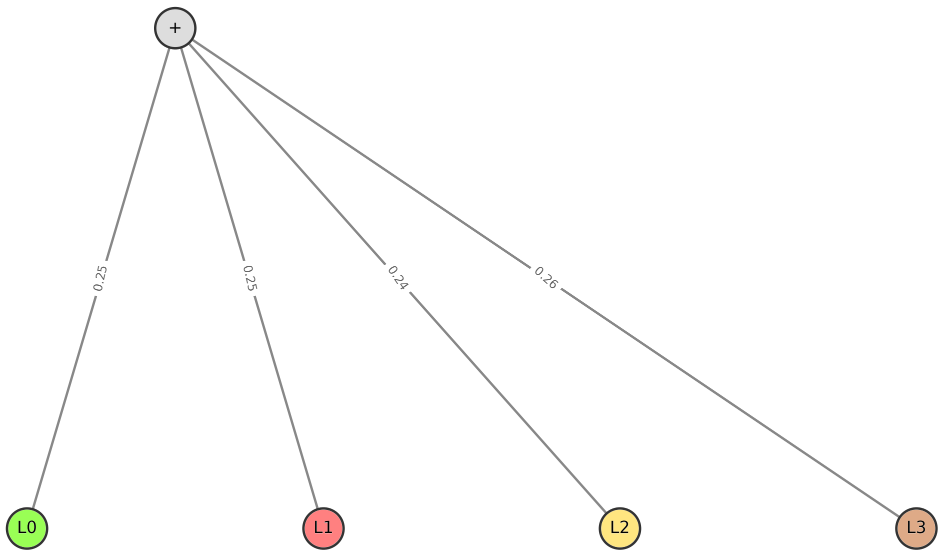

Figure 4.4: The top network with color-coded latent interface nodes indicating a bijective relationship with the similarly color-coded nodes in Ir.

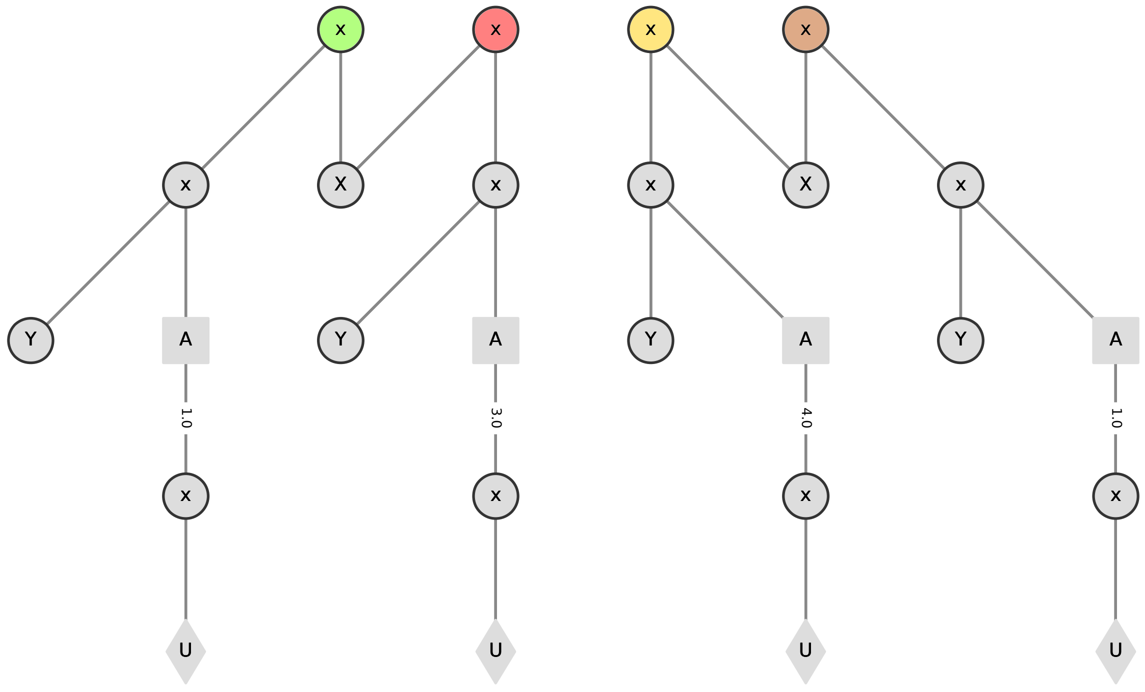

Figure 4.5: Root interface nodes obtained using the network in Fig. 4.3. The colored product nodes function as the root interface nodes.

product nodes and adding a sum node with children as a latent interface leaf node for each interface root node created from these product nodes. The weight on each out-going edge from this sum node to latent interface node can be obtained by multiplying the weights of out-going edges from sum node to product node that leads to the state distribution the latent node corresponds to. This corresponds to weights $(0.52, 0.48, 0.49, 0.51)$ on out-going edges from sum nodes below the blue product nodes in Fig 4

4.4 Building an initial template

The network from the previous step containing the root interface nodes forms a basis for creating the initial template. Let $|Ir|$ denote the number of root interface nodes created in the previous step. We begin by creating a subnetwork S_L rooted at a sum node with as many children as $|Ir|$. Each of the children is a leaf *latent interface node* observing the following relationship, $f(l_i) = ir_i$, $i = 1, 2, \ldots, |Ir|$, and f is a bijective relationship. In other words, each of the latent interface node corresponds to a distinct root interface node. The weights on the edges are equal and correspond to a uniform distribution.

Beginning at *each* root node in Ir, we traverse the graph to the bottom-most sum, product, or max node. In case of a product node, we add a new edge and link it to a new subnetwork S_L. In case of a sum or max node, each of its children nodes is now replaced by a product node with two outgoing edges. One of these edges links to the previous child node while the other edge links to S_L. Including all latent nodes in S_L can be effectively thought of as having observed a state and taken a decision at time step t results in reaching all the other states in the next time step $t + 1$ with equal probability. We show the initial template network for the example grid problem in Fig. 4.6.

Since, each latent interface node is added by using a bijective mapping f, all the latent nodes l_i have same scope. Hence, sum node S_L is complete. Adding S_L as described above

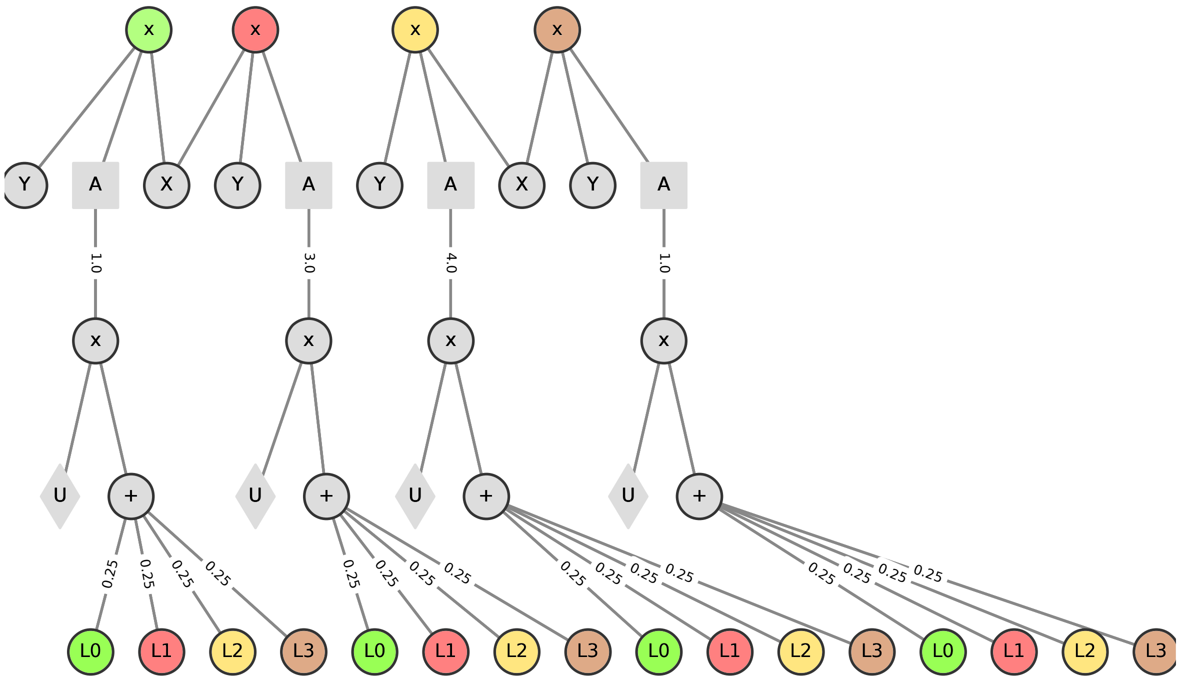

Figure 4.6: An initial template network for the grid problem obtained by attaching S_L to the bottom-most product node following each node in Ir. Similarly colored nodes in the sets L and Ir are related.

does not change properties of any of the nodes for validity. Moreover, scope of all I_r is same. Therefore, the initial template network is sound.

4.5 Learning the final template network

Parameters of the template network are the edge weights of the outgoing edges from the sum nodes including S_L. We adapt the hard expectation-maximization algorithm for SPNs [23, 21] to the recurrent structure of the template network to update the structure and parameters of the initial template. Broadly, it involves performing a bottom-up pass during which the likelihoods of each sum, product, and max node are calculated using a data record

Algorithm 7: Create initial template network

```
input:  Set of root interface nodes $Ir$
output: Initial template root interface nodes $Ir$
1  $L \leftarrow \emptyset$
2  for $each\ ir\ in\ Ir$ do
3      Create a Latent interface node $l$ and $f(l) \rightarrow ir$
4      $L \leftarrow L \cup l$
5  for $each\ ir\ in\ I$ do
6      set of nodes seen $E \leftarrow \emptyset$
7      $Queue \leftarrow ir.root$
8      while $Queue\ is\ not\ \emptyset$ do
9          $node \leftarrow Queue.dequeue$
10         if $(each\ c\ is\ Leaf\ node,\ \forall c \in node.children)$ then
11             Create Sum node $S_L$ /* bottom sum interface node */
12             $S_L.children \leftarrow L$
13             $S_L.weights \leftarrow 1/numOf(S_L.children)$
14             if $node\ is\ product$ then
15                 Add $S_L$ as child of product node
16             else
17                 for $each\ l\ that\ is\ Leaf\ node\ in\ set\ of\ node.children\ C_n$ do
18                     Create Product node $P$
19                     $P.children \leftarrow S_L \cup l$
20                     $C_n \leftarrow C_n \setminus l$
21                     $C_n \leftarrow C_n \cup P$
22         else
23             for $each\ c \in node.children$ do
24                 if $c\ is\ not\ in\ E$ then
25                     Add $c$ to $E$
26                     Enqueue $c$ into $Queue$
27 Initial template root interface nodes $Ir \leftarrow$ set of root interface nodes $Ir$
```

$\langle \tau^0, \tau^1, \ldots, \tau^{T-1} \rangle$. [1] This is followed by a top-down (backpropagation) pass beginning at the rooted top network, which selects a maximum likelihood path and updates the counts on

[1]If the leaf node is a distribution instead of being an indicator variable, then entering a value yields a likelihood as well.

the edges from sum nodes along that path.

Data from the last tuple τ^{T-1} is entered in the leaf random variable nodes of the bottom-most template (recall that the bottom-most template network has its leaf latent nodes removed). Likelihoods are propagated upwards through the network by performing the sum, product, and max operations represented by the nodes until we obtain a likelihood for each in node in Ir^{T-1}. Using the bijection function that relates the nodes in L with the nodes in Ir, we may propagate the likelihoods of the nodes in Ir^{T-1} to the corresponding latent nodes in L^{T-2}. The above mentioned bottom-up pass is repeated using the likelihoods of the leaf latent nodes and data from tuple τ^{T-2} entered into the leaf random variable nodes of time step $T-2$ template, thereby yielding another set of likelihoods for the nodes. Regressing in data to time step 0, the bottom-up pass continues assigning a likelihood to each node in the network terminating when the root node of the top network is reached.

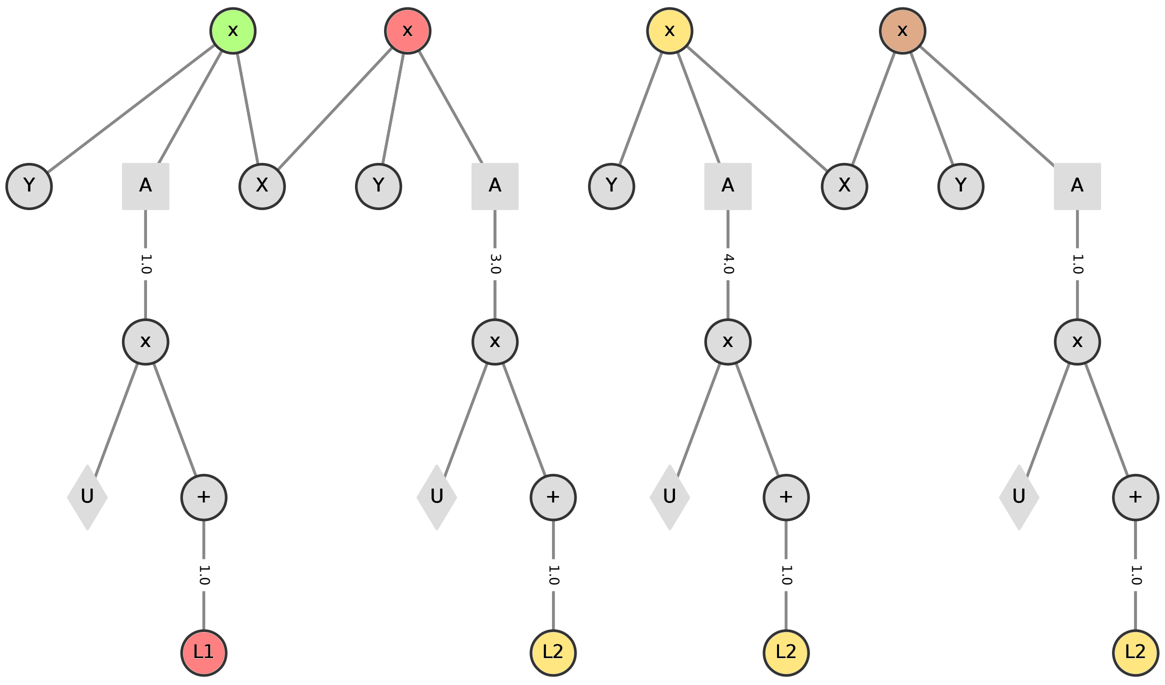

Figure 4.7: The final template network from the bottom and top-down passes for the grid problem. Notice that we retain the leaf latent nodes at each S_L in the initial template with the maximum likelihoods only.

Given the likelihoods computed at each node for each time step, the top-down pass begins at the root node of the top network and at time step 0. It traverses downward visiting each node, selecting the child node with the highest likelihood for each sum node (including subnetwork S_L) and updating the count (initialized at zero) on the edge connecting the sum node to the child, selecting the child with the highest likelihood for the max node, and following each edge of the product node. The bijection mapping is used to go from the leaf latent nodes with maximum likelihood in time step $t-1$ to the mapped root interface nodes of time step t. An edge from a sum node chosen again has its previous count incremented by 1. The top-down pass terminates at the bottom-most network representing time step $T-1$.

New weights of outgoing edges from each sum node are obtained as: $\frac{\text{count on edge from sum} \rightarrow \text{child node}}{\#\ \text{sum node visited}}$. Thereafter, any leaf latent nodes (and indeed any children of a sum node) with zero counts are pruned. We may perform both the bottom-up and top-down passes without actually unfolding the template network by following the implicit links represented by the bijection mapping. Applying this step, the learned final template for our illustrative grid problem is shown in Fig. 4.7.

Since we only drop the latent interface nodes, the scope of S_L does not change and the template network is still sound.

4.6 MEU and Policy Evaluation

We may evaluate the RSPMN formed by interfacing the top network with the learned final template network iterated as many times as the length of each data record to compute the MEU for each state and obtain a policy from the MEU values.

The MEU value is obtained by evaluating the template network bottom-up as in an SPMN as described in Section 2.3.2. The utility values of the leaf latent interface nodes of the bottom-most template (last time step) are set to zero. After evaluating the bottom-most

Algorithm 8: Final template network

```
input: Initial template root interface nodes $Ir$; Top Network $\mathcal{S}_O$
output: Final template root interface nodes
1  $T \leftarrow$ length of sequence
2  $ll^{-1}, ll^0, \ldots, ll^{T-1} \leftarrow$ EmptyList
3  for each time step $t$ from $T-1 \ldots 0$ do
4      if $t = T$ then
5          Remove leaf latent nodes from $Ir$
6          Assign leaf values $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^t$ to leaves in $Ir$
7          $ll^t \leftarrow$ bottomUpPass($Ir$)
8      else
9          Assign leaf values $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^t$ to leaves in $Ir$
10         Assign $Ir$ values from $ll^{t+1}$ to latent leaves in $Ir$
11         $ll^t \leftarrow$ bottomUpPass($Ir$)
12 Assign $Ir$ values from $ll^0$ to latent leaves of $\mathcal{S}_O$
13 $ll^{-1} \leftarrow$ bottomUpPass($\mathcal{S}_O$)
14 for each time Step $t$ from $-1 \ldots m-1$ do
15     if $t = -1$ then
16         TopDownPass($\mathcal{S}_O$)
17         $l \leftarrow$ leaf latent interface node reached
18     else
19         $I_l \leftarrow$ root interface node corresponding to $l$
20         TopDownPass($I_l$)
21         Increment counts on outgoing edges visited on sum nodes
22         $l \leftarrow$ leaf latent interface node reached
23     Update weights on sum node $S_L$ whose children are $L$ in $Ir$ using counts
24     Drop branches with zero counts on $S_L$
25     Final template root interface nodes $Ir \leftarrow Ir$
```

network, each root interface node of the template network holds a utility value. In the next iteration, the utility values of the leaf latent interface nodes are assigned the utility values of the corresponding root nodes computed in the previous iteration, and the process is repeated until time step 0 and the top network is evaluated. Assuming that the template network learned a model of the true transitions well, each bottom-up pass through the template can be thought of as performing one Bellman update in the value iteration technique.

In the first iteration, the utility values at leaf latent nodes are set to 0 and all the leaf nodes are marginalised by setting all the leafs to a likelihood of 1. The values are passed up by applying the operators at each node for likelihood and utility as described in as described in Section 2.3.2. At the end of first iteration, each root interface node, which corresponds to a particular state in the domain holds a utility value and a corresponding likelihood. In the next iteration, the utility and likelihood values of these root interface nodes are set to the corresponding leaf latent nodes from which they are mapped. This can be repeated until the length of the data sequence T or even infinitely until utility values converge. Subsequently, each node of the template holds a corresponding utility value.

Recall from Section 3.2 that before making a decision at a decision node D_i the variables preceding it in the partial order $P^{\prec}$ are observed. To obtain a decision value at a given decision variable D_i,and some observation of variables in the information sets preceding D_i, the above process to compute MEU is run until $T-1$ iterations $(T, T-1, \ldots, 1)$ or until values converge. In the next iteration, the template network is interfaced with the top network and variables are assigned values corresponding to observed state. Leaf latent nodes are assigned values that correspond to their mapped root interface nodes from the previous iteration. The interfaced network is then evaluated as an SPMN and a top-down pass is done to select the best decision at D_i as in an SPMN.

As we can see, each iteration is linear in size of the template network and it requires at most T iterations to compute the MEU for a horizon of T time steps. Hence, the complexity of computing the MEU for a finite horizon of T time steps is $O(TE)$ where E is the size of template network measured in terms of edges and T is the number of time steps.

Chapter 5

Experiments

In this chapter, we report the experimental results of the modified LearnSPMN and the LearnRSPMN algorithms and provide an analysis of the results. In Section 5.1, we discuss the improvement in likelihood of modified LearnSPMN over the original LearnSPMN algorithm. In Sections 5.2, we discuss in detail the performance of LearnRSPMN algorithm using various metrics on a test-bed of data sets collected from seven sequential decision-making domains that adhere to the schema given in Section 3.2. We show that the LearnRSPMN algorithm reaches close to optimal values by comparing it against Value Iteration and DQN [17] techniques . We also show that it outperforms modified LearnSPMN and Batch-constrained Q-learning [6] on various domains.

We implemented modified LearnRSPMN in the SPFlow library [18]

5.1 Modified LearnSPMN

Table 5.1 shows a comparison of likelihoods between original LearnSPMN algorithm against the modified LearnSPMN algorithm. We tested it on a test-bed of five data sets. Four of these data sets (except Marbles) are from [15]. Notice that the MEU values for Ex-

Dataset	**Original** LearnSPMN	**Modified** LearnSPMN **I**	**Modified** LearnSPMN **I&C**	**MEU**
Marbles	-18.77	-0.86	-0.52	4.49
Export Textiles	-12.67		-1.08	1917933.05
Test Streptococal	-62.04	-1.14	-1.12	54.92
Computer Diagnostic	-47.18	-0.89	-0.89	242.80
Power Plant Air Pollution	-19.61	-2.03	-1.82	$1.77E-16$

Table 5.1: Log likelihood on original LearnSPMN vs Modified LearnSPMN algorithm. **I** column gives the likelihood values on Modified LearnSPMN with modifications on independence testing, and **I&C** on both independence and clustering changes. **MEU** column gives the MEU values on **I&C** changes of modified LearnSPMN

port textiles and Power Plant Air Pollution data sets are large positive and negative values. But these utility values are reflected in the data sets. Notice the improvements in the log-likelihoods after the modifications. The log-likelihoods of all the data sets improved in orders of magnitude after the modifications. While the difference between independence changes, independence and clustering changes in the modified LearnSPMN are small, we observed more sensible structures on both independence and clustering changes compared to just independence changes.

5.2 LearnRSPMN

In this section we discuss at length the experimental results of the LearnRSPMN algorithm on various metrics. We begin by describing evaluation testbed, dataset sizes and the data collection strategy. We then draw comparison between RSPMNs and SPMNs on sizes, learning and evalution times. We then compare the MEUs and policies obtained using RSPMNs, SPMNs, **BCQs** against the optimal MEUs and policies.

5.2.1 Evaluation testbed

There are few existing data sets on simulations of decision-making domains. Due to this, we created a new testbed of seven data sets, listed in Table 5.2 and available for download at `https://github.com/c0derzer0/RSPMN`. Four of these data sets are simulations of these domains present in OpenAI's Gym and the remaining three are simulations of RDDLSim [24] MDP domains. Each data set is generated by using a random policy which interacts with the environment and collecting the ⟨state, action, reward⟩ generated at each step. Each episode is run for T time steps, which is selected to be sufficient to reach the goal state. A new episode is started if either the last time step is reached or if the agent reaches the goal state or some other terminal state.

Data set	$\|X\|$**,** $\|D\|$	**#Episodes**	T	**|Columns|**	**|SPMN|**	**|RSPMN|**
GridUniverse^1	(1, 1)	100K	8	24	138,492	(13, 210)
FrozenLake^1	(1, 1)	100K	8	24	1,068,246	(18, 401)
Maze^1	(2, 1)	100K	8	24	352,312	(11, 184)
Taxi^1	(4, 1)	20K	50	150	-	(80, 1815)
SkillTeaching^2	(12, 4)	100K	10	170	-	(137, 4878)
Elevators^2	(13, 4)	200K	10	180	-	(143, 5390)
CrossingTraffic^2	(18, 4)	100K	15	345	-	(82, 2349)

Table 5.2: Superscript 1 denotes simulations of Gym domains and ^{2} denotes simulations of RDDL-Sim domains. $|X|, |D|$ gives the numbers of state and decision variables in the domain, |Columns| is the total number of columns in each data record. We also report the size of the learned structures. |SPMN| and |RSPMN| gives the sizes of the (top, template) networks respectively. ' - ' denotes that the SPMN was not learned in 12 hrs on an Ubuntu Intel i7 64GB RAM PC.

For each data set in the testbed, we learn an SPMN using the LearnSPMN algorithm. This was made possible by padding the sequences so that all sequences have the same length – on reaching a terminal state, the agent stayed in that state until the length of the sequence is T. The top and template networks of an RSPMN were learned using our LearnRSPMN (Algorithm 2).

5.2.2 SPMN and RSPMN size comparision

We show the sizes of the learned networks as the total number of nodes in each, in the ultimate two columns of Table 5.2. Notice the blow up in the sizes of the SPMNs learned for the sequential data sets. For the larger RDDLSim domains, the sizes of the top and template networks also grow but we did not observe a disproportionate growth, while the SPMNs could not be learned.

The blow up in the sizes of the SPMNs happens because the SPMN has many repeated structures for the state distributions over time. The SPMN grows in depth with time step T creating new structures to model data at each time step. Each of these is also connected to previous time step structures. If a same state s is reached from multiple states say k in the previous step, k copies of structures corresponding to s are connected to k states from previous step. This leads to a blow up in the number of nodes in the SPMN as the time steps T grow. Whereas, the LEARNRSPMN algorithm identifies a distinct state and creates only one copy corresponding to its data distribution.

5.2.3 MEU and policy comparisons

Table 5.3, which reports the key results, compares the MEU from the start state of each domain as obtained by evaluating the learned RSPMNs and any learned SPMNs with the (near-)optimal values. We obtained the latter from converged DQNs for the Gym domains and by solving the MDP using value iteration for the RDDLSim domains. Observe that RSPMNs yield MEUs that are very close to the optimal and significantly better than those from the learned SPMNs. Clearly, the sequential data sets do not have sufficient episodes to learn high-quality SPMNs. Moreover, it may not be possible to learn an SPMN if we increase the size of the datasets. Also, since the SPMN is learnt over specific number of time steps, there is no guarantee that the MEU values from SPMN are converged values.

	MEU			Average reward			
Data set	**Optimal**	**RSPMN**	**SPMN**	**RSPMN**	**BCQ**	Δ %	**LL (RSPMN)**
GridUniverse	6	6	6	5.9	5.9	0	-0.87
FrozenLake	0.8	0.818	0.13	0.8	0.3	62.5	-6.17
Maze	0.966	0.966	0.052	0.96	0.96	0	-0.86
Taxi	8.9	9	-	8.9	-200	60.25	-2.45
SkillTeaching	-3.022	-3.06	-		-5.42	83.3	-2.09
Elevators	-7.33	-7.47	-		-1.52	80	-4.8
CrossingTraffic	-4.428	-4.425	-		27.98	94.7	-8.44

Table 5.3: Our key results comparing the MEUs of the optimal policy, learned RSPMNs, SPMNs, and batch-constrained Q-learning. Δ % gives the policy deviations between the policies obtained from RSPMN and the optimal ones.

RSPMNs and SPMNs also represent a model of the environment as present in the data, which then plays a role in the MEU and policy computation. So, how well did the structure learning method capture the environment dynamics? To answer this question, we simulated the policies obtained from the RSPMNs in their respective Gym environments and noted down the average rewards. [1] Sometimes, it may happen that the MEU values returned from the RSPMN may be close to optimal values, but the policy modeled by the structure may not be optimal. It may also happen that some states may end up having values close to optimal MEU. Deploying the policies on the domain and calculating the average rewards helps bolster the MEU values obtained from the RSPMN as it shows the performance of the policy on the domain.

Table 5.3 shows that these average rewards nearly match the MEUs obtained directly from the RSPMNs. This implies that the networks are modeling the environment accurately and the policies and MEUs obtained are indeed optimal. We also compare with the average rewards of policies learned by the discrete batch-constrained Q-learning [7] on the data sets as a baseline, a technique similar to DQNs but constrained to learning from a batch of data. For RDDLSim domains, we report on the Q-values of the start states. Clearly, BCQ

[1]We are currently implementing the RDDLSim domains in Gym to allow simulations of our policies.

expects far more data to learn a good policy. We also made an interesting observation that when BCQ was allowed to learn using data generated from a sub-optimal behavioral agent (DQN) on infinite horizon on FrozenLake, BCQ was able to generate close to optimal average reward. But when the environment was restricted to a horizon of 10, it failed to learn a good policy and the average rewards fell to 0.1. On the other hand, RSPMNs generalise well on data with limited number of time steps. It could also be possible that the default Neural Networks and parameters of the publicly available BCQ implementation from the author may not have been sufficient to model the domains accurately. This could indicate that it may be hard for BCQ to generalise well on data with smaller horizons or the NNs used may not be sufficient to model the data.

Next, we report the deviation in policy learned by the RSPMN from the optimal one. This is the total number of states where the actions differ and reported as a percentage Δ % of the total number of states. Notice the large deviations for FrozenLake, Taxi, and the RDDLSim domains although the learned policies show MEUs close to the optimal. This is likely due to the presence of multiple optimal policies in these large domains. Again, the fact that the average rewards from RSPMN policies are close to optimal MEUs show that the learned policy from RSPM is one of the optimal policies. For the sake of completeness, we also report the log likelihoods of the models in the last column.

	Template learning			**MEU eval**	
Data set	**Initial**	**Final**	**SPMN learning**	**RSPMN**	**SPMN**
GridUniverse	2m 26.33s	1m 1.49s	4h 25m 31.72s	0.72s	8.8s
FrozenLake	1m 49.8s	2m 02.78s	12h 5m 40.77s	23.21s	1m 26.85s
Maze	2m 51.19s	54.84s	2h 31m 49.51s	0.62s	24.87s
Taxi	9m 21.79s	2h 28m 15.75s	-	18.45s	-
SkillTeaching	59m 5.87s	29m 28.49s	-	3.84s	-
Elevators	1h 19m 3.91s	4h 19m 29.53s	-	20s	-
CrossingTraffic	8m 46.14s	1h 37m 53.17s	-	18.45s	-

Table 5.4: Initial and final template learning times are shown in first two columns. These are significantly less than those learning the large SPMNs. Run times of MEU evaluations on the learned SPMNs and RSPMNs are shown next.

Table 5.4 shows our final set of results on the clock time it takes for learning the initial template structure, learning the final template of the RSPMN and learning the large SPMNs when possible. The time to learn an SPMN was capped at about 12 hrs. Observe that both learning and evaluating the large SPMNs takes a *few orders of magnitude* longer than learning the templates. However, the template learning times also increase for the larger RDDLSim domains with Elevators taking more than 4 hours. On the other hand, the MEU evaluation remains quick for all the domains taking less than a minute.

Chapter 6

Conclusion and Future work

Sum-Product Networks are interesting because they have the twin benefits of tractable inference and data-driven learning which is difficult in other probabilistic graphical models. Several extensions to SPNs like Recurrent SPNs that generalise SPNs to sequential reasoning and Sum-Product-Max Networks that extend them to decision-making are appealing because they represent a shift in paradigm in probabilistic models from expert-driven handcrafted models to data-driven learning.

First, we presented a modified structure learning algorithm for SPMNs with detailed as well as intuitive reasoning for the modifications. We then described in depth the evaluation of SPMNs to compute the MEUs and obtain best decisions based on MEUs. We showed from our experiments that, the modifications to LearnSPMN improve the log-likelihoods by orders of magnitude compared against original LearnSPMN algorithm.

Along the lines of the recent developments and extensions of SPNs, we presented RSPMNs, a new graphical framework to model sequential decision-making problems in perfectly-observed contexts. We first presented the theoretical framework of RSPMNs. We defined a template network and a top network, that constitute an RSPMN. We defined a soundness property of the template network which enables the RSPMN to be valid. We proved this by showing that an unfolded RSPMN forms a valid SPMN.

Having defined the constituents of RSPMNs and their properties for forming a valid structure, we proposed a novel four step structure learning algorithm LEARNRSPMN to learn a sound template network. We gave detailed reasoning for each of the four steps coupled with pseudo-code and illustrations. We then described in detail the evaluation of RSPMN which performs a Bellman update(assuming template network closely models true transitions) to compute the MEU and obtain the best decisions. We also showed that the complexity of computing the MEU is linear in size of the template network.

We demonstrated from our experiments that, recurrent generalizations of SPMNs do not suffer from an exponential blow-up in size with sequence length that SPMNs suffer from by comparing the sizes, learning times and MEU computation times of RSPMNs and SPMNs. We also showed that RSPMNs reach optimal values on batch RL applications and outperform SPMNs and neural network based **BCQs**.

One of the limitations of model is that it creates a root interface node for each state of the domain reflected in the data set. This may lead to a large growth in size of the network if the state state space is large. Also, these models work efficiently on fully observed domains.

A logical extension of this work would be to extend them to partially observable domains. Another interesting area would be to investigate an online structure learning algorithm for RSPMNs. This would be useful as it can be used in conjunction with offline RSPMN on reinforcement learning domains. An offline RSPMN can be learned from already available data using LEARNRSPMN and the resulting RSPMN can be deployed on the environment. While deployed on the environment, the RSPMN structure can be updated in online fashion as more data becomes available. This is realistic because an online structure learning [11] for an RSPN already exists and modifications specific to RSPMNs may be feasible.

Furthermore, since RSPMNs model the whole environment dynamics instead of learning just a value function, it would be interesting to research their applicability for off-policy evaluations [9] where the environment model is learned from the data, which can be then used for evaluating different policies.

References

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

[2] Adnan Darwiche. "A differential approach to inference in Bayesian networks". In: *Journal of the ACM (JACM)* 50.3 (2003), pp. 280–305.

[3] Adnan Darwiche. "A differential approach to inference in Bayesian networks". In: *Journal of the ACM* 50.3 (2003), pp. 280–305. DOI: 10.1145/765568.765570.

[4] Rina Dechter and Robert Mateescu. "AND/OR search spaces for graphical models". In: *Artificial intelligence* 171.2-3 (2007), pp. 73–106.

[5] Damien Ernst, Pierre Geurts, and Louis Wehenkel. "Tree-based batch mode reinforcement learning". In: *Journal of Machine Learning Research* 6.Apr (2005), pp. 503–556.

[6] Scott Fujimoto, David Meger, and Doina Precup. "Off-policy deep reinforcement learning without exploration". In: *arXiv preprint arXiv:1812.02900* (2018).

[7] Scott Fujimoto et al. "Benchmarking Batch Deep Reinforcement Learning Algorithms". In: *arXiv preprint arXiv:1910.01708* (2019).

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

[9] Omer Gottesman et al. "Combining parametric and nonparametric models for off-policy evaluation". In: *International Conference on Machine Learning*. 2019, pp. 2366–2375.

[10] Jinbo Huang, Mark Chavira, and Adnan Darwiche. "Solving MAP Exactly by Searching on Compiled Arithmetic Circuits." In: *AAAI*. Vol. 6. 2006, pp. 3–7.

[11] Agastya Kalra et al. "Online structure learning for feed-forward and recurrent sum-product networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018, pp. 6944–6954.

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

[13] L.-J. Lin. "Self-Improving Reactive Agents based on Reinforcement Learning, Planning and Teaching". In: *Machine Learning* 8.293–321 (1992).

[14] Mazen Melibari, Pascal Poupart, and Prashant Doshi. "Sum-Product-Max Networks for Tractable Decision Making". In: *Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*. 2016, pp. 1846–1852.

[15] Mazen Melibari, Pascal Poupart, and Prashant Doshi. "Sum-product-max networks for tractable decision making". In: *IJCAI International Joint Conference on Artificial Intelligence* 2016-January (2016), pp. 1846–1852. ISSN: 10450823.

[16] Mazen Melibari et al. "Dynamic Sum Product Networks for Tractable Inference on Sequence Data". In: *International Biennial Conference on Probabilistic Graphical Models (PGM), JMLR: Workshop & Conference Proceedings, Vol 52*. 2016, pp. 345–355.

[17] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[18] Alejandro Molina et al. *SPFlow: An Easy and Extensible Library for Deep Probabilistic Learning using Sum-Product Networks*. 2019.

[19] Iago Paris, Raquel Sanchez-Cauce, and Francisco Javier Diez. “Sum-product networks: A survey”. In: *arXiv preprint arXiv:2004.01167* (2020).

[20] James D Park and Adnan Darwiche. “A differential semantics for jointree algorithms”. In: *Artificial Intelligence* 156.2 (2004), pp. 197–216.

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

[22] Hoifung Poon and Pedro Domingos. “Sum-product networks: A new deep architecture”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2011. ISBN: 9781467300629. DOI: 10.1109/ICCVW.2011.6130310. arXiv: 1202.3732.

[23] Hoifung Poon and Pedro Domingos. “Sum-product networks: A new deep architecture”. In: *12th Conf. on Uncertainty in Artificial Intelligence (UAI)*. 2011, pp. 2551–2558.

[24] Scott Sanner. “Relational Dynamic Influence Diagram Language (RDDL): Language Description”. 2010.

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