On the Development of Deep Convolutional Sum-Product Networks

A Thesis
Submitted to the Faculty of Graduate Studies and Research
In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Computer Science
University of Regina

By
Jhonatan S. Oliveira
Regina, Saskatchewan
December 2019

Copyright © 2020: Jhonatan S. Oliveira

Abstract

A probabilistic graphical model (PGM) is a formal mathematical description of a problem domain. A *Bayesian network* (BN) is a PGM defined by a directed acyclic graph (DAG) and a set of conditional probability tables (CPTs). Two tasks can be employed when reasoning about a problem expressed as a BN: modeling and inference. In modeling, we want to express the problem domain as a DAG in which independencies among the variables involved can be read. There are two main algorithms for testing independencies from a BN DAG, namely, d-separation and m-separation. In this thesis, we begin by introducing *Darwinian networks* (DNs), which are, in some way, a clearer representation of a BN. By using DNs, we derived a new way of testing independencies in a BN, called *rp-separation*, which is a faster alternative to d-separation. Another practical application of DNs was *simple propagation*, which is the current state-of-the-art join tree inference algorithm in BNs.

A *sum-product network* (SPN) is another type of PGM defined by a DAG and a set of parameters. An SPN permits tractable inference, while inference is generally NP-hard in BNs. Furthermore, SPNs can be compiled from BNs or learned from data. In this thesis, we first resolve the inconsistency between the SPN scope definition and the CPT label when compiling a BN into an SPN. Next, we empirically explore new methods for learning an SPN structure and parameters. Finally, we introduce *deep convolutional sum-product networks* (DCSPNs), which use a convolutional neural network to build a correct SPN. DCSPNs exploit the commonly used tensor libraries for neural networks, while still guaranteeing correctness as a PGM. Experimental results show that DCSPNs are comparable to state-of-the-art methods in image completion tasks.

Acknowledgments

I want to thank my supervisor, Dr. Cory Butz, for his teachings and friendship. His support forms the base of my understanding of a researcher. This thesis is the product of our motivated discussions, insightful walks, and (many) cups of coffee.

Also, I thank my family for their continuous support. In particular, my parents, Jair and Luzinete, and my sister Rafaela. This work would not be possible without your love.

Moreover, I want to thank my friends André Evaristo and André (Lobo) Teixeira for their companionship and fruitful research discussions.

Finally, I am grateful to the Faculty of Graduate Studies and Research (FGSR) for their financial support with a Graduate Research Fellowship.

Dedication

To my life pillars, Vó Carlota and Tia Ninil.
Your love is the knowledge I will always seek.

Contents

Abstract i

Dedication iii

Table of Contents iv

Chapter 1 Introduction 1

Chapter 2 Bayesian Networks 6

2.1 Modeling 6

2.2 Inference 8

Chapter 3 Darwinian Networks 12

3.1 Darwinian Networks 13

3.1.1 Introduction 13

3.1.2 Adaptation 16

3.1.3 Evolution 19

3.2 Testing independencies 20

3.3 Performing Inference 24

3.4 Representing Other Bayesian and Non-Bayesian Frameworks 26

3.4.1 Chain Graph Models 26

3.4.2 Relational Databases 29

3.5 Elimination Orderings 32

3.6 Practical Applications of Darwinian Networks 34

3.6.1 Simple Propagation 35

3.6.2 i-Separation 37

3.7 Conclusion 39

Chapter 4 Determining Good Elimination Orderings with Darwinian Networks **43**
4.1 Introduction . . . 43
4.2 Background . . . 44
4.2.1 Elimination Orderings in Bayesian Networks . . . 44
4.3 Elimination Orderings in Darwinian Networks . . . 45
4.3.1 Representing Min-Neighbours in DNs . . . 45
4.3.2 Representing Min-Weight in DNs . . . 45
4.3.3 Representing Min-Fill in DNs . . . 46
4.3.4 Representing Weighted-Min-Fill in DNs . . . 46
4.4 Equivalence . . . 47
4.5 Potential Energy Heuristic . . . 48
4.6 Analysis . . . 49
4.7 Conclusion . . . 51

Chapter 5 Relevant Path Separation: A Faster Method for Testing Independencies in Bayesian Networks **54**
5.1 Introduction . . . 54
5.2 Background . . . 56
5.3 Relevant Path Separation . . . 57
5.4 Experimental Results and Analysis . . . 60
5.4.1 Variations of rp-Separation . . . 62
5.5 Refining m-Separation . . . 62
5.6 Related Work . . . 65
5.7 Conclusion . . . 66

Chapter 6 Exploiting Symmetry of Independence in d-Separation **68**
6.1 Introduction . . . 68
6.2 Darwiche's Implementation of d-Separation . . . 69
6.3 d-Separation Exploiting Symmetry . . . 71
6.4 Experimental Results . . . 75
6.5 Conclusion . . . 76

Chapter 7 Sum-Product Networks **82**

7.0.1 Arithmetic Circuits 82
7.0.2 Sum-Product Networks 84
7.0.3 Mixture Model Interpretation 87

Chapter 8 Resolving Inconsistencies of Scope Interpretations in Sum-Product Networks 88

8.1 Introduction 88
8.2 Scope Under an AC Interpretation 90
8.3 A Scope Definition for Both SPN Interpretations 92
8.4 Analysis 93
8.5 Conclusions 95

Chapter 9 On Learning the Structure of Sum-Product Networks 96

9.1 Introduction 96
9.2 Background Knowledge 97
9.2.1 Chopping Methods 97
9.2.2 Slicing Methods 100
9.3 A Variety of Possible SPNs 103
9.4 Experimental Analysis of SPN Depth 104
9.5 Experimental Analysis of SPN Accuracy 108
9.6 Experimental Analysis of SPN Sparseness 110
9.7 Conclusions 113

Chapter 10 Deep Convolutional Sum-Product Networks 114

10.1 Introduction 114
10.2 Sum-Product Networks 115
10.3 The Building Blocks of DCSPNs 116
10.4 Deep Convolutional SPNs 119
10.5 Experiments 122
10.6 Conclusion 125

Chapter 11 Future Work: Focused Learning in Sum-Product Networks 126

11.1 Introduction 126

11.2 A Two-Phase Method for Focused Learning 128
11.3 Improving Image Completion 129
11.4 Discussion 132
11.5 Conclusions 133

Chapter 12 Conclusion **134**

References **135**

List of Tables

3.1 Four given relations $r_1(a, b, c)$, $r_2(b, c, e)$, $r_3(b, c, d)$, and $r_4(e, f)$. . . . 30

3.2 (i) The universal relation r defined by the join of the four relations in Table 3.1. (ii) The projection of relation r_1 in Table 3.1 (i) onto $\{b, c\}$. 31

3.3 Desired projections of the universal relation r in Table 3.2 (i). 32

3.4 Average time in seconds of LP and SP to propagate messages and compute posteriors over 100 runs in each BN. 41

5.1 Comparison of REACHABLE and RP-REACHABLE with 1000 randomly generated independencies $I(X, Y, Z)$ in each BN. Average times are reported in seconds and winners in bold. 61

5.2 In each BN, 1000 independencies are randomly generated. The third and forth columns indicate whether rp-separation is faster in practice using marking or pruning. The fifth and sixth columns compare m-separation and refined m-separation. Average times are reported in seconds and winners in bold. 63

6.1 With 1% of evidence, comparison of DARWICHE and SYMMETRIC D-SEPARATION with 1000 randomly generated independencies in each BN. 76

6.2 With 5% of evidence, comparison of DARWICHE and SYMMETRIC D-SEPARATION with 1000 randomly generated independencies in each BN. 77

6.3 With 10% of evidence, comparison of DARWICHE and SYMMETRIC D-SEPARATION with 1000 randomly generated independencies in each BN. 78

6.4 With 25% of evidence, comparison of Darwiche and Symmetric d-Separation with 1000 randomly generated independencies in each BN. 81
9.1 Dataset with 4 features and 10 instances. 100
9.2 20 Benchmark datasets used in unsupervised deep learning. 108
9.3 SPN depth for each combination. 109
9.4 SPN accuracy for each combination. 111
9.5 SPN sparseness for each combination. 112
10.1 Mean squared error (MSE) scores in Olivetti Face. 115
10.2 MSE scores in Caltech datasets. 123

List of Figures

2.1 (i) A DAG $\mathcal{B}$. (ii) Sub-DAG $\mathcal{B}^s$. (iii) Moralization $\mathcal{B}^s_m$. (iv) $\mathcal{B}^s_m$ with d and its edges deleted. 7

2.2 [84] Given query $P(e|b = 0)$ posed to BN $\mathcal{B}$ in (i), pruning barren variables g and f in (ii) and independent by evidence variable a in (iii). (iv) is $\mathcal{B}^s_m$. 8

2.3 Applying AR in (2.6)-(2.10) to eliminate a. 9

2.4 LP message passing with evidence $h = 0$. 11

3.1 Unique graphical representations of (3.11)-(3.14), respectively. 14

3.2 (i) Graphically representing the multiplication in (3.15). (ii) Graphically representing the division in (3.16). 15

3.3 Graphically representing the marginalization of variable a in (3.17). . 16

3.4 Testing adaptation twice in the DN $\mathcal{D}$ in (i). 17

3.5 Graphically representing VE's computation. 20

3.6 Graphically representing AR's computation. 25

3.7 Graphically representing LP's computation. 27

3.8 (i) A PDAG. (ii) A DN representing the CGM in Example 16. 28

3.9 A join tree for the semijoin program in Example 19. 31

3.10 DNs can represent the relational database in Example 18. 32

3.11 (i)-(vi) respectively represent the relational database computation in (3.19)-(3.24), given the DNs in Figure 3.10. 33

3.12 Utilizing DNs to visualize the "one in, one out" property exploited in SP. Graphically depicting the potentials in $\mathcal{F}$ of (3.31) at N_2 in (i). After eliminating variable g, (ii) shows $\mathcal{F}$ of (3.32). After eliminating e, (iii) shows $\mathcal{F}$ of (3.33). Finally, eliminating h gives $\mathcal{F}$ in (3.34) shown in (iv). 40

3.13 When testing the independence $I(a, de, g)$ in the DAG shown in (i), i-separation builds the sub-DAG depicted in (ii). 42
4.1 (i) a BN $\mathcal{B}$. (ii) the moralization $\mathcal{B}_m$. (iii) adding edges between a's neighbours in $\mathcal{B}_m$. (iv)-(vii) MN and WMF can determine $\sigma = (c, a, b, e)$. (viii)-(xi) MW can determine $\sigma = (a, c, b, e)$. (xii)-(xv) MF can determine $\sigma = (c, b, a, e)$. 52
4.2 DNs $\mathcal{D}$ in (i) and $\mathcal{D}'$ in (viii). (ii)-(iv) eliminating trait c by merging, replication, and natural selection, respectively. (ii)-(vii) determines $\sigma = (c, a, b, e)$. (ix)-(xii) determines $\sigma = (a, c, b, e)$. (xiii)-(xvi) determines $\sigma = (c, b, a, e)$. 53
5.1 (i) testing independence $I(a, e, g)$ in a Bayesian network $\mathcal{B}$; (ii) the active part of $\mathcal{B}$; (iii) the relevant part of $\mathcal{B}$; (iv) the intersection of the active and relevant parts. 55
5.2 Testing $I(a, e, g)$ in the BN $\mathcal{B}$ in Figure 5.1 (i) with m-separation in (i)-(iii) and with refined m-separation in (i), (iv), and (v). 64
6.1 When testing $I(D, G, H)$ in Figure 6.1i, active paths between D and H with respect to G can only involve variables in the sub-DAG in Figure 6.1ii defined over $DGH \cup An(DGH)$. Darwiche's sub-DAG in Figure 6.1iii. 70
6.2 Consider v-structure node $C \in Y$ in $I(X, Y, Z)$. (6.2i) Approaching C from the top, say from A, renders B reachable. (6.2ii) Approaching C from the bottom, makes B unreachable. 72
7.1 A Bayesian network. 82
7.2 An AC for the Bayesian network in Figure 7.1 following (58). 83
7.3 A Sum-product network. 85
9.1 An SPN over four features X_0, X_1, X_2, and X_3. 98
9.2 SPN $\mathcal{S}_1$ learned from the dataset in Table 9.1 using g-test for chopping and k-means for slicing. 104
9.3 SPN $\mathcal{S}_2$ learned from the dataset in Table 9.1 using g-test for chopping and GMM for slicing. 105
9.4 SPN $\mathcal{S}_3$ learned from the dataset in Table 9.1 using MI for chopping and k-means for slicing. 105

9.5 SPN $\mathcal{S}_4$ learned from the dataset in Table 9.1 using MI for chopping and GMM for slicing. 106
10.1 Input, convolutional, and sum-pooling layers in a CNN representing input, sum, and product layers in an SPN. 117
10.2 A CSPN represents a valid SPN and is vectorized, but also can suffer from being shallow. 118
10.3 *Channel* (a)-(b) and *width* (c)-(d) augmentations. 120
10.4 A DCSPN, depicting a rich DAG structure of convolutional and sum-pooling layers, while still being a valid SPN. 121
10.5 Columns show original, DCSPN, DCGAN, and P&D. The first and second rows show left-completion and bottom-completions, respectively. Left and right pictures are from datasets Olivetti Face and Caltech Face, respectively. 123
10.6 DCSPNs performed well on a dataset with 65 images. Left-completion of the original image in (i) by DCSPN (ii), P&D (iii), and DCGAN (iv). 124
11.1 An SPN $\mathcal{S}$ over RVs $\mathbf{X} = \{X_1, X_2\}$. 127
11.2 Two-phase method for focused in learning. Let $\mathbf{x}$ be an instance from the dataset. Phase 1 detects underperforming variables by using an accuracy metric with a performance threshold p and clustering them in large groups as determined by a cluster-size threshold c. Phase 2 maximizes the marginal likelihood from Phase 1. 127
11.3 The MPE bottom-half completion of (i) yields a blocky texture in (ii). 131
11.4 Detecting an underperforming marginal involves (i) clustering underperforming variables (colour green), (ii) focusing on the large clusters by ignoring small clusters, as depicted in (iii). The underperforming marginal is given by the remaining clusters, as shown in (iv). 131

Chapter 1

Introduction

Probabilistic models are a formal mathematical description for a probability distribution. We can use a probabilistic model to perform reasoning in a problem domain involving random variables. This technique is especially useful when making decisions under uncertainty. A *probabilistic graphical model* (PGM) is the marriage of probabilistic models and graph theory. Here, a graph is used to represent conditional independencies holding in the probability distribution. *Bayesian networks* (BNs) [61] and *sum-product networks* (SPNs) [70] are PGMs that we investigate in this thesis. A problem domain can be modeled as a BN with a *directed acyclic graph* (DAG) and the strengths of relationships are quantified by *conditional probability tables* (CPTs). In an SPN, a problem domain is represented as a DAG containing three types of nodes: leaf distributions, sums, and products.

In this thesis, we make 7 contributions (Chapters 3-4 and Chapters 8-10) in BNs and SPNs by extending their problem domain modeling capabilities. These contributions are all published through 22 papers, which have been cited 43 times in total. In particular, our deep convolutional sum-product networks paper, published in AAAI 2019 [5], has already been cited 3 times so far this year. In Chapter 3 *Darwinian networks* (DNs) [7] are put forth to simplify working with BNs. A CPT $P(X|Y)$ is viewed as a population $p(X, Y)$ with combative traits X and docile traits Y. More generally, a BN is seen as a set of populations. In DNs, how populations adapt to the deletion of other populations corresponds precisely with testing independencies in BNs. Once abstract concepts like merge and replication are used to represent multiplication, division, and addition, it readily follows that DNs can also represent *variable elimination* (VE) [84], *arc-reversal* (AR) [58, 73], and *lazy propagation*

(LP) [50]. Good elimination orderings, which are of practical importance, can be computed in DNs. Besides providing a single platform for testing independencies, performing inference, and determining good elimination orderings, we show how DNs simplify *d-separation* [61], *m-separation* [46, 84], VE, AR, LP, and min-neighbours.

In Chapter 4, we show how *min-neighbours* (MN), *min-weight* (MW), *min-fill* (MF), and *weighted-min-fill* (WMF) heuristics can be represented in DNs. More importantly, we propose a new heuristic, called *potential energy* (PE), based on DNs themselves. Our analysis of PE shows that it can: (i) better scoring of a variable; (ii) better model the multiplication of the probability tables for the chosen variable; (iii) more clearly model the marginalization of the chosen variable; and (iv) maintain a one-to-one correspondence between the remaining variables and probability tables.

Relevant path separation (rp-separation) is introduced in Chapter 5 as a faster method for testing independencies in BNs. The salient feature of rp-separation is that it explores the *intersection* between the active and relevant parts of a BN. We introduce the notion of a *relevant* path and establish that *irrelevant* paths are either active paths that are doomed to become blocked or active paths that terminate before reaching a variable of interest. Rather than exploring all active paths, rp-separation displays impressive performance in practice by only exploring active paths that are relevant. In real-world or benchmark BNs, rp-separation is faster in 17 of 19 cases with an average time savings of 53%, culminating with being nearly twice as fast in the largest BN.

In Chapter 6, we propose Symmetric d-Separation as the first d-separation implementation that exploits the symmetry of probabilistic conditional independence. Consider random variables X, Y, and Z. Since $I(X,Y,Z)$ is equivalent to $I(Z,Y,X)$ by symmetry, the test can be answered by computing R_Z, the nodes reachable from Z along active paths with respect to Y, and then testing whether

$$R_Z \cap X = \emptyset. \tag{1.1}$$

It is important to realize that R_X and R_Z can contain different nodes and may have distinct cardinalities. Our analysis of reachability highlights that *v-structure* [44] nodes play an important role in this regard. More specifically, it may be better to approach observed v-structure nodes from the bottom. Approaching an observed v-structure node from a child blocks an active path, whereas an active path can continue

when approaching an observed v-structure node from a parent. Hence, a measure, called *depth*, is suggested to decide whether the search should run from start to goal or from goal to start. One salient feature is that depth can be computed during the pruning optimization step of finding the ancestor set of $X \cup Y \cup Z$. An empirical comparison is conducted against a clever implementation of d-separation suggested by [17], which we refer to as DARWICHE. The experimental results are promising in two aspects. The effectiveness increases with network size, as well as with the amount of observed evidence, culminating with an average time savings of 9% in the 9 largest BNs used in our experiments.

Two theoretical SPN contributions are given in Chapter 8. Our first contribution is a new scope definition that suits the AC interpretation of an SPN. This definition considers the fact that ACs are compiled from Bayesian networks. Thus, the random variables at the leaf nodes come from probabilities in Bayesian network conditional probability tables. The scope of the leaf nodes then are labeled according to the variables in the conditional probability tables. The scope of a sum node removes one random variable from the union of the scope of its children. This reflects the interpretation of sum nodes as being marginalization. The scope of a multiplication node is the union of its children's scope. We establish that the completeness property of SPN can be defined under the AC interpretation of scope. Unfortunately, consistency and decomposability can not be applied using this scope definition, meaning that the definition can not be used for verifying tractability of an SPN. Moreover, the proposed scope definition for ACs can be non-intuitive for leaf nodes under a mixture model interpretation. For example, the scope of a leaf node can consider variables that are not random variables for an SPN leaf node, yet appear in a conditional probability table from the Bayesian network. The second contribution of this chapter is a novel scope definition with two advantages. First, it can be used when defining all three SPN properties. Second, it addresses the counter-intuitive points in both the mixture model and AC interpretations. Here, we define the scope of sum nodes more intuitively from an AC point of view by treating them as marginalization. Hence, the scope of sum nodes removes one random variable from the union of the scope of its children. Moreover, we make the scope of leaf nodes more intuitive from a mixture model point of view by only considering variables that are random variables for the SPN leaf node. Since both mixture models and ACs can be compiled into SPNs,

the new scope definition improves the understanding of SPNs for different research communities, while still guaranteeing tractable inference during learning.

In Chapter 9, we perform an empirical study of LearnSPN. We consider g-test and mutual information for chopping and k-means and GMMs for slicing. Our experiments, conducted on 20 real-world datasets, suggest that the deepest SPNs tend to be learned when using mutual information for chopping and k-means for slicing. Second, our results show that the pair of g-test and GMM tend to yield the most accurate SPNs, especially on larger datasets. These results suggest that the particular combination of mutual information and k-means may suffer especially from *overfitting* [80]. Lastly, we examine the sparseness of the learned SPN. Our experiments show that the pair of g-test and GMM often yield SPNs with fewer edges. This knowledge is beneficial to SPN learning algorithms that penalize networks with more edges [48]. Our findings can have far reaching influence, since SPNs have been applied in image completion [25, 65, 70], computer vision [1], classification [30], and speech and language modeling [12, 67]. Thereby, this study extends the deep learning literature in both practical and theoretical directions.

Deep convolutional sum-product networks (DCSPNs) are introduced in Chapter 10. DCSPNs permit the convolutional and sum-pooling layers to form rich DAG structures by augmenting layer tensors under conditions that maintain *decomposability* and *completeness*. As a decomposable and complete SPN is a valid SPN, our main result is that DCSPNs are a larger subclass of CNNs that define valid SPNs. DCSPNs are a rigorous probabilistic model. As such, they can exploit probabilistic reasoning, including *marginal* inference and *most probable explanation* (MPE) inference. This allows an alternative method for learning DCSPNs using vectorized differentiable MPE. We show how to vectorize MPE using a mask algorithm and how it plays a role similar to the GANs generator. Image sampling is yet another application demonstrating the robustness of DCSPNs. This involves a minor modification to the mask algorithm. Our preliminary results on image sampling are promising, since the DCSPN sampled images exhibit variability. Experimental results on left- and bottom-completion like those in Table 10.1 show DCSPNs achieve state-of-the-art by building deeper structures using both vertical and horizontal sum-pooling windows, which leverage local structure in the image data in both directions. Applying a simple low pass filter as a post-processing smoothing operation lowers the *mean squared error* (MSE) score

from 910 to 802 for left-completion in Olivetti.

Chapter 2

Bayesian Networks

Let $U = \{v_1, v_2, \ldots, v_n\}$ be a finite set of variables, each with a finite domain, and V be the domain of U, that is the cartesian product of the domain of variables $v_1, v_2, \ldots, v_n$. Let $\mathcal{B}$ denote a *directed acyclic graph* (DAG) on U. A *directed path* from v_1 to v_k is a sequence $v_1, v_2, \ldots, v_k$ with arcs (v_i, v_{i+1}) in $\mathcal{B}$, $i = 1, 2, \ldots, k-1$. For each $v_i \in U$, the *ancestors* of v_i, denoted $An(v_i)$, are those variables having a directed path to v_i, while the *descendants* of v_i, denoted $De(v_i)$, are those variables to which v_i has a directed path. For a set $X \subseteq U$, we define $An(X)$ and $De(X)$ in the obvious way. The *children* $Ch(v_i)$ and *parents* $Pa(v_i)$ of v_i are those v_j such that $(v_i, v_j) \in \mathcal{B}$ and $(v_j, v_i) \in \mathcal{B}$, respectively. An *undirected path* in a DAG is a path ignoring directions. A *path* in an undirected graph is defined similarly. A singleton set $\{v\}$ may be written as v, $\{v_1, v_2, \ldots, v_n\}$ as $v_1 v_2 \cdots v_n$, and $X \cup Y$ as XY.

2.1 Modeling

d-Separation [61] tests independencies in DAGs and can be presented as follows [17]. Let X, Y, and Z be pairwise disjoint sets of variables in a DAG $\mathcal{B}$. We say X and Z are *d-separated* by Y, denoted $I_{\mathcal{B}}(X, Y, Z)$, if at least one valve on every undirected path between X and Z is closed. There are three kinds of valves v: (i) a *sequential* valve means v is a parent of one of its neighbours and a child of the other; (ii) a *divergent* valve is when v is a parent of both neighbours; and, (iii) a *convergent* valve is when v is a child of both neighbours. A valve v is either open or closed. A sequential or divergent valve is *closed*, if $v \in Y$. A convergent valve is *closed*, if $(v \cup De(v)) \cap Y = \emptyset$. For example, suppose $X = a$, $Y = c$, and $Z = f$ in DAG $\mathcal{B}$

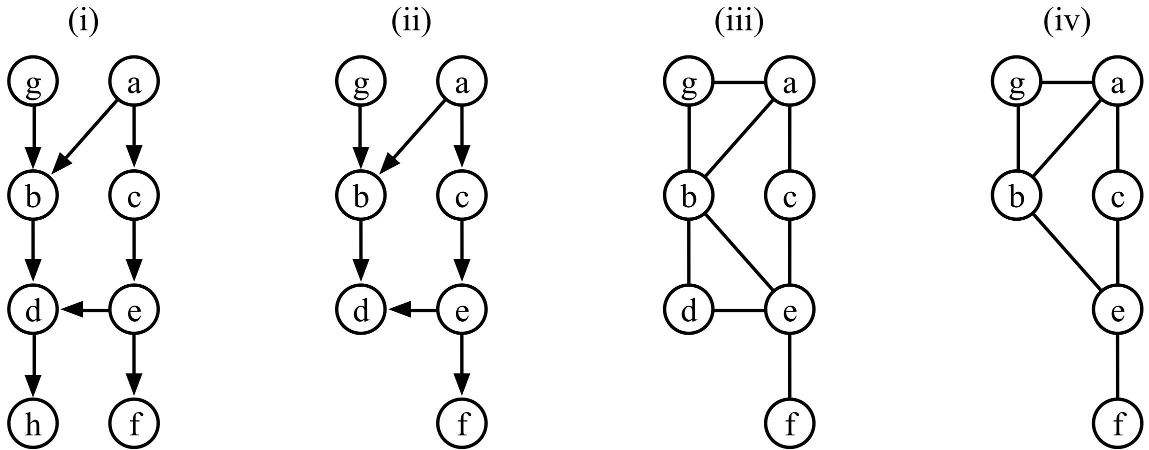


Figure 2.1: (i) A DAG $\mathcal{B}$. (ii) Sub-DAG $\mathcal{B}^s$. (iii) Moralization $\mathcal{B}^s_m$. (iv) $\mathcal{B}^s_m$ with d and its edges deleted.

depicted in Figure 2.1 (i). To test $I_B(a, c, f)$ there are two undirected paths from a to f. On the path (a, c), (c, e), (e, f), valve c is closed, since c is a sequential valve and $c \in Y$. Valve d is closed on the other path, since d is a convergent valve and $\{d, h\} \cap Y = \emptyset$. As both paths from a to f have a closed valve, $I_{\mathcal{B}}(a, c, f)$ holds.

m-Separation [46, 84] is another method for testing independencies in DAGs, and is equivalent to d-separation. Let X, Y, and Z be pairwise disjoint sets of variables in a DAG $\mathcal{B}$. Then m-separation tests $I_{\mathcal{B}}(X, Y, Z)$ with four steps: (i) construct the sub-DAG of $\mathcal{B}$ induced by $XYZ \cup An(XYZ)$, yielding $\mathcal{B}^s$; (ii) construct the *moral graph* [47] of $\mathcal{B}^s$, denoted $\mathcal{B}^s_m$, by adding an undirected edge between each pair of parents of a common child and then dropping directionality; (iii) delete Y and its incident edges; and (iv) if there exists a path from any variable in X to any variable in Z, then $I_{\mathcal{B}}(X, Y, Z)$ does not hold; otherwise, $I_B(X, Y, Z)$ holds. For example, in Figure 2.1, to test $I_{\mathcal{B}}(a, d, f)$ in $\mathcal{B}$ of (i), the sub-DAG $\mathcal{B}^s$ is in (ii). $\mathcal{B}^s_m$ is shown in (iii). Removing d and incident edges gives (iv). Since there exists a path from a to f, $I_{\mathcal{B}}(a, d, f)$ does not hold.

A *potential* on V is a function ϕ such that $\phi(v) \geq 0$ for each $v \in V$, and at least one $\phi(v) > 0$. A *uniform* potential on V is a function 1 that sets $1(v) = 1/k$, where $v \in V$, $k = |V|$ and $|\cdot|$ denotes set cardinality. Henceforth, we say ϕ is on U instead of V. A *joint probability distribution* is a potential P on U, denoted $P(U)$, that sums to one. For disjoint $X, Y \subseteq U$, a *conditional probability table* (CPT) $P(X|Y)$ is a potential over XY that sums to one for each value y of Y.

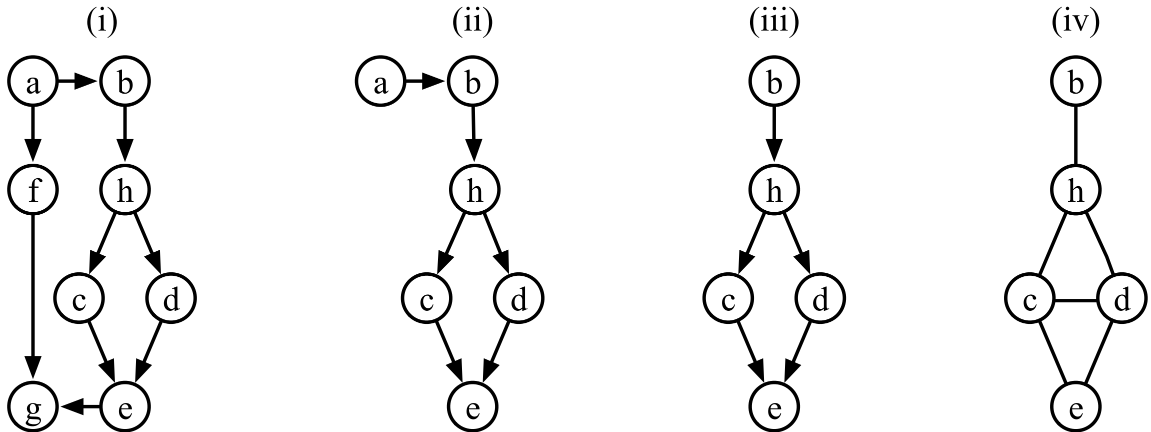


Figure 2.2: [84] Given query $P(e|b=0)$ posed to BN $\mathcal{B}$ in (i), pruning barren variables g and f in (ii) and independent by evidence variable a in (iii). (iv) is $\mathcal{B}^s_m$.

A *Bayesian network* (BN) [61] is a DAG $\mathcal{B}$ on U together with CPTs $P(v_1|Pa(v_1))$, $P(v_2|Pa(v_2))$, $\ldots$, $P(v_n|Pa(v_n))$. For example, Figure 2.2 (i) shows a BN, where CPTs $P(a), P(b|a), \ldots, P(g|e,f)$ are not shown.

We call $\mathcal{B}$ a BN, if no confusion arises. The product of the CPTs for $\mathcal{B}$ on U is a joint probability distribution $P(U)$. The *conditional independence* [61] of X and Z given Y holding in $P(U)$ is denoted $I(X,Y,Z)$. It is known that if $I_{\mathcal{B}}(X,Y,Z)$ holds by d-separation (or m-separation) in $\mathcal{B}$, then $I(X,Y,Z)$ holds in $P(U)$.

2.2 Inference

Variable elimination (VE) [84] computes $P(X|Y=y)$ from a BN $\mathcal{B}$ as follows: (i) all barren variables are removed recursively, where v is *barren* [84], if $Ch(v)=\emptyset$ and $v \notin XY$; (ii) all independent by evidence variables are removed, giving $\mathcal{B}^s$, where v is an *independent by evidence* variable, if $I(v,Y,X)$ holds in $\mathcal{B}$ by m-separation; (iii) build a uniform distribution $1(v)$ for any root of $\mathcal{B}^s$ that is not a root of $\mathcal{B}$; (iv) set Y to $Y=y$ in the CPTs of $\mathcal{B}^s$; (v) determine an elimination ordering σ from the moral graph $\mathcal{B}^s_m$; (vi) following σ, eliminate variable v by multiplying together all potentials involving v, and then summing v out of the product; and, (vii) multiply together all remaining potentials and normalize to obtain $P(X|Y=y)$. For example [84], in Figure 2.2, given $P(e|b=0)$ and BN $\mathcal{B}$ in (i), g and f are barren (ii) and a is independent by evidence (iii) for steps (i) and (ii). In steps (iii) and (iv), VE builds

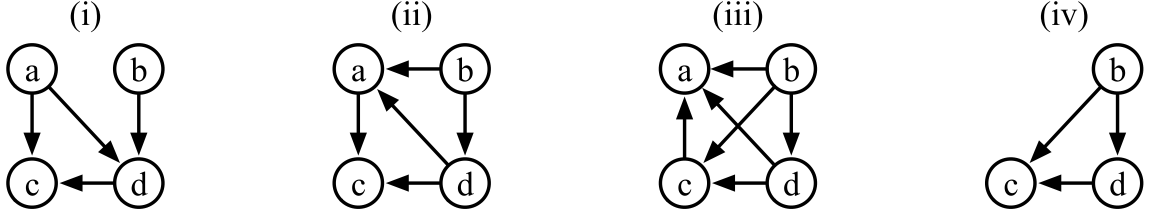


Figure 2.3: Applying AR in (2.6)-(2.10) to eliminate a.

$1(b)$ and updates $P(h|b)$ as $P(h|b=0)$. Step (v) can determine $\sigma = (c, d, h)$ from $\mathcal{B}^s_m$ shown in (iv). Step (vi) computes (step (vii) is discussed later):

$$P(c, e|d, h) = P(c|h) \cdot P(e|c, d), \tag{2.2}$$

$$P(e|d, h) = \sum_{c} P(c, e|d, h), \tag{2.3}$$

$$P(e|h) = \sum_{d} P(d|h) \cdot P(e|d, h), \tag{2.4}$$

$$P(e|b = 0) = \sum_{h} P(h|b = 0) \cdot P(e|h). \tag{2.5}$$

Arc-reversal (AR) [58, 73], unlike VE's step (vi), eliminates variable v_i by reversing the arc (v_i, v_j) between v_i and each child v_j in $Ch(v_i)$. In order to reverse arc (v_i, v_j), $De(v_i) \cap An(v_j) = \emptyset$ must hold. For example, to eliminate variable a in Figure 2.3 (i), AR must reverse (a,d) in (ii), and then (a,c) in (iii), giving (iv), as:

$$P(a, d|b) = P(a) \cdot P(d|a, b), \tag{2.6}$$

$$P(d|b) = \sum_{a} P(a, d|b), \tag{2.7}$$

$$P(a|b, d) = P(a, d|b) \; / \; P(d|b), \tag{2.8}$$

$$P(a, c|b, d) = P(a|b, d) \cdot P(c|a, d), \tag{2.9}$$

$$P(c|b, d) = \sum_{a} P(a, c|b, d). \tag{2.10}$$

Lazy propagation (LP) [50] computes the posterior probabilities for each non-evidence variable using a join tree, denoted $\mathcal{T}$, which is constructed from a BN $\mathcal{B}$. A *join tree* [75] is a tree with sets of variables as nodes, and with the property that any variable in two nodes is also in any node on the path between the two. Evidence on a variable v decreases the domain of each potential with v in its domain

by v [50]. LP consists of two phases of message passing, an inward phase from the leaf nodes to a chosen root node, and then an outward phase from the root back out to the leaves. Before a node can send its message to a neighbour, it must first receive messages from all its other neighbours. A node N computes its messages to a particular neighbour N' as follows: (i) d-separation is applied on $\mathcal{B}$ to remove irrelevant potentials; (ii) potentials for barren variables are removed; and (iii) any remaining variables in $N - N'$ are marginalized away similar to VE step (vi). The root node can compute posterior probabilities of its variables when the inward phase terminates. All nodes can compute posterior probabilities of their variables when the outward phase completes.

Consider join tree $\mathcal{T}$ in Figure 2.4 with BN $\mathcal{B}$ understood from the CPTs. Let $h = 0$ and node $\{i, j, k\}$ be the root. Node $\{d, f, g\}$, for instance, must wait to send its message to node $\{b, g, h, i, j\}$ until has received messages from nodes $\{c, d\}$ and $\{e, f\}$. It computes message $P(g)$ in step (iii) by marginalizing d and f from $P(d)$, $P(f)$ and $P(g|d, f)$. Upon receiving $P(g)$, and $P(b)$ from node $\{a, b\}$, $\{b, g, h, i, j\}$ computes its message to $\{i, j, k\}$ as follows. In (i), $I(bg, h, ijk)$ holds in $\mathcal{B}$ by d-separation, meaning that $P(b)$, $P(g)$ and $P(h = 0|b, g)$ can be safely removed, and $P(i|h = 0)$ and $P(j|h = 0)$ are passed to $\{i, j, k\}$. At node $\{k, l\}$, l is barren in step (ii). The outward phase follows.

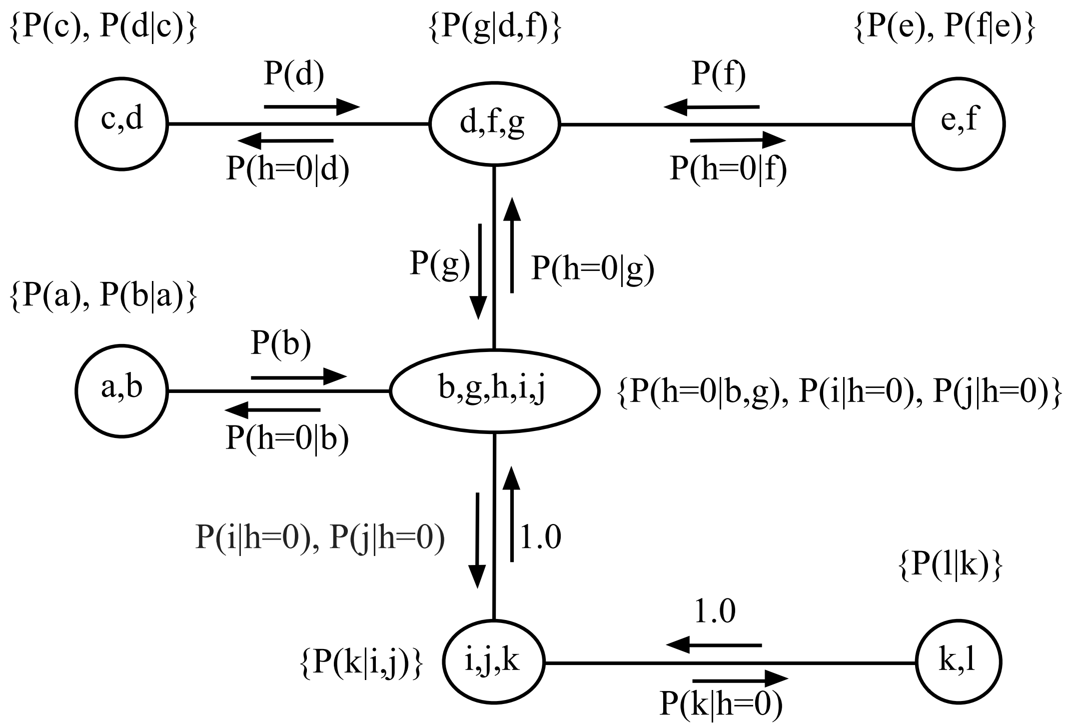


Figure 2.4: LP message passing with evidence $h = 0$.

Chapter 3

Darwinian Networks

Many different platforms, techniques and concepts can be employed while modeling and reasoning with *Bayesian networks* (BNs) [61]. A problem domain is modeled initially as a *directed acyclic graph* (DAG), denoted $\mathcal{B}$, and the strengths of relationships are quantified by *conditional probability tables* (CPTs). Independencies are tested in $\mathcal{B}$ using *d-separation* [61] or *m-separation* [46, 84]. Reasoning with a BN can be done using $\mathcal{B}$, including inference algorithms such as *variable elimination* (VE) [84] and *arc-reversal* (AR) [58, 73], or with a secondary structure called a join tree and denoted $\mathcal{T}$, as in *lazy propagation* (LP) [50]. Considering exact inference in discrete BNs, a common task, called belief update, is to compute posterior probabilities given evidence (observed values of variables). Before performing number crunching, two kinds of variables can safely be removed, namely, *barren* variables [84] and what we call *independent given evidence* variables [84, 50]. LP and VE treat the removal of these variables as separate steps. Furthermore, LP and VE involve multiple platforms. LP conducts inference on $\mathcal{T}$ and test independencies in $\mathcal{B}$. VE first prunes barren variables from a DAG $\mathcal{B}$, giving a sub-DAG $\mathcal{B}^s$, and then prunes independent by evidence variables from the *moralization* [47] of $\mathcal{B}^s$, denoted $\mathcal{B}^s_m$. VE can also use $\mathcal{B}^s_m$ to determine an *elimination ordering* [42] using the *min-neighbours* [44] heuristic. By adapting a few well-known concepts in biology [11, 15, 18, 20, 21], all of the above can be unified into one platform to be denoted $\mathcal{D}$.

Darwinian networks (DNs) [7] are put forth to simplify working with BNs. A CPT $P(X|Y)$ is viewed as a population $p(X, Y)$ with combative traits X and docile traits Y. More generally, a BN is seen as a set of populations. In DNs, how populations adapt to the deletion of other populations corresponds precisely with testing

independencies in BNs. Once abstract concepts like merge and replication are used to represent multiplication, division, and addition, it readily follows that DNs can also represent VE, AR, and LP. Good elimination orderings, which are of practical importance, can be computed in DNs. Besides providing a single platform for testing independencies, performing inference, and determining good elimination orderings, we show how DNs simplify d-separation, m-separation, VE, AR, LP, and min-neighbours.

The theoretical foundation of DNs is provided in full. We show how DNs can represent LP. It is demonstrated how DNs can represent *chain graph models* (CGMs) [44], which are an extension of BNs, as well relational databases, a non-Bayesian framework for data management. Also, we highlight the robustness of DNs by reviewing how DNs can determine good elimination orderings. Practical benefits of DNs include faster algorithms for inference and modeling.

Related works include [22, 23, 32]. Gogate and Domingos [32] focus on unifying approximate inference, whereas our focus is exact. Dechter [22, 23] has already given *bucket elimination* as an elegant framework for unifying inference. We extend this line of investigation to topics not yet unified, including testing independencies, determining good elimination orderings, and representing extended and non-Bayesian frameworks.

3.1 Darwinian Networks

We provide a gentle introduction to the main concepts of Darwinian networks, and then formally discuss the central tasks of adaptation and evolution.

3.1.1 Introduction

We motivate the introduction of Darwinian networks by underscoring the fuzziness in representing BN CPTs with undirected graphs.

Example 1. *Consider the undirected edge* (a, b) *between two variables* a *and* b*. The*

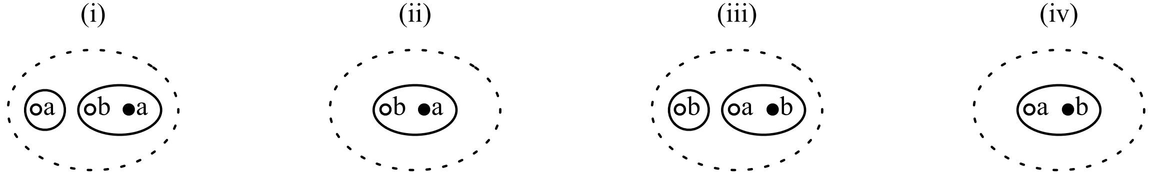


Figure 3.1: Unique graphical representations of (3.11)-(3.14), respectively.

set of CPTs represented by this edge is not unique and possibly could include:

$$\{P(a), P(b|a)\}, \tag{3.11}$$

$$\{P(b|a)\}, \tag{3.12}$$

$$\{P(b), P(a|b)\}, \tag{3.13}$$

and

$$\{P(a|b)\}. \tag{3.14}$$

To avoid the ambiguity highlighted in Example 1, we sought a new graphical representation of CPTs that would maintain a one-to-one correspondence between the graph and the set of CPTs.

Given a CPT $P(X|Y)$, each variable in X is represented by a white circle. Each variable in Y is represented by a black circle. A closed curve encloses all circles for X and Y. A set of CPTs is enclosed in a dashed closed curve.

Example 2. *The four sets of CPTs in (3.11)-(3.14) are represented in Figure 3.1 (i)-(iv), respectively.*

Observe that there is a clear one-to-one correspondence between (3.11)-(3.14) and Figure 3.1 (i)-(iv).

We also seek to clearly model operations on CPTs. For example, consider the multiplication of CPTs $P(a)$ and $P(b|a)$, yielding CPT $P(a, b)$:

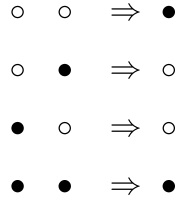
$$P(a, b) \;=\; P(a) \cdot P(b|a). \tag{3.15}$$

Figure 3.2: (i) Graphically representing the multiplication in (3.15). (ii) Graphically representing the division in (3.16).

Or consider the division of CPT $P(a, b)$ by CPT $P(b)$, giving CPT $P(a|b)$:

$$P(a|b) = P(a, b)/P(b). \tag{3.16}$$

The multiplication in (3.15) and the division in (3.16) can be readily modeled in Figure 3.2 based upon the following colour combinations:



Marginalization is another important operation needed to be modeled. For example, consider the marginalization of variable a from the CPT $P(a, b|c)$:

$$P(b|c) = \sum_{a} P(a, b|c). \tag{3.17}$$

The marginalization in (3.17) can be conveniently represented in Figure 3.3. Here, two steps are used. First, the representation of the CPT $P(a, b|c)$ in Figure 3.3 (i) makes an imperfect copy of itself, as depicted in Figure 7 (ii). Second, the representation of the original CPT $P(a, b|c)$ is removed, as illustrated in Figure 3.3 (iii).

Upon reflection, Figures 3.1-3.3 take on a biological feel. Consequently, these concepts are named according to biological terms that a layperson may be familiar with. The graphical representation of a CPT is called a *population*. The circles within a population are called *traits*. A white trait is *combative*, while a black trait is *docile*.

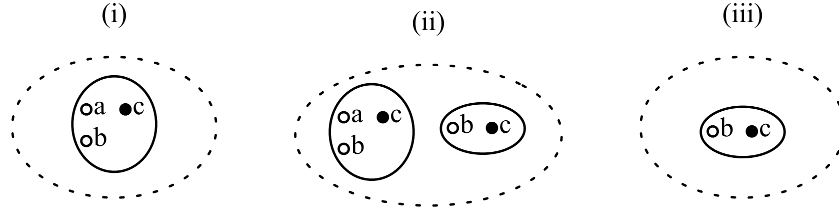


Figure 3.3: Graphically representing the marginalization of variable a in (3.17).

Note that a combative trait reflects a variable on the left side of the conditioning bar in a CPT, whereas a docile trait reflects a variable on the right side of the conditioning bar in a CPT. By *merge*, we mean the process of combining two populations. Thus, merge graphically represents multiplication and division of CPTs. Lastly, the process of making a copy of a population is called *replication*, and the removal of populations is done by *natural selection*. We formalize these ideas in the next subsection.

3.1.2 Adaptation

Darwinian networks are a graphical model. The most basic unit in a Darwinian network is a trait.

Definition 3.1. *A trait, denoted t, is a named element.*

A trait is depicted by a circle along with its name t. A named circle can be either black or white. A white circle depicts a *combative* trait, denoted t_c. A black circle depicts a *docile* trait, denoted t_d.

Definition 3.2. *A* population*, denoted $p(C, D)$, contains a non-empty set CD of traits, where C and D are disjoint, C is exclusively combative, and D is exclusively docile.*

A population is depicted by a closed curve around its traits. For example, Figure 3.4 (i) shows eight populations, including $p(b, ag)$, short for $p(\{b\}, \{a, g\})$, illustrated with a closed curve around the (white) combative trait b and two (black) docile traits a and g.

Definition 3.3. *A* Darwinian network *(DN), denoted $\mathcal{D}$, is a finite, multiset of populations.*

A DN $\mathcal{D}$ is depicted by a dashed closed curve around its populations. For example, Figure 3.4 (i) depicts a DN $\mathcal{D} = \{p(a), p(b, ag), p(c, a), p(d, be), p(e, c), p(f, e), p(g), p(h, d)\}$, where $p(C, \emptyset)$ is succinctly written $p(C)$.

All combative traits in a given DN $\mathcal{D}$ are defined as $T_c(\mathcal{D}) = \{t_c \mid t_c \in C$, for at least one $p(C, D) \in \mathcal{D}\}$. All docile traits in $\mathcal{D}$, denoted $T_d(\mathcal{D})$, are defined similarly. For example, considering DN $\mathcal{D}$ in Figure 3.4 (i), then $T_c(\mathcal{D}) = \{a_c, b_c, c_c, d_c, e_c, f_c, g_c, h_c\}$. In addition, $T_d(\mathcal{D}) = \{a_d, b_d, c_d, d_d, e_d, g_d\}$.

Populations are classified based upon characteristics of their traits. For adaptation, barren populations need only to be classified. Later, for evolution, we will extend the classification.

Given two DNs $\mathcal{D}$ and $\mathcal{D}'$, let t_c be a trait in $T_c(\mathcal{D})$. Trait t_c is *strong*, if $t_c \in T_c(\mathcal{D}')$; otherwise, t_c is *weak*. Trait t_c is *relict*, if $t_d \notin T_d(\mathcal{D})$. The notions of strong, weak, and relict are defined analogously for t_d.

Given DNs $\mathcal{D}$ and $\mathcal{D}'$, a population $p(t_c, D)$ is *barren* in $\mathcal{D}$, if t_c is relict, and both t_c and t_d are weak. For example, consider the DNs $\mathcal{D}$ in Figure 3.4 (i) and $\mathcal{D}' = p(acf)$ in Figure 3.4 (v). Population $p(d, be)$ is not barren, since d_c is not relict. Population $p(h, d)$ is barren, since h_c is relict, and h_c and h_d are weak.

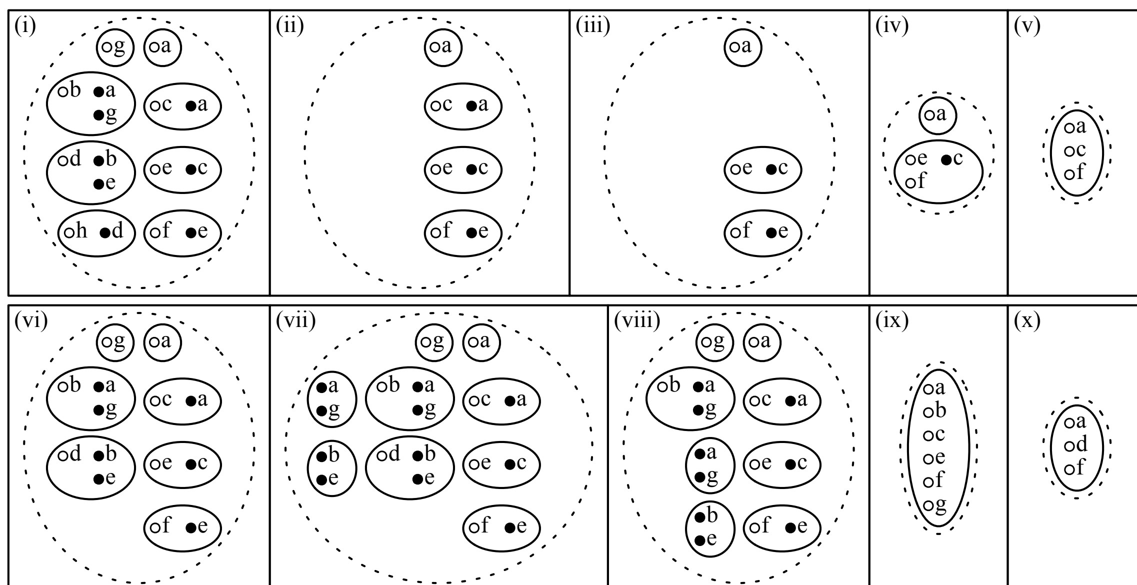


Figure 3.4: Testing adaptation twice in the DN $\mathcal{D}$ in (i).

In adaptation, *natural selection* removes recursively all barren populations from a DN $\mathcal{D}$ with respect to another DN $\mathcal{D}'$.

Example 3. Referring to Figure 3.4, let us apply natural selection on the DN $\mathcal{D}$ in (i) with respect to DN $\mathcal{D}'$ in (v). First, barren population $p(h,d)$ is removed. Population $p(d,be)$ now is barren, since d_c is relict, and d_c and d_d are weak. Natural selection removes $p(d,be)$ and, in turn, $p(b,ag)$ and $p(g)$, giving (ii).

Docilization of a DN $\mathcal{D}$ adds $p(\emptyset,D)$ to $\mathcal{D}$, for every population $p(C,D)$ in $\mathcal{D}$ with $|D|>1$. For example, the docilization of Figure 3.4 (ii) is itself, while the docilization of Figure 3.4 (vi) adds populations $p(\emptyset,ag)$ and $p(\emptyset,be)$, giving Figure 3.4 (vii).

To *delete* a population $p(C,D)$ from a DN $\mathcal{D}$ is to remove all occurrences of it from $\mathcal{D}$. For example, the deletion of $p(c,a)$ from Figure 3.4 (ii) gives Figure 3.4 (iii).

Two populations *merge* together as follows: for each trait t appearing in either population, if t is combative in exactly one of the two populations, then t is combative in the merged population; otherwise, t is docile. For example, the merge of populations $p(e,c)$ and $p(f,e)$ in Figure 3.4 (iii) is population $p(ef,c)$ in Figure 3.4 (iv).

Let $\mathcal{P}_X$, $\mathcal{P}_Y$, and $\mathcal{P}_Z$ be pairwise disjoint subsets of populations in a DN $\mathcal{D}$ and let DN $\mathcal{D}'=p(C)$, where $C=T_c(\mathcal{P}_X\mathcal{P}_Y\mathcal{P}_Z)$. We test the *adaptation* of $\mathcal{P}_X$ and $\mathcal{P}_Z$ given $\mathcal{P}_Y$, denoted $A(\mathcal{P}_X,\mathcal{P}_Y,\mathcal{P}_Z)$, in $\mathcal{D}$ with four simple steps: (i) let natural selection act on $\mathcal{D}$ with respect to $\mathcal{D}'$, giving $\mathcal{D}^s$; (ii) construct the docilization of $\mathcal{D}^s$, giving $\mathcal{D}^s_m$; (iii) delete $p(C,D)$ from $\mathcal{D}^s_m$, for each $p(C,D)$ in $\mathcal{P}_Y$; and, (iv) after recursively merging populations sharing t_c and t_d, if there exists a population containing both a combative trait in $T_c(\mathcal{P}_X)$ and a combative trait in $T_c(\mathcal{P}_Z)$, then $A(\mathcal{P}_X,\mathcal{P}_Y,\mathcal{P}_Z)$ fails; otherwise, $A(\mathcal{P}_X,\mathcal{P}_Y,\mathcal{P}_Z)$ succeeds.

Example 4. Let us test $A(p(a),p(c,a),p(f,e))$ in the DN $\mathcal{D}$ of Figure 3.4 (i), where $\mathcal{P}_X=p(a)$, $\mathcal{P}_Y=p(c,a)$, and $\mathcal{P}_Z=p(f,e)$. As $T_c(\{p(a),p(c,a),p(f,e)\})=\{a_c,c_c,f_c\}$, we obtain the DN $\mathcal{D}'=p(acf)$ in Figure 3.4 (v). In step (i), natural selection gives $\mathcal{D}^s$ in Figure 3.4 (ii). In step (ii), docilization of $\mathcal{D}^s$ gives $\mathcal{D}^s_m$ in Figure 3.4 (ii). In step (iii), the deletion of $p(c,a)$ from $\mathcal{D}^s_m$ gives Figure 3.4 (iii). Recursively merging populations in step (iv) yields Figure 3.4 (iv). As no population in Figure 3.4 (iv) contains a_c in $T_c(p(a))$ and f_c in $T_c(p(f,e))$, $A(p(a),p(c,a),p(f,e))$ succeeds.

Example 5. Let us now test $A(p(a), p(d, be), p(f, e))$ in the DN $\mathcal{D}$ of Figure 3.4 (i). In this case, DN $\mathcal{D}^{'} = p(adf)$ is shown in Figure 3.4 (x). In step (i), natural selection removes barren population $p(h, d)$ as shown in Figure 3.4 (vi). In step (ii), docilization of Figure 3.4 (vi) gives Figure 3.4 (vii). In step (iii), $p(d, be)$ is deleted as depicted in Figure 3.4 (viii). Recursively merging populations in step (iv) yields Figure 3.4 (ix). By definition, $A(p(a), p(d, be), p(f, e))$ fails, since the population in Figure 3.4 (ix) contains a_c and f_c.

3.1.3 Evolution

As promised, population classification is extended.

Let $\mathcal{P}_Y = \{p(t_c, D) \mid p(t_c, D) \in \mathcal{D}$ and $t_d \in D^{'}\}$ and $\mathcal{P}_Z = \{p(t_c, D) \mid p(t_c, D) \in \mathcal{D}$ and $t_c \in C^{'}\}$, given DNs $\mathcal{D}$ and $\mathcal{D}^{'} = p(C^{'}, D^{'})$. In $\mathcal{D}$, $p(t_c, D)$ is *independent*, if $A(p(t_c, D), \mathcal{P}_Y, \mathcal{P}_Z)$ succeeds, and is *evident*, if t_d is strong, and D is all relict. Population $p(C, D)$ in a DN $\mathcal{D}$ is *spent*, if there exists $p(C^{'}, D)$ in $\mathcal{D}$ such that $C^{'} \subset C$ and $C - C^{'}$ is all relict. In Figure 3.5, with $\mathcal{D}$ in (ii) and $D^{'} = p(e, b)$ in (xiii), $p(a)$ is independent as $A(p(a), p(b, a), p(e, cd))$ succeeds, where $\mathcal{P}_Y = p(b, a)$ and $\mathcal{P}_Z = p(e, cd)$. In $\mathcal{D}$ of (iii) and $D^{'}$ of (xiii), $p(b, a)$ is evident as b_d is strong, and a_d is relict. In $\mathcal{D}$ of (vi), $p(ce, dh)$ is spent as $p(e, dh)$ is in $\mathcal{D}$ and c_c is relict.

New populations can be created in a DN as follows. *Replication* of a population $p(C, D)$ gives $p(C, D)$, as well as any set of populations $p(C^{'}, D)$, where $C^{'} \subset C$. For instance, replication of $p(ce, dh)$ in Figure 3.5 (v) can yield $p(ce, dh)$ and $p(e, dh)$ in Figure 3.5 (vi).

The *evolution* of a DN $\mathcal{D}$ into a DN $\mathcal{D}^{'}$ occurs by natural selection removing recursively all barren, independent, and spent populations, merging existing populations, and replicating to form new populations.

Example 6. In Figure 3.5, consider one explanation of the evolution of $\mathcal{D}$ in (i) into $\mathcal{D}^{'} = p(e, b)$ in (xiii). Natural selection removes barren populations $p(g, ef)$ and $p(f, a)$, yielding (ii). Next, natural selection removes independent population $p(a)$, giving (iii), and evident population $p(b, a)$, yielding (iv). Then, $p(c, h)$ and $p(e, cd)$ merge to form $p(ce, dh)$ in (v). Replication gives (vi). The rest of the example involves natural selection (vii), merge (viii), replication (ix), natural selection (x), merge (xi), replication (xii), and natural selection (xiii), leaving $\mathcal{D}^{'}$ with population $p(e, b)$.

Figure 3.5: Graphically representing VE's computation.

3.2 Testing independencies

Testing independencies in BNs can be seen as testing adaptation in DNs. d-Separation can use specialized terminology not referenced in inference such as open sequential valves and closed divergent valves. In contrast, no specialized concepts are used in adaptation. And whereas m-separation requires DAGs, sub-DAGs, and moral graphs, adaptation uses but one platform.

$\mathcal{D} = \{p(v, Pa(v)) \mid P(v|Pa(v))$ is in $\mathcal{B}\}$ is the DN for a given BN $\mathcal{B}$. Conversely, the *directed graph* (or simply *graph*) $\mathcal{G}(\mathcal{D})$ of a DN $\mathcal{D}$ has variables $T_c(\mathcal{D})$ and arcs $\{(v_i, v_j) \mid p(C, D) \in \mathcal{D}$ and $v_i \in D$ and $v_j \in C\}$. The *undirected graph* $U(\mathcal{D})$ of a DN $\mathcal{D}$ has variables $T_c(\mathcal{D})$ and edges $\{(v_i, v_j) \mid p(C, D) \in \mathcal{D}$ and $v_i, v_j \in CD\}$.

Lemma 1. *Every BN $\mathcal{B}$ can be represented as a DN $\mathcal{D}$.*

Proof. Let $\mathcal{D} = \{p(v, Pa(v))|P(v|Pa(v))$ is in $\mathcal{B}\}$. By definition, $\mathcal{D}$ is a DN. □

Example 7. *Consider the BN $\mathcal{B}$ in Figure 2.2 (i). As Lemma 1 guarantees, $\mathcal{B}$*

can be represented as the DN $\mathcal{D}$ in Figure 3.5 (i), where the CPT $P(g|e,f)$ in $\mathcal{B}$ is represented as population $p(g,ef)$ in $\mathcal{D}$.

Lemma 2. *Let $\mathcal{D}$ be the DN for a BN $\mathcal{B}$. Then the graph of $\mathcal{D}$ is $\mathcal{B}$, that is, $\mathcal{G}(\mathcal{D}) = \mathcal{B}$.*

Proof. Let U be the set of variables in $\mathcal{B}$. Each variable $v \in U$ has exactly one CPT $P(v|Pa(v))$ in $\mathcal{B}$. Now, by construction, $\mathcal{D}$ contains the population $p(v, Pa(v))$, for each $v \in U$. Therefore, the combative traits of $\mathcal{D}$ are exactly the variables in U, i.e., $T_c(\mathcal{D}) = U$.

Consider any arc (v_i, v_j) in $\mathcal{B}$. By definition, $v_i \in Pa(v_j)$. By construction, the population corresponding to the CPT $P(v_j|Pa(v_j))$ is $p(v_j, Pa(v_j))$. Thus, v_i is a docile trait in $p(v_j, Pa(v_j))$. Hence, (v_i, v_j) is an arc in $\mathcal{G}(\mathcal{D})$. A similar argument shows that every arc (v_i, v_j) in $\mathcal{G}(\mathcal{D})$ is an arc in $\mathcal{B}$. Thus, the arcs of $\mathcal{B}$ are exactly the arcs of $\mathcal{G}(\mathcal{D})$. □

Example 8. *Consider the DN $\mathcal{D}$ in Figure 3.5 (ii). The graph $\mathcal{G}(\mathcal{D})$ is $\mathcal{B}$ in Figure 2.2 (ii).*

Let $\mathcal{D}$ be the DN for a BN $\mathcal{B}$ on U. The populations for $W \subseteq U$, denoted $\mathcal{P}_W$, are $\mathcal{P}_W = \{p(C,D) \mid p(C,D) \in \mathcal{D} \text{ and } C \subseteq W\}$. Thus, given pairwise disjoint subsets X, Y, and Z in $\mathcal{B}$, it is necessarily the case that $\mathcal{P}_X$, $\mathcal{P}_Y$, and $\mathcal{P}_Z$ are pairwise disjoint populations in $\mathcal{D}$.

Lemma 3. *Let $\mathcal{B}^s$ be the sub-DAG constructed from a BN $\mathcal{B}$ in step (i) of testing the independence $I(X,Y,Z)$ using m-separation. Then $\mathcal{B}^s = \mathcal{G}(\mathcal{D}^s)$, where $\mathcal{D}^s$ is the DN constructed in step (i) of testing $A(\mathcal{P}_X, \mathcal{P}_Y, \mathcal{P}_Z)$ in the DN $\mathcal{D}$ for $\mathcal{B}$.*

Proof. If $U = (XYZ \cup An(XYZ))$, then $\mathcal{B}^s = \mathcal{B}$ and $\mathcal{D}^s = \mathcal{D}$. The claim follows from Lemma 2. Thus, let $W = U - (XYZ \cup An(XYZ))$ be non-empty. Since $\mathcal{B}$ is a DAG, there necessarily exists a variable $v_j \in W$ that is a leaf in $\mathcal{B}$. Thus, v_j will be pruned in step (i) of m-separation. Now consider the population $p(v_j, Pa(v_j))$ in $\mathcal{D}$ corresponding to v_j. Let us denote v_j by t. Trait t does not appear in any other population in $\mathcal{D}$ because variable v_j only appears in the CPT $P(v_j|Pa(v_j))$. Hence, t is relict. Moreover, both t_c and t_d do not appear in the DN $\mathcal{D}' = \{p(XYZ)\}$, since variable $v_j \notin XYZ$. By definition, t_c and t_d are weak. Then, by definition, population $p(v_j, Pa(v_j))$ is barren. Natural selection will then remove it from $\mathcal{D}$.

The removal of variable v_j and its incident edges from $\mathcal{B}$ necessarily yields a sub-DAG $\mathcal{B}'$ on $U - v_j$. If $W - v_j$ is non-empty, there must exist a variable in $W - v_j$ that is a leaf in $\mathcal{B}'$. A similar argument as above holds to remove recursively all variables in W and the corresponding populations from $\mathcal{D}$. Thus, the populations in $\mathcal{D}^s$ correspond exactly to the CPTs defining $\mathcal{B}^s$. By Lemma 2, $G(\mathcal{D}^s) = \mathcal{B}^s$. □

Example 9. *Step (i) of m-separation when testing $I_{\mathcal{B}}(a, d, f)$ in the BN $\mathcal{B}$ of Figure 2.1 (i) constructs the sub-DAG $\mathcal{B}^s$ in Figure 2.1 (ii). On the other hand, step (i) of adaptation when testing $A(p(a), p(d, be), p(f, e))$ in the DN $\mathcal{D}$ in Figure 3.4 (i) constructs the DN $\mathcal{D}^s$ in Figure 3.4 (vi). As guaranteed by Lemma 3, $\mathcal{B}^s = \mathcal{G}(\mathcal{D}^s)$.*

Lemma 4. *$\mathcal{B}^s_m = U(\mathcal{D}^s_m)$, where $\mathcal{B}^s_m$ is the moralization of $\mathcal{B}^s$ in Lemma 3 and $\mathcal{D}^s_m$ is the docilization of $\mathcal{D}^s$ in Lemma 3.*

Proof. Moralization of sub-DAG adds an undirected edge between each pair of parents of a common child and then drops directionality. Let v_i and v_j be parents of a common child v_k in $\mathcal{B}^s_m$. Then, undirected edge (v_i, v_j) appears in the moralization $\mathcal{B}^s$. Now, $v_i, v_j \in Pa(v_k)$. Thus, v_i and v_j are docile traits in the population $p(v_k, Pa(v_k))$ in $\mathcal{D}^s$ corresponding to the CPT $P(v_k|Pa(v_k))$. By definition, the docilization $\mathcal{D}^s_m$ of $\mathcal{D}^s$ includes population $p(\emptyset, Pa(v_k))$, since $|Pa(v_k)| > 1$. In turn, this means that the undirected edge (v_i, v_j) appears in $U(\mathcal{D}^s_m)$, since an edge (v_i, v_j) is added when $p(C, D) \in \mathcal{D}$ and $v_i, v_j \in CD$. A similar argument shows that every undirected edge $(v_i, v_j) \in U(\mathcal{D}^s_m)$ also is an undirected edge in $\mathcal{B}^s_m$. Therefore, $\mathcal{B}^s_m = U(\mathcal{D}^s_m)$. □

Example 10. *Recall the moralization $\mathcal{B}^s_m$ in Figure 2.1 (iii) and the docilization $\mathcal{D}^s_m$ in Figure 3.4 (vii), when testing $I_{\mathcal{B}}(a, d, f)$ and $A(p(a), p(d, be), p(f, e))$, respectively. As Lemma 3 guarantees, $\mathcal{B}^s_m = U(\mathcal{D}^s_m)$.*

Lemma 5. *The undirected graph of the DN obtained by deleting the populations in $\mathcal{P}_Y$ from $\mathcal{D}^s_m$ is the same graph obtained by deleting Y and its incident edges from $\mathcal{B}^s_m$, where $\mathcal{D}^s_m$ and $\mathcal{B}^s_m$ are in Lemma 4.*

Proof. The deletion of Y and its incident edges from $\mathcal{B}^s_m$ removes all variables in Y and all edges involving Y. In DNs, $\mathcal{P}_Y = \{p(C, D) \mid p(C, D) \in \mathcal{D} \text{ and } C \subseteq Y\}$. Thus, deleting $\mathcal{P}_Y$ from $\mathcal{D}^s_m$, denoted $\mathcal{D}^s_m - \mathcal{P}_Y$, removes precisely those populations $p(v, Pa(v))$ corresponding to the CPTs of the variables in Y. Since only the combative

traits in $\mathcal{D}^s_m - \mathcal{P}_Y$ are the variables in $U(\mathcal{D}^s_m - \mathcal{P}_Y)$, the variables are the same as in $\mathcal{B}^s_m$ after deleting Y. This also means that no edge can involve a variable of Y in the undirected graph $U(\mathcal{D}^s_m - \mathcal{P}_Y)$. Since $U(\mathcal{D}^s_m)$ is the same as $\mathcal{B}^s_m$, and $U(\mathcal{D}^s_m - \mathcal{P}_Y)$ deletes Y and its incident edges, it follows that $U(\mathcal{D}^s_m - \mathcal{P}_Y)$ is the same as $\mathcal{B}^s_m$ with Y and its incident edges removed. □

Example 11. *When testing $A(p(a), p(d, be), p(f, e))$, deleting population $p(d, be)$ in $\mathcal{P}_Y$ from Figure 3.4 (vii) gives Figure 3.4 (viii). The undirected graph of the DN in Figure 3.4 (viii) is Figure 2.1 (iv). This is the same graph obtained by deleting variable d and incident edges from $\mathcal{B}^s_m$ in Figure 2.1 (iii) in testing $I_{\mathcal{B}}(a, d, f)$ using m-separation.*

Theorem 3.1. [7] *$I(X, Y, Z)$ holds in a BN $\mathcal{B}$ if and only if $A(\mathcal{P}_X, \mathcal{P}_Y, \mathcal{P}_Z)$ succeeds in the DN $\mathcal{D}$ for $\mathcal{B}$.*

Proof. ($\Rightarrow$) Suppose $I(X, Y, Z)$ holds in a BN $\mathcal{B}$. Let $\mathcal{B}^s$ be constructed in step (i) of m-separation when testing $I(X, Y, Z)$. Let $\mathcal{B}^s_m$ be the moralization of $\mathcal{B}^s$ in step (ii) of m-separation. Let $\mathcal{D}$ be the DN for $\mathcal{B}$. Let $\mathcal{D}^s$ be constructed in step (i) of adaptation when testing $A(\mathcal{P}_X, \mathcal{P}_Y, \mathcal{P}_Z)$ in $\mathcal{D}$. Let $\mathcal{D}^s_m$ be the docilization of $\mathcal{D}^s$ in step (iii) of adaptation. By Lemma 1 and Lemma 2, $\mathcal{G}(\mathcal{D}) = \mathcal{B}$. By Lemma 3, $\mathcal{G}(\mathcal{D}^s) = \mathcal{B}^s$. By Lemma 4, $U(\mathcal{D}^s_m) = \mathcal{B}^s_m$. By Lemma 5, the undirected graph of the DN obtained by deleting the populations in $\mathcal{P}_Y$ from $\mathcal{D}^s_m$ is the same graph obtained by deleting Y and its incident edges from $\mathcal{B}^s_m$. By assumption, there is no path from a variable in X to a variable in Z after deleting Y and its incidents edges from $\mathcal{B}^s_m$. By construction, the recursive merging process in step (iv) of adaptation corresponds precisely with path existence in step (iv) of m-separation. Hence, there is no population containing both a combative trait in $T_c(\mathcal{P}_X)$ and a combative trait in $T_c(\mathcal{P}_Z)$. By definition, $A(\mathcal{P}_X, \mathcal{P}_Y, \mathcal{P}_Z)$ succeeds.

($\Leftarrow$) Follows in a similar fashion as for ($\Rightarrow$). □

Theorem 3.1 indicates that testing adaptation in DNs can be used to test independencies in a BN $\mathcal{B}$ replacing d-separation and m-separation.

Example 12. *$I_{\mathcal{B}}(a, c, f)$ holds by d-separation in Figure 2.1 (i) and $A(p(a), p(c, a), p(f, e))$ succeeds in Example 4. Similarly, $I_{\mathcal{B}}(a, d, f)$ does not hold in Figure 2.1 (i) by m-separation and $A(p(a), p(d, be), p(f, e))$ fails as shown in Example 5.*

3.3 Performing Inference

BN inference using VE, AR, and LP is unified and simplified using DN evolution.

Recall how VE computes query $P(e|b=0)$ posed to the BN $\mathcal{B}$ in Figure 2.2 (i). Referring to Figure 3.5, $\mathcal{B}$ is $\mathcal{D}$ in (i), while $P(e|b=0)$ is DN $\mathcal{D}'$ in (xiii). The removal of barren populations $p(g,ef)$ and $p(f,a)$ in (ii) corresponds to VE pruning barren variables g and f in Figure 2.2 (ii). Natural selection removes independent population $p(a)$ in (iii) and VE removes independent by evidence variable a in Figure 2.2 (iii). VE then builds $1(b)$ for the evidence variable b, while natural selection removes evident population $p(b,a)$ in (iv). As for the elimination of c, d, and h in (2.2) - (2.5): the multiplication in (2.2) is the merge of $p(c,h)$ and $p(e,cd)$ in (iv), yielding $p(ce,dh)$ in (v); the marginalization in (2.3) is the replication $p(ce,dh)$ and $p(e,dh)$ in (vi), followed by the removal of spent population $p(ce,dh)$ in (vii); (2.4) is shown in (vii) - (x); and, (2.5) is in (x) - (xiii).

The robustness of DNs only is partially revealed in this example in which DNs detect and remove barren variables, detect and remove an independent by evidence variable, and represent multiplication and marginalization to eliminate variables c, d, and h. Below, we show how DNs represent division and normalization akin to VE's step (vii). Later, in Section 6, we show how DNs can determine the elimination ordering $\sigma = (c,d,h)$ used above by VE. And, VE can impose superfluous restrictions on the order in which variables are removed by using separate steps, i.e., barren in (i), independent by evidence in (ii), and evidence in (iii). For instance, if $P(f|c)$ is posed to $\mathcal{B}$ in Figure 2.1 (i), then VE necessarily removes variables h, d, b, g, a, c in this order. In DNs, natural selection can remove populations following this order, or a, c, h, d, b, g, or even a, h, c, d, b, g, for instance.

Note that $p(e,b)$ above represents $P(e|b)$ and not $P(e|b=0)$. Additional notation, explicitly denoting evidence values such as $p(e,b=0)$ could be introduced, but will not be, since trait characteristics and population classification would remain unchanged.

Now recall the AR example in (2.6) - (2.10) and consider the DN evolution in Figure 3.6. The BN $\mathcal{B}$ in Figure 2.3 (i) is the DN $\mathcal{D}$ in Figure 3.6 (i). The multiplication $P(a,d|b) = P(a) \cdot P(d|a,b)$ in (2.6) is the merge of populations $p(a)$ and $p(d,ab)$ in Figure 3.6 (i), yielding $p(ad,b)$ in Figure 3.6 (ii). The marginalization in (2.7) is the replication in Figure 3.6 (iii). And, the division $P(a|b,d) = P(a,d|b)/P(d|b)$ in (2.8)

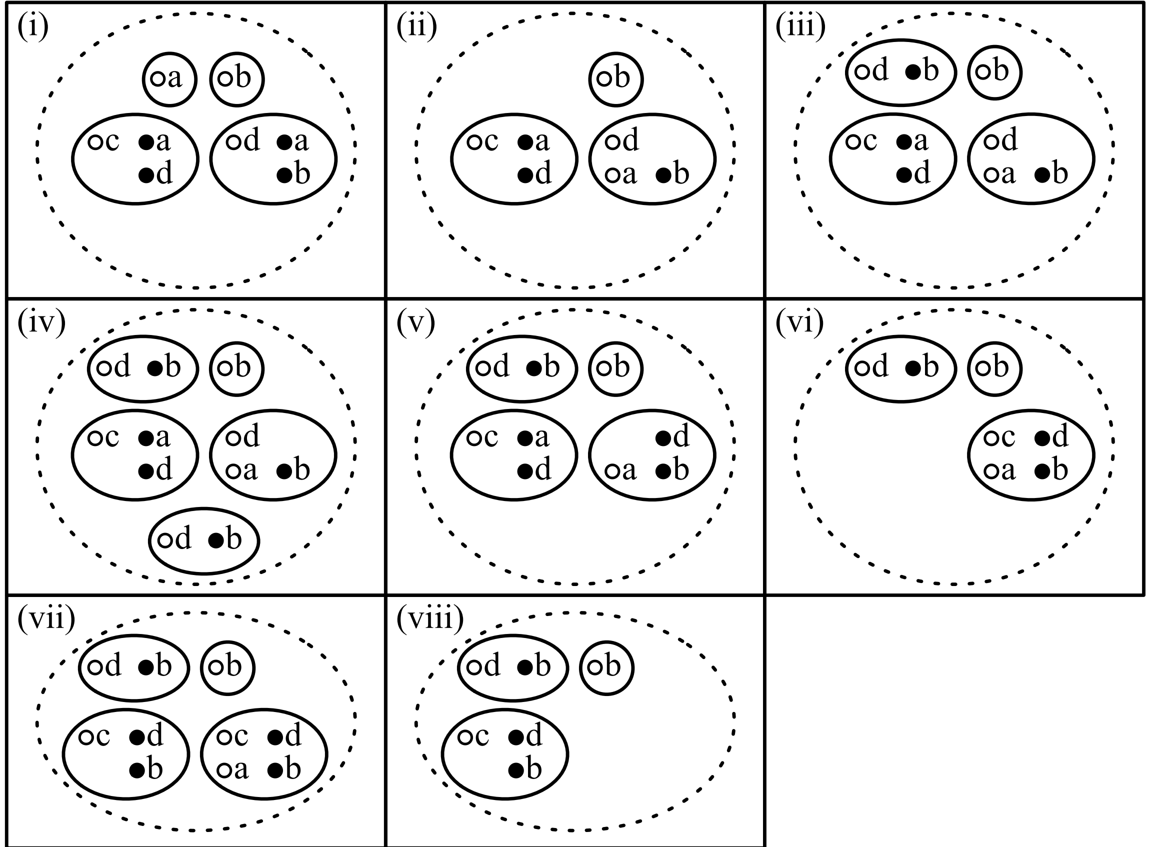


Figure 3.6: Graphically representing AR's computation.

can be the replication $p(ad, b)$ and $p(d, b)$ in Figure 3.6 (iv), followed by the merge of $p(ad, b)$ and $p(d|b)$, giving $p(a, bd)$, in Figure 3.6 (v). The multiplication in (2.9) is the merge in Figure 3.6 (vi). Lastly, the marginalization in (2.10) is the replication in Figure 3.6 (vii), followed by natural selection removing spent population $p(ac, bd)$, giving Figure 3.6 (viii). Observe that DN evolution simplifies AR in that merge can represent both multiplication and division. Thus, DNs can represent VE's step (vii), which multiplies to obtain $P(X, Y = y)$, marginalizes to yield $P(Y = y)$, and divides to obtain $P(X|Y = y)$.

Akin to Darwin's [18] finches on the Galápagos Islands, populations can *migrate* between DNs, but in evolution natural selection also removes rejected populations. Given DNs $\mathcal{D}$ and $\mathcal{D}'$, population $p(C, D)$ in $\mathcal{D}$ is *rejected*, if both t_c and t_d are weak, for at least one trait t in CD. In Figure 3.7, given DN $\mathcal{D} = \{p(a), p(b, a)\}$ in (i) and

$\mathcal{D}' = \{p(a, h), p(b, h)\}$ in (x), if $p(c)$ migrates to $\mathcal{D}$, then it is rejected, since both c_c and c_d are weak.

Example 13. In Figure 3.7, consider one explanation of the evolution of the DNs in (i) into the respective DNs in (x). In (i), $p(i, h)$ and $p(j, h)$ can replicate and migrate as in (ii). Evolution in (i) can yield $p(b)$, $p(d)$ and $p(f)$ in (ii) that migrate in (iii). Evolution in (iii) can create $p(g)$ in (iv) that migrates in (v). The remainder follows similarly.

The evolution of multiple DNs in Example 13 corresponds to the inward phase of the LP example in Section 2. Observe how propagated messages in Figure 2.4 such as $P(g)$, $P(i|h = 0)$, and $P(j|h = 0)$ correspond precisely to migrating populations $p(g)$, $p(i, h)$, and $p(j, h)$ in Figure 3.7. Posterior probabilities $P(i|h = 0)$, $P(j|h = 0)$, and $P(k|h = 0)$ can be computed at the root node $\{i, j, k\}$ at the end of the inward phase, as in Figure 3.7 (v). Posterior probabilities of all non-evidence variables $P(a|h = 0), P(b|h = 0), \ldots, P(l|h = 0)$ can be computed at the end of the outward phase, as in Figure 3.7 (x). DNs simplify LP in a few ways. DNs perform adaptation and evolution in $\mathcal{D}$, but LP tests independencies in $\mathcal{B}$ and propagates in $\mathcal{T}$. In LP, separate steps are used to detect barren variables and independent by evidence variables, while natural selection finds both. LP requires a node to receive all messages before deciding the irrelevant potentials in message construction. For example, in Figure 2.2, LP forces node $\{b, g, h, i, j\}$ to wait for message $P(g)$ before determining that it is irrelevant for message construction to node $\{i, j, k\}$. In contrast, DNs allow populations $p(i, h)$ and $p(j, h)$ to migrate immediately in Figure 3.7 (ii) before $p(g)$ migrates in Figure 3.7 (v).

3.4 Representing Other Bayesian and Non-Bayesian Frameworks

We demonstrate how DNs can represent another Bayesian framework, called chain graph models, as well as a non-Bayesian framework, namely, relational databases.

3.4.1 Chain Graph Models

The discussion of *chain graph models* (CGMs) in this section draws from [44].

Figure 3.7: Graphically representing LP's computation.

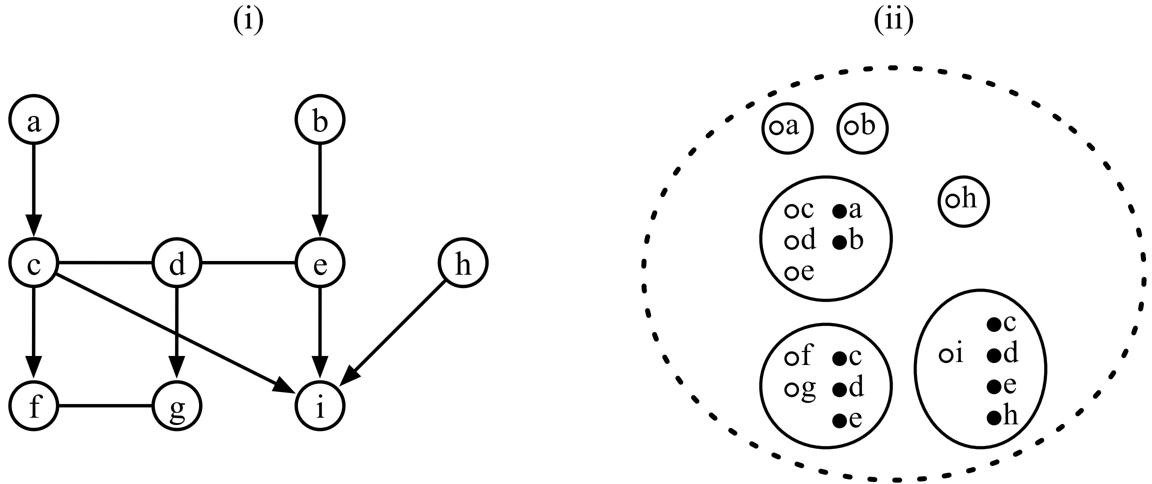


Figure 3.8: (i) A PDAG. (ii) A DN representing the CGM in Example 16.

A *partially directed acyclic graph* (PDAG), also known as a chain graph, is an acyclic graph containing both directed and undirected edges. The acyclicity requirement ensures that the PDAG can be decomposed into a directed graph of *chain components*, where the variables within each chain component are connected to each other only with undirected edges.

Example 14. *[44] Given variables $U = \{a, b, c, d, e, f, g, h, i\}$, one PDAG is depicted in Figure 3.8 (i), where the six chain components are $\{a\}$, $\{b\}$, $\{c, d, e\}$, $\{f, g\}$, $\{h\}$, and $\{i\}$.*

Each chain component X is associated with a *conditional random field* (CRF) that defines a CPT $P(X|Y)$ of X given its parents Y in the PDAG.

Example 15. *In the PDAG of Figure 3.8 (i), the six CPTs associated with the chain components are $P(a)$, $P(b)$, $P(c, d, e|a, b)$, $P(f, g|c, d, e)$, $P(h)$, and $P(i|c, d, e, h)$.*

A PDAG can factorize a probability distribution as a product of each of the chain components given its parents. Such a representation is called a *chain graph model* (CGM).

Example 16. *The PDAG in Figure 3.8 (i) and the CPTs in Example 15 define a CGM $\mathcal{M}$ on variables $U = \{a, b, c, d, e, f, g, h, i\}$:*

$$P(U) = P(a) \cdot P(b) \cdot P(c, d, e|a, b) \cdot P(f, g|c, d, e) \cdot P(h) \cdot P(i|c, d, e, h). \quad (3.18)$$

DNs can represent CGMs.

Definition 3.4. *Let $\mathcal{M}$ be a CGM on a set U of variables. The DN $\mathcal{D}$ defined by $\mathcal{M}$ is* $\mathcal{D} = \{p(X,Y) \mid P(X|Y) \text{ is in } \mathcal{M}\}$.

Example 17. *The CGM $\mathcal{M}$ in Example 16 can be represented by the DN $\mathcal{D}$ in Figure 3.8 (ii).*

Representing the testing of independencies in CGMs using *c-separation* [44] remains as future work.

3.4.2 Relational Databases

Our discussion on *relational databases* follows the eloquent treatment provided by [51].

Let U be a finite set of attributes, each with a finite domain. A relation over attributes CD is denoted $r(\underline{C}, D)$, where C is the primary key and D is the set of non-key attributes in the relational scheme.

Example 18. *Table 3.1 shows four relations: $r_1(\underline{a}, \underline{b}, \underline{c})$, $r_2(b, c, \underline{e})$, $r_3(b, c, \underline{d})$, and $r_4(e, \underline{f})$.*

The *natural join* $\bowtie$ of the four relations in Table 3.1 gives the *universal relation*, denoted r, in Table 3.2 (i). The *projection* of relation r_1 in Table 3.1 (i) onto attributes $\{b, c\}$, denoted $\pi_{bc}(r_1)$, is illustrated in Table 3.2 (ii).

One computational task in relational databases is to transform given relations into projections of the universal relation. Provided that the database scheme is *acyclic* [4], this can be accomplished using a semijoin program.

Example 19. *Let us transform the four relations in Table 3.1 into projections of the universal relation r in Table 3.2 (i). As required, the database scheme is acyclic and, hence, can be illustrated as the join tree in Figure 3.9. The inward pass of one semijoin program involves building and passing messages m_1, m_2, and m_3, as depicted*

Table 3.1: Four given relations $r_1(a,b,c)$, $r_2(b,c,e)$, $r_3(b,c,d)$, and $r_4(e,f)$.

(i)

a	b	c
0	0	0
1	0	1
1	1	0
1	2	1

(ii)

b	c	e
0	0	1
0	1	0
1	0	2

(iii)

b	c	d
0	0	0
0	0	1
0	1	2
1	0	3
2	1	4

(iv)

e	f
0	0
1	1
2	2

in Figure 3.9. The database computation for the inward pass is:

$$m_1 = \pi_{bc}(r_1), \tag{3.19}$$

$$r_2' = r_2 \bowtie m_1, \tag{3.20}$$

$$m_2 = \pi_{bc}(r_3), \tag{3.21}$$

$$r_2'' = r_2' \bowtie m_2, \tag{3.22}$$

$$m_3 = \pi_e(r_2''), \tag{3.23}$$

$$r_4' = r_4 \bowtie m_3. \tag{3.24}$$

The outward pass is then:

$$m_4 = \pi_e(r_4'), \tag{3.25}$$

$$r_2''' = r_2'' \bowtie m_4, \tag{3.26}$$

$$m_5 = \pi_{bc}(r_2'''), \tag{3.27}$$

$$r_3' = r_3 \bowtie m_5, \tag{3.28}$$

$$m_6 = \pi_{bc}(r_2'''), \tag{3.29}$$

$$r_1' = r_1 \bowtie m_6. \tag{3.30}$$

The important point in Example 19 is that the semijoin program in (3.19)-(3.30) converts the given relations in Table 3.1 into the desired projections in Table 3.3.

DNs can represent relational databases. Here, a population $p(C,D)$ represents a relation $r(\underline{C},D)$ with primary key $\underline{C}$ and non-key attributes D.

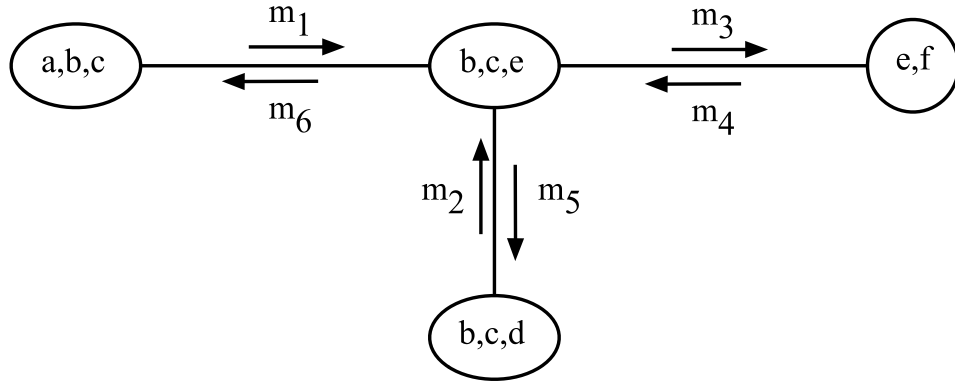


Figure 3.9: A join tree for the semijoin program in Example 19.

Table 3.2: (i) The universal relation r defined by the join of the four relations in Table 3.1. (ii) The projection of relation r_1 in Table 3.1 (i) onto $\{b, c\}$.

(i)

a	b	c	d	e	f
0	0	0	0	1	1
0	0	0	1	1	1
1	0	1	2	0	0
1	1	0	3	2	2

(ii)

b	c
0	0
0	1
1	0
2	1

Example 20. *The four relations in Example 18 are represented as the respective populations $p(abc)$, $p(e, bc)$, $p(d, bc)$, and $p(f, e)$.*

When DNs represent relational databases, merge means natural join $\bowtie$, and replication is projection π. Thereby, DNs can model computation in relational databases. To represent a semijoin program, the relation $r(\underline{C}, D)$ at each join tree node is represented as a DN $\{p(\underline{C}, D)\}$. For instance, Figure 3.10 depicts the DNs in our running example.

Example 21. *Recall the DNs in Figure 3.10. Then Figure 3.11 (i)-(vi) respectively represent the relational database computation in (3.19)-(3.24).*

It is common knowledge that division is required in the outward pass of probability

Table 3.3: Desired projections of the universal relation r in Table 3.2 (i).

(i)

a	b	c
0	0	0
1	0	1
1	1	0

(ii)

b	c	e
0	0	1
0	1	0
1	0	2

(iii)

b	c	d
0	0	0
0	0	1
0	1	2
1	0	3

(iv)

e	f
0	0
1	1
2	2

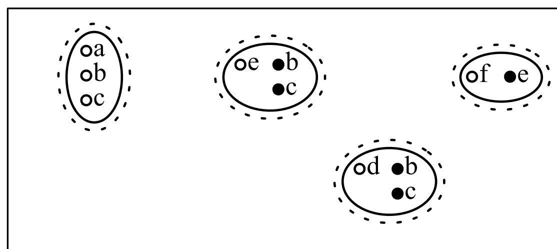


Figure 3.10: DNs can represent the relational database in Example 18.

propagation in join trees [75]. On the contrary, there is no requirement for division in the outward pass of relational database computation in join trees, since there are no probabilities to be concerned with. In DNs, recall that division is handled by defining the merge of two combative traits to yield a docile trait. Consequently, when DNs represent relational databases, two populations *merge* together as follows: for each trait t appearing in either population, if t is docile in both populations, then t is docile in the merged population; otherwise, t is combative. Under this definition, the reader can verify the DN representation of the outward pass in (3.25)-(3.30) starting from Figure 3.11 (vi).

3.5 Elimination Orderings

Determining good elimination orderings σ is an important practical issue. Empirical results show that *min-neighbours* is one of four heuristics that performs surprisingly well in practice [44]. Given that $P(X|Y = y)$ is posed to a BN $\mathcal{B}$, the score

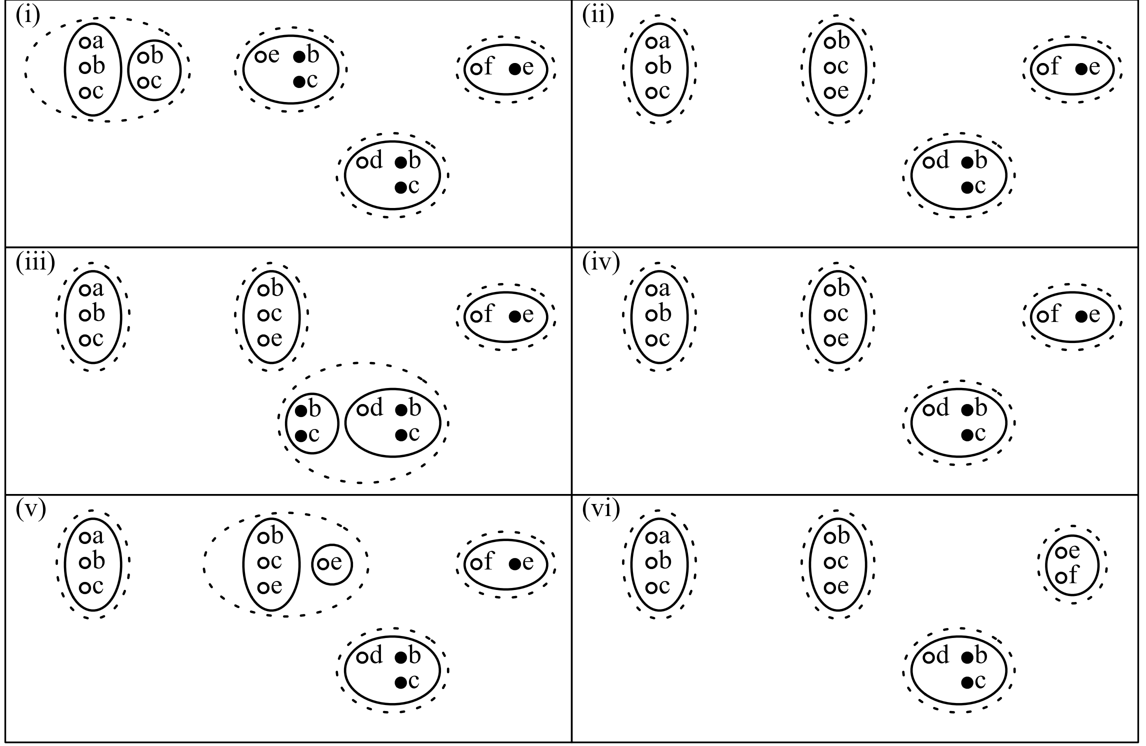


Figure 3.11: (i)-(vi) respectively represent the relational database computation in (3.19)-(3.24), given the DNs in Figure 3.10.

$s(v)$ of variable v is the number of edges involving v in $\mathcal{B}^s_m$. A variable v with a minimum $s(v)$ is removed after adding edges among all of v's neighbours. The algorithm recursively repeats until all variables not in XY are removed. For example, in Figure 2.2, $P(e|b = 0)$ is posed to $\mathcal{B}$ in (i) and $\mathcal{B}^s_m$ is in (iv). The scores then are $s(c) = 3$, $s(d) = 3$, and $s(h) = 3$. Removing, for example, c after adding edge (e, h) yields the undirected graph $\mathcal{U}_1 = \{(b, h), (d, h), (d, e), (e, h)\}$. Now, as $s(d) = 2$ and $s(h) = 3$, d is removed, giving $\mathcal{U}_2 = \{(b, h), (e, h)\}$, and then h, leaving $\mathcal{U}_3 = \{(b, e)\}$. Thus, $\sigma = (c, d, h)$.

Min-neighbours can be represented in DNs [8]. Given a BN $\mathcal{B}$ and a query $P(X|Y = y)$, let $\mathcal{D}$ be the DN for $\mathcal{B}$, DN $\mathcal{D}' = p(X, Y)$, and let DN $\mathcal{D}^s$ be obtained by natural selection acting as in evolution on $\mathcal{D}$ with respect to $\mathcal{D}'$. The weak combative traits t_c of $\mathcal{D}^s$ are removed recursively based upon a minimum score. The

score $s(t_c)$ is the number of traits appearing with either t_c or t_d. Eliminate t_c by recursively merging all populations with t_c or t_d, yielding $p(C, D)$, replicating $p(C, D)$ and $p(C - t_c, D)$, and natural selection removing spent population $p(C, D)$. For example, given $\mathcal{B}$ in Figure 2.2 (i) and $P(e|b = 0)$, then $\mathcal{D}$ is in Figure 3.5 (i), $\mathcal{D}'$ in (xiii), and $\mathcal{D}^s$ in (iv). The weak combative traits are c_c, d_c, and h_c. The score of c_c is 3, since h_d appears with c_c in $p(c, h)$, and e_c and d_d appear with c_d in $p(e, cd)$. The scores of d_c and h_c are also 3. Eliminating, for example, c_c yields the DN in Figure 3.5 (vii), after merging $p(c, h)$ and $p(e, cd)$ as $p(ce, dh)$, replicating $p(ce, dh)$ and $p(e, dh)$, and natural selection removing $p(ce, dh)$. Now, the score of d_c is 2 and that of h_c is 3. Thus, in Figure 3.5 (vii), we eliminate d_c, giving (x), followed by h_c, giving (xiii), and $\sigma = (c, d, h)$.

It can be verified that if $\mathcal{D}^s$ is the DN for a sub-DAG $\mathcal{B}^s$ constructed from a DAG $\mathcal{B}$, then $\mathcal{B}^s_m = \mathcal{U}(\mathcal{D}^s)$. For example, given $\mathcal{D}^s$ in Figure 3.5 (iv) for the sub-DAG $\mathcal{B}^s$ in Figure 2.2 (iii), then $\mathcal{U}(\mathcal{D}^s)$ is $\mathcal{B}^s_m$ in Figure 2.2 (iv). By construction, the scores $s(t_c)$ in the DN $\mathcal{D}^s$ are identical to the scores $s(v)$ in $\mathcal{B}^s_m$. Also, by construction, the undirected graph of the DN, after eliminating t_c with a lowest score, is $\mathcal{B}^s_m$ with the corresponding variable v eliminated. For example, the undirected graphs of the DNs in Figure 3.5 (vii), (x), and (xiii), are respectively $\mathcal{U}_1$, $\mathcal{U}_2$, and $\mathcal{U}_3$ above. DNs simplify min-neighbours by not requiring moralization. Similarly, DNs can represent *min-weight*, *min-fill*, and *weighted-min-fill* [8]. Moreover, DN have led to a new heuristic, called *population energy*, for which preliminary experimental results are promising [8].

3.6 Practical Applications of Darwinian Networks

This section describes two applications of DNs that are not themselves to be counted as thesis contributions. Besides being a robust framework for representing various frameworks and methods, DNs also lead to faster approaches to inference and modeling.

Algorithm 1 Simple Message Construction

1: **procedure** SMC($\mathcal{F}$, S, Y)
2: $\quad\mathcal{F}$ = RemoveBarren($\mathcal{F}$, S)
3: $\quad$**while** $\exists\ \phi(X) \in \mathcal{F}$ with $v \notin S - Y$ and $v' \in S - Y$ **do**
4: $\quad\quad\mathcal{F}$ = SumOut(v, $\mathcal{F}$)
5: $\quad$**return** $\{\phi(X) \in \mathcal{F} \mid X \subseteq S\}$

3.6.1 Simple Propagation

DNs were utilized as the basis of a novel join tree propagation algorithm, called *simple propagation* (SP) [10]. The key tenet of SP is easily visualized using populations in DNs. In this manner, SP exploits the factorization of potentials, but without the overhead required in LP.

SP only differs from LP in how messages are constructed. SP uses Algorithm 1, called *Simple Message Construction* (SMC), to construct the message sent from a node N_1 to a neighbour node N_2, where $\mathcal{F}$ is the factorization at N_1, the separator $S = (N_1 \cap N_2) \cup Y$, and Y is the evidence.

In Example 22, we emphasize starting at (3.31) the key tenet of SP, namely, line 3 of SMC. By "one in, one out," we mean a potential in $\mathcal{F}$ has a non-evidence variable in the separator and another non-evidence variable not in the separator.

Example 22. *[10] Consider a join tree with the following three nodes:*

$$\begin{aligned} N_1 &= \{a, b, c\}, \\ N_2 &= \{b, c, d, e, f, g, h, i, j, l, m\}, \textit{ and} \\ N_3 &= \{i, j, k, l, m\}, \end{aligned}$$

and edges (N_1, N_2) *and* (N_2, N_3)*. Let the observed evidence be* $d = 0$*. Those CPTs containing* d *are updated by keeping only those rows with* $d = 0$*. Assigning the CPTs to* N_1*,* N_2*, and* N_3 *can yield the following respective factorizations* $\mathcal{F}_1$*,* $\mathcal{F}_2$*, and* $\mathcal{F}_3$*:*

$$\begin{aligned} \mathcal{F}_1 &= \{P(a), P(b|a), P(c|a)\}, \\ \mathcal{F}_2 &= \{P(d=0|b,c), P(e|d=0), P(f|d=0,e), \\ &\qquad P(g|e), P(h|e), P(i|d=0,h), P(j|i), P(m|g,l)\}, \\ \mathcal{F}_3 &= \{P(k|j), P(l|k)\}. \end{aligned}$$

Let N_3 be the root of the join tree. Now, each node is augmented with d.

The discussion becomes interesting after node N_1 sends its message to node N_2. N_1 calls SMC$(\mathcal{F}, S, Y)$ *with $\mathcal{F} = \{P(a), P(b|a), P(c|a)\}$, $S = \{b, c, d\}$, and $Y = \{d\}$. In line 2, no potentials are removed by* RemoveBarren*. In line 3, potential $P(b|a)$ contains non-evidence variable b in the separator and non-evidence variable a not in the separator. In line 4, SP eliminates a as:*

$$P(b,c) \;=\; \sum_a P(a,b,c) \;=\; \sum_a P(a) \cdot P(b|a) \cdot P(c|a)$$

Hence, N_1 sends message $P(b,c)$ to N_2.

Node N_2 calls SMC$(\mathcal{F}, S, E)$ *to compute its message to node N_3, where factorization $\mathcal{F}$ is:*

$$\mathcal{F} \;=\; \mathcal{F}_2 \cup \{P(b,c)\},$$

separator $S = \{d, i, j, l, m\}$, and $Y = \{d\}$. In line 2, $P(f|d=0,e)$ is removed from $\mathcal{F}$ by RemoveBarren*, giving*

$$\mathcal{F} = \{P(b,c), P(d=0|b,c), P(e|d=0), P(g|e), P(h|e), P(i|d=0,h), P(j|i), P(m|g,l)\}. \tag{3.31}$$

DNs are exploited in Figure 3.12 (i) to visualize the factorization $\mathcal{F}$ in (3.31). In population $p(m, gl)$, for instance, m and l are non-evidence variables in the separator and g is a non-evidence variable not in the separator. Thus, in line 4 of SMC*, g is eliminated as:*

$$P(m|e,l) \;=\; \sum_g P(g|e) \cdot P(m|g,l),$$

yielding the new factorization:

$$\mathcal{F} \;=\; \{P(b,c), P(d=0|b,c), P(e|d=0), P(h|e), P(i|d=0,h), P(j|i), P(m|e,l)\}. \tag{3.32}$$

Once again, DNs are exploited in Figure 3.12 (ii) to visualize the factorization $\mathcal{F}$ in (3.32). In the population, say $p(m, el)$, m and l are non-evidence variables in the

separator and e is a non-evidence variable not in the separator. Thus, SP eliminates e as

$$P(h, m|d = 0, l) \;=\; \sum_{e} P(e|d = 0) \cdot P(h|e) \cdot P(m|e, l),$$

giving the following factorization:

$$\mathcal{F} \;=\; \{P(b, c), P(d = 0|b, c), P(i|d = 0, h), P(j|i), P(h, m|d = 0, l)\}. \quad (3.33)$$

DNs are utilized in Figure 3.12 (iii) to visualize the factorization $\mathcal{F}$ in (3.33). Now, in population $p(i, dh)$, for instance, i is a non-evidence variable in the separator and h is a non-evidence variable not in the separator. Thus, SP eliminates h as

$$P(i, m|d = 0, l) \;=\; \sum_{h} P(i|d = 0, h) \cdot P(h, m|d = 0, l),$$

giving

$$\mathcal{F} \;=\; \{P(b, c), P(d = 0|b, c), P(j|i), P(i, m|d = 0, l)\}, \quad (3.34)$$

as illustrated in Figure 3.12 (iv). As there are no longer potentials with the "one in, one out" property, SP sends message $m_2 = \{P(j|i), P(i, m|d = 0, l)\}$ to N_3 in line 5. The outward phase is not described.

Table 3.4 [10] reports an empirical comparison between SP and LP. The experiments were performed on the 28 benchmark BNs listed in column 1. Column 2 shows the number of variables in each BN. The average computation time in seconds is calculated over 100 runs and reported for LP and SP in columns 3 and 4, respectively. Out of 28 BNs, SP was faster in 18 cases, tied LP in 5 cases, and was slower than LP in 5 cases. SP tends to be faster than LP because SP simply exploits the "one in, one out" property in the factorization of potentials.

3.6.2 i-Separation

DNs have led to a faster approach for testing independencies in BNs, called *i-separation* [6].

We first show how DNs revealed superfluous computation in m-separation. The docilization step of adaptation can be refined to add $p(\emptyset, D)$ only for $p(C, D) \in \mathcal{P}_Y$ with $|D| > 1$. Adding $p(\emptyset, D)$ for $p(C, D) \notin \mathcal{P}_Y$ is extraneous, since the merge of $p(C, D)$ and $p(\emptyset, D)$ is $p(C, D)$.

Example 23. *Recall testing adaptation* $A(p(a), p(d, be), p(f, e))$ *in Example 5, as illustrated in Figure 3.4 (vi)-(x). In step (ii), population* $p(\emptyset, ag)$ *is added. However, in step (iii),* $p(\emptyset, ag)$ *is merged with population* $p(b, ag) \notin \mathcal{P}_{\mathcal{Y}}$*, giving* $p(b, ag)$ *itself. Hence, adding* $p(\emptyset, ag)$ *is wasteful.*

Since adaptation in DNs mimics m-separation, the wastefulness in Example 23 highlights erroneous computation in m-separation.

When testing $I(X, Y, Z)$ in a BN $\mathcal{B}$ using m-separation, the moralization step need only to add edges between parents of a common child v when $v \in Y$.

Example 24. *Recall the moralization of Figure 2.1 (iii) when testing* $I_{\mathcal{B}}(a, d, f)$*. Edge* (b, e) *is essential as* $d \in Y$*. However, edge* (a, g) *is superfluous as* $b \notin Y$*.*

More generally, adding superfluous edges means creating paths that do not have to be examined. Turning to d-separation, we observed the same wastefulness.

Example 25. *Consider testing* $I(a, de, g)$ *in the DAG* $\mathcal{B}$ *of Figure 3.13 (i) using d-separation. The path* (a, c) *is active. Then the longer path* (a, c)*,* (c, f) *is active. Continuing on, the path* (a, c)*,* (c, f)*,* (f, h) *is active. Finally, the path* (a, c)*,* (c, f)*,* (f, h)*,* (g, h) *is blocked, since* h *is a closed convergent valve. However, the reader can verify that it is impossible to have a active path starting at variable* a*, passing through variable* f*, and reaching variable* g*. Hence, testing these paths in d-separation is wasteful.*

Example 25 highlights the importance of not checking paths needlessly.

Inaugural-separation (*i-separation*) was proposed in [6] as a novel method for testing independencies in BNs. In [6], the notion of an *inaugural* variable was introduced, the salient feature of which is that in testing $I(X, Y, Z)$, any path from X to Z involving an inaugural variable is blocked. This means that paths involving inaugural variables can be safely ignored and pruned from the BN.

Example 26. *Let us test* $I(a, de, g)$ *in the DAG* $\mathcal{B}$ *of Figure 3.13 (i) using i-separation. Variables* f *and* h *are inaugural, and are thus pruned from* $\mathcal{B}$ *yielding the sub-DAG in Figure 3.13 (ii). Only the paths in (ii) are checked, yielding that* $I(a, de, g)$ *holds.*

Example 26 emphasizes how DNs led to the identification of wastefulness in d-separation.

An empirical comparison between d-separation and i-separation was reported in [6]. The result showed that i-separation was faster than d-separation on all test BNs with 186 or more variables.

3.7 Conclusion

While many works have generalized BNs, we seek to simplify reasoning with BNs. DNs, a biological perspective of BNs, are surprisingly simple, yet remarkably robust. DNs can represent the testing of independencies using d-separation and m-separation, belief update using VE, AR, and LP, and the determination of good elimination orderings using min-neighbours. DNs simplify each of these separate techniques, while unifying them into one platform. We have also shown hows DNs can represent chain graph models and relational databases. Practical benefits of DNs include faster algorithms for inference and modeling. The DN keystone is the novel representation of a population $p(C, D)$ using both combative traits C (coloured white) and docile traits D (coloured black).

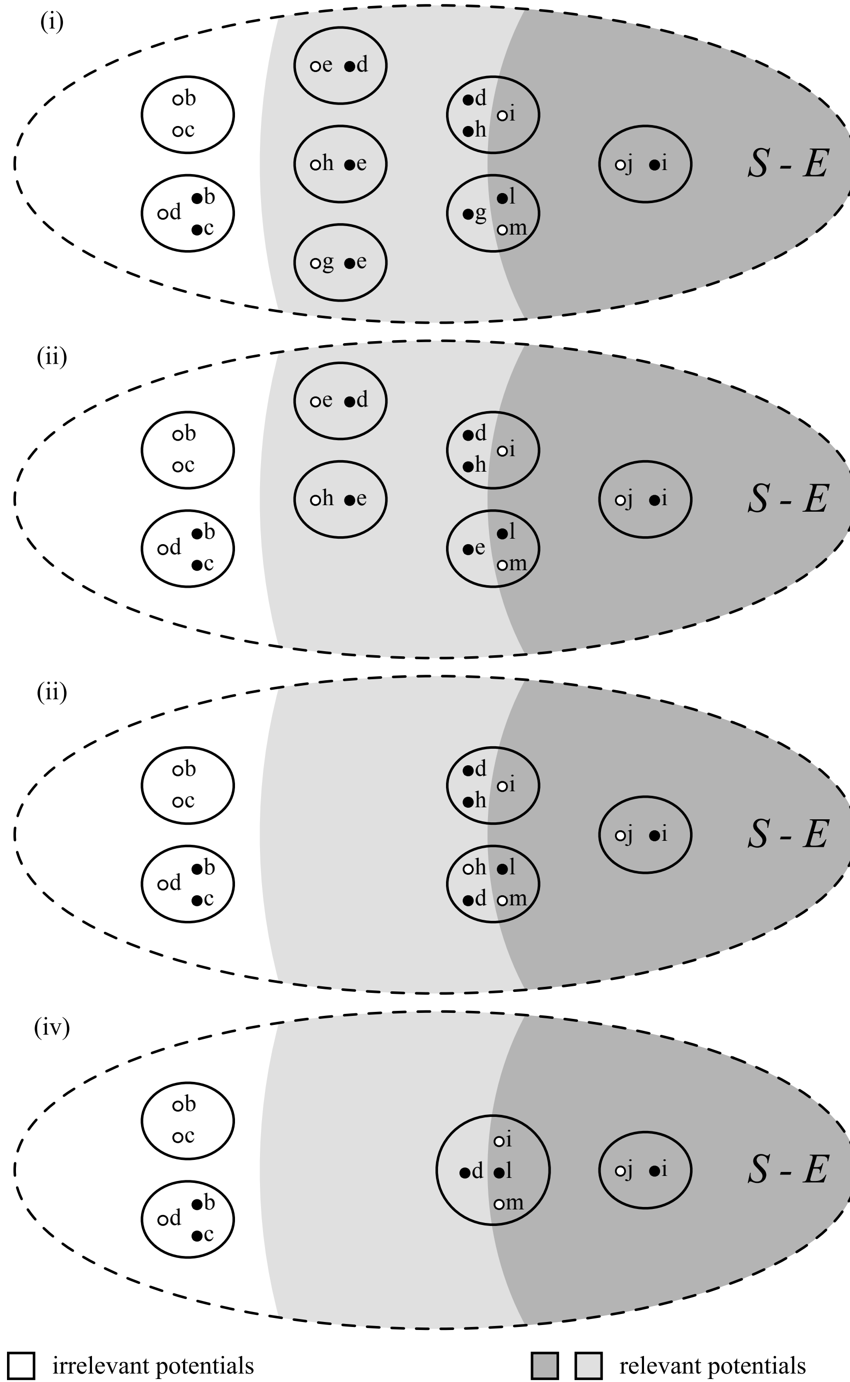


Figure 3.12: Utilizing DNs to visualize the “one in, one out” property exploited in SP. Graphically depicting the potentials in $\mathcal{F}$ of (3.31) at N_2 in (i). After eliminating variable g, (ii) shows $\mathcal{F}$ of (3.32). After eliminating e, (iii) shows $\mathcal{F}$ of (3.33). Finally, eliminating h gives $\mathcal{F}$ in (3.34) shown in (iv).

Table 3.4: Average time in seconds of LP and SP to propagate messages and compute posteriors over 100 runs in each BN.

BN	Variables	LP	SP	Savings
Water	32	0.06	0.05	17%
Oow	33	0.07	0.06	14%
Oow_Bas	33	0.04	0.03	25%
Mildew	35	0.05	0.04	20%
Oow_Solo	40	0.07	0.06	14%
Hkv2005	44	0.23	0.27	-17%
Barley	48	0.09	0.10	-11%
Kk	50	0.09	0.09	0%
Ship	50	0.16	0.17	-6%
Hailfinder	56	0.02	0.02	0%
Medianus	56	0.04	0.03	25%
3Nt	58	0.02	0.01	50%
Hepar_Ii	70	0.03	0.03	0%
Win95Pts	76	0.03	0.03	0%
System_V57	85	0.06	0.05	17%
Fwe_Model8	109	0.14	0.15	-7%
Pathfinder	109	0.12	0.11	8%
Adapt_T1	133	0.04	0.04	0%
Cc145	145	0.10	0.08	20%
Munin1	189	0.54	0.75	-39%
Andes	223	0.15	0.13	13%
Cc245	245	0.20	0.18	10%
Diabetes	413	0.34	0.31	9%
Adapt_T2	671	0.24	0.22	8%
Amirali	681	0.45	0.41	9%
Munin2	1003	0.49	0.45	8%
Munin4	1041	0.61	0.57	7%
Munin3	1044	0.66	0.64	3%

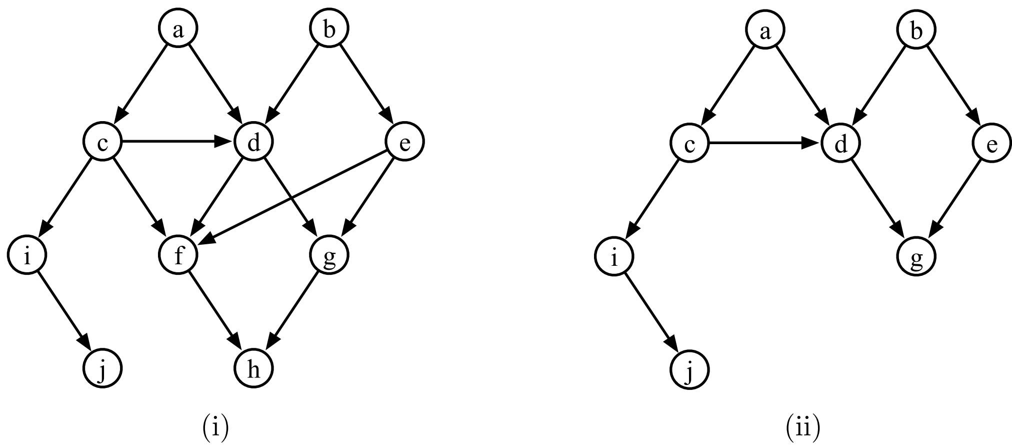


Figure 3.13: When testing the independence $I(a, de, g)$ in the DAG shown in (i), i-separation builds the sub-DAG depicted in (ii).

Chapter 4

Determining Good Elimination Orderings with Darwinian Networks

4.1 Introduction

Darwinian networks (DNs) [7] were proposed to simplify working with *Bayesian networks* (BNs) [61]. DNs are a richer representation allowing them to simplify both modelling and inference.

An important practical consideration in BN inference is determining good *elimination orderings*, denoted σ [42]. Empirical results show that *min-neighbours* (MN), *min-weight* (MW), *min-fill* (MF), and *weighted-min-fill* (WMF) are four heuristics that perform well in practice [44]. Given a query $P(X|Y)$ posed to a BN $\mathcal{B}$, all variables except XY are recursively eliminated from the moralization $\mathcal{B}_m$ based upon a minimum score $s(v)$.

In this chapter, we show how these four heuristics can be represented in DNs. More importantly, we propose a new heuristic, called *potential energy* (PE), based on DNs themselves. Our analysis of PE shows that it can: (i) better score a variable; (ii) better model the multiplication of the probability tables for the chosen variable; (iii) more clearly model the marginalization of the chosen variable; and (iv) maintain a one-to-one correspondence between the remaining variables and probability tables.

4.2 Background

4.2.1 Elimination Orderings in Bayesian Networks

We use the following running example throughout the paper. Assuming query $P(f|d)$ is posed to the BN $\mathcal{B}$ in Figure 4.1 (i), variables a, b, c, and e must be recursively eliminated from the *moralization* [61] $\mathcal{B}_m$ in Figure 4.1 (ii). Eliminating a, for instance, involves adding edges between a's neighbours as in (iii) and then removing a as in (viii).

In *min-neighbours* (MN), the score $s(v)$ of variable v is the number of edges involving v.

Example 27. *Referring to Figure 4.1, MN can determine elimination ordering* $\sigma = (c, a, b, e)$ *by (ii), (iv)-(vii).*

In *min-weight* (MW), the score $s(v)$ of a variable v is the product of the domain cardinalities of v's neighbours. The domain cardinality of a is 5, while the rest are binary.

Example 28. *Referring to Figure 4.1, MW can determine* $\sigma = (a, c, b, e)$ *by (ii), (viii)-(xi).*

In *min-fill* (MF), the score $s(v)$ of a variable v is the number of edges that need to be added between v's neighbours due to v's elimination.

Example 29. *Referring to Figure 4.1, MF can determine* $\sigma = (c, b, a, e)$ *by (ii), (iii), (xii)-(xv).*

In *weighted-min-fill* (WMF), the score $s(v)$ of variable v is the sum of the weights of the edges that need to be added between v's neighbours due to v's elimination. The weight of an edge is the product of the domain cardinalities of its constituent vertices.

Example 30. *Referring to Figure 4.1, WMF can determine* $\sigma = (c, a, b, e)$ *by (ii), (iv)-(vii).*

4.3 Elimination Orderings in Darwinian Networks

A trait t with a minimum score is eliminated by recursively merging all populations with t, yielding $p(C, D)$, replicating $p(C, D)$ as $p(C, D)$ and $p(C - t, D)$, and letting natural selection remove spent population $p(C, D)$.

Given two DNs $\mathcal{D}$ and $\mathcal{D}'$, recursively eliminate traits appearing in $\mathcal{D}$ but not $\mathcal{D}'$. We will use the same running example for each heuristic, namely, $\mathcal{D}$ is in Figure 4.2 (i) and $\mathcal{D}'$ is in Figure 4.2 (viii). By the above, each heuristic seeks to eliminate traits a, b, c, and e from $\mathcal{D}$.

We now present four heuristics for scoring traits in DNs.

4.3.1 Representing Min-Neighbours in DNs

Two traits t and t' in a DN $\mathcal{D}$ are *related*, if they appear together in at least one population in $\mathcal{D}$; otherwise, they are *unrelated*. For example, in Figure 4.2 (i), trait a is related to traits b, c, d, and e. Traits a and f are unrelated, since they do not appear together in any population.

Definition 4.1. *To represent MN, the score $s(t)$ of a trait t is the number of traits related to t.*

Example 31. *In Figure 4.2 (i), the score $s(a)$ of trait a is 4, since a is related to traits b, c, d, and e. Similarly, $s(b)$ is 3, $s(c)$ is 3, and $s(e)$ is 5. Eliminating a trait with a minimum score, say c, by merging in (ii), replicating in (iii), and letting natural selection act, gives (iv). Now, $s(a)$ and $s(b)$ both are 3, and $s(e)$ is 4. Eliminating, say a, yields (v). Here, both $s(b)$ and $s(e)$ are 3. Eliminating, say b, gives (vi). Then eliminating e yields (vii). Therefore, $\sigma = (c, a, b, e)$.*

4.3.2 Representing Min-Weight in DNs

Given a DN $\mathcal{D}$ representing a BN $\mathcal{B}$, the *energy* of a trait t is the domain cardinality of the variable v to which it corresponds. For example, given that variable b in Example 27 is binary, then the energy of trait b in Example 31 is 2.

Definition 4.2. *To represent MW, the score $s(t)$ of a trait t is the product of the energies of the traits related to t.*

Example 32. *As the DN $\mathcal{D}$ in Figure 4.2 (i) represents the BN $\mathcal{B}$ in Figure 4.1 (i), the energies of traits a, b, c, d, e, and f are 5, 2, 2, 2, 2, and 2, respectively. The traits related to trait a in $\mathcal{D}$ are b, c, d and e. Thus, the score $s(a)$ is 16. Similarly, $s(b)$ and $s(c)$ both are 20, and $s(e)$ is 80. Trait a is eliminated giving (ix). Here, $s(b)$ and $s(e)$ are each 16, while $s(c)$ is 8. Eliminating trait c yields (x). Now, $s(b)$ and $s(c)$ both are 8. Eliminating, say b, gives (xi). Lastly, eliminating trait e yields (xii). Thus, DNs can represent MW determining elimination order $\sigma = (a, c, b, e)$.*

4.3.3 Representing Min-Fill in DNs

Definition 4.3. *To represent MF, the score of a trait t is the number of pairs of traits that are related to t, but are unrelated themselves.*

Example 33. *The traits related to a in $\mathcal{D}$ are b, c, d, and e. Traits b and c are unrelated, as are b and d. Thus, $s(a)$ is 2. Similarly, $s(b)$ is 1, $s(c)$ is 0, and $s(e)$ is 5. Eliminating c gives Figure 4.2 (xiii). Here, both $s(a)$ and $s(b)$ are 1, and $s(e)$ is 3. Eliminating, say b, yields (xiv). Now, $s(a)$ and $s(e)$ both are 1. Eliminating, say a, gives (xv). Then, e is eliminated leaving (xvi). Therefore, DNs can represent MF determining elimination order $\sigma = (c, b, a, e)$.*

4.3.4 Representing Weighted-Min-Fill in DNs

Definition 4.4. *To represent WMF, the score $s(t)$ of a trait t is the product of the energies of the pairs of traits related to t that are unrelated themselves.*

Example 34. *From Example 33, the two pairs of traits related to a that are unrelated themselves are b and c, and b and d. The product of the energies for b and c is 4, as is that for b and d. Thus, the score $s(a)$ of trait a is 8. Similarly, $s(b)$ is 10, $s(c)$ is 0, and $s(e)$ is 26. Eliminating c gives (iv). Here, $s(a)$ is 4, $s(b)$ is 10, and $s(e)$ is 18. Eliminating a yields (v). Now, $s(b)$ and $s(e)$ both are 4. Eliminating, say b, gives (vi). Finally, e is eliminated yielding (vii). Thus, DNs can represent WMF determining elimination order $\sigma = (c, a, b, e)$.*

4.4 Equivalence

The *undirected graph* $U(\mathcal{D})$ of a DN $\mathcal{D}$ has variables $T_c(\mathcal{D})$ and edges $\{(v_i, v_j) \mid p(C, D) \in \mathcal{D}$ and $v_i, v_j \in CD\}$, namely, $U(\mathcal{D})$ is the moralization $\mathcal{B}_m$ [7]. For instance, $U(\mathcal{D})$ of the DN $\mathcal{D}$ in Figure 4.2 (i) is the undirected graph $\mathcal{B}_m$ shown in Figure 4.1 (ii).

Lemma 6. *MN can be equivalently represented in DNs.*

Proof. Given $\mathcal{D}$ is the DN for a BN $\mathcal{B}$. Then $U(\mathcal{D})$ is the moralization $\mathcal{B}_m$. By construction, the scoring of traits in $\mathcal{D}$ is precisely the scoring traits in $\mathcal{B}_m$. Also by construction, the undirected graph $U(\mathcal{D})$ of the DN $\mathcal{D}$ obtained by eliminating a trait with a minimum score corresponds exactly to the undirected graph with the corresponding variable eliminated. Finally, the set of traits recursively eliminated is the set of variables recursively eliminated, since $\mathcal{D}' = \{p(X, Y)\}$ is the DN corresponding to the query $P(X|Y)$ posed to $\mathcal{B}$. □

Example 35. *The BN $\mathcal{B}$ in Figure 4.1 (i) is represented as the DN $\mathcal{D}$ in Figure 4.2 (i), $U(\mathcal{D})$ is the moralization $\mathcal{B}_m$ in Figure 4.1 (ii), and the given query $P(f|d)$ is represented by the DN $\mathcal{D}' = \{p(f, d)\}$ in Figure 4.2 (viii). The scoring of variable and traits precisely coincides, namely, $s(a) = 4$, $s(b) = 3$, $s(c) = 3$, and $s(e) = 5$. Eliminating variable c gives Figure 4.1 (iv), while eliminating trait c gives Figure 4.2 (iv), of which the undirected graph is Figure 4.1 (iv). Next, the scoring of variables and traits are the same, namely, both $s(a)$ and $s(b)$ are 3, and $s(e)$ is 4. Eliminating variable a yields Figure 4.1 (v), while eliminating trait a gives Figure 4.2 (v), of which the undirected graph is Figure 4.1 (v). Once again, the scoring is the same, i.e., $s(b)$ and $s(e)$ both are 3. Eliminating variable b gives Figure 4.1 (vi), while eliminating trait b gives Figure 4.2 (vi), of which the undirected graph is Figure 4.1 (vi). Finally, eliminating variable e yields Figure 4.1 (vii), while eliminating trait e gives Figure 4.2 (vii), of which the undirected graph is Figure 4.1 (vii). Therefore, both representations obtain the same elimination ordering $\sigma = (c, a, b, e)$.*

Lemma 7. *MW, MF, and WMF each can be equivalently represented in DNs.*

The proof of Lemma 7 is similar to that of Lemma 6.

4.5 Potential Energy Heuristic

We put forth a new heuristic, called *potential energy*, which is based upon the rich representation of DNs.

Recall that a DN $\mathcal{D}$ can represent a BN $\mathcal{B}$. The *energy of a population* $p(C, D)$ is the domain cardinality of the CPT to which it corresponds. For example, the energy of population $p(b, a)$ representing the CPT $P(b|a)$ is 10, since b is binary and the domain- cardinality of a is 5.

Definition 4.5. *In* potential energy *(PE), the score of a trait t is the sum of the population energies built by recursively merging the populations containing t.*

Example 36. *Consider using PE to score trait a in the DN $\mathcal{D}$ in Figure 4.2 (i). Merging $p(a)$ and $p(b, a)$ gives $p(ab)$ with energy 10. Then, merging $p(ab)$ and $p(e, acd)$ gives $p(abe, cd)$ with energy 80. Hence, $s(a)$ is 90.*

Note that PE is not necessarily unique, since it depends upon the order in which populations are merged.

Example 37. *In Example 36, first merge $p(a)$ with $p(e, acd)$, yielding $p(ae, cd)$ with energy 40. Now, merge $p(ae, cd)$ with $p(b, a)$, giving $p(abe, cd)$ with energy 80. Therefore, $s(a)$ is 120.*

Henceforth, populations will be recursively merged always using the two populations with the lowest energies. For example, $s(a)$ will be determined by merging $p(a)$ and $p(b, a)$ first as in Example 36 rather than $p(a)$ and $p(e, acd)$ as in Example 37.

Example 38. *Similar to $s(a)$ being 90 in Example 36, the scores of $s(b)$, $s(c)$, and $s(e)$ are 40, 40, and 160, respectively. Removing, say c, gives Figure 4.2 (xiii). The scores of $s(a)$, $s(b)$, and $s(e)$ now are 50, 40, and 80, respectively. Removing b yields Figure 4.2 (xiv). Here, $s(a)$ and $s(e)$ are each 40. Removing, say a, gives Figure 4.2 (xv), and then e, yields Figure 4.2 (xvi). Therefore, PE determines elimination ordering $\sigma = (c, b, a, e)$.*

4.6 Analysis

We show that PE is more refined than MN, MW, MF, and WMF in the following aspects: (i) scoring a variable; (ii) multiplying the probability tables; (iii) marginalizing the variable; and (iv) representing the remaining information.

(i) Scoring a variable. Consider, for example, the multiplications needed to eliminate variable a from the BN in Figure 4.1 (i):

$$P(a, b, e|c, d) \ = \ P(a) \cdot P(b|a) \cdot P(e|a, c, d). \tag{4.35}$$

Generally, MF and WMF tend to work better on more problems and, not surprisingly, WMF usually has the most significant gains when there is some significant variability in the sizes of variable domains [44]. This is because MF and WMF estimate the computation to take place in the RHS of (4.35), for example, by utilizing the RHS itself. PE also considers the RHS of (4.35), but in more detail than both MF and WMF. PE counts the number of multiplications that can be used to compute the product of all probability tables involving the variable being scored.

Example 39. *In Example 36, PE scores a as 90, since computing the product of the RHS of (4.35) takes 10 multiplications for $p(a)$ and $p(b|a)$, followed by 80 multiplications for $p(a, b)$ and $p(e, acd)$. Thus, a PE score $s(a)$ of 90 means that 90 multiplications can be used to compute (4.35).*

(ii) Multiplying the probability tables. Once a variable with a minimum score is chosen, DNs more accurately depict the multiplication of its probability tables.

Example 40. *Suppose c is the first variable to be eliminated in our running example. In the moralization $\mathcal{B}_m$ of Figure 4.1 (ii), edges are added between all of c's neighbours, yielding $\mathcal{B}_m$ itself. Thus, by using undirected graphs to model multiplication, no change was made in the graphical representation even though the following multiplication takes place:*

$$P(c, e|a, d) = P(c) \cdot P(e|a, c, d). \tag{4.36}$$

In stark contrast, PE explicitly represents this multiplication by merging populations $p(e)$ and $p(e, acd)$ in Figure 4.2 (i), yielding population $p(ce, ad)$ in Figure 4.2 (ii).

Example 40 illustrates how merging in DNs is a more descriptive graphical representation of multiplication compared to adding edges in a undirected graph.

(iii) Marginalizing the variable. Once the appropriate probability tables have been multiplied, DNs more accurately depict the subsequent marginalization.

Example 41. *Continuing from Example 40, variable* c *is marginalized in BN inference as follows:*

$$P(e|a,d) \;=\; \sum_{c} P(c,e|a,d). \tag{4.37}$$

It is not obvious how newly created CPT $P(e|a,d)$ *is graphically represented by removing* c *and its incident edges* (a,c), (c,d), *and* (c,e) *from Figure 4.1 (ii) yielding the undirected graph in Figure 4.1 (iv). On the other hand, PE clearly articulates this marginalization by replicating population* $p(ce,ad)$ *in Figure 4.2 (ii) as itself and population* $p(e,ad)$ *in Figure 4.2 (iii), and then letting natural selection remove spent population* $p(ce,ad)$ *in Figure 4.2 (iv).*

The key point of Example 41 is how replication and natural selection in DNs provide a better graphical description of marginalization in BN inference than the deletion of the variable being marginalized and its incident edges from an undirected graph.

(iv) Representing the remaining information. After a variable with a minimum score is eliminated, PE maintains a one-to-one correspondence between the remaining variables and populations.

Example 42. *Continuing from Example 41,* c*'s elimination results in Figure 1 (iv). It is unclear how this undirected graph corresponds to the remaining probability tables:*

$$P(a), P(b|a), P(d), P(e|a,d), \text{ and } P(f|b,e). \tag{4.38}$$

On the contrary, in PE, there is a one-to-one correspondence between the probabilities tables in (4.38) and populations in Figure 4.2 (iv):

$$p(a), p(b,a), p(d), p(e,ad), \text{ and } p(f,be). \tag{4.39}$$

Similar remarks hold after the elimination of variables a, b, *and* e, *in (v), (vi), and (vii) of Figure 4.2, respectively.*

4.7 Conclusion

We have shown how four well-known heuristics for determining elimination orderings can be equivalently represented in DNs, a richer representation of BNs. We proposed PE as a novel heuristic based on DNs themselves. Our analysis has shown that PE is a more refined heuristic in four aspects. Future work will include an empirical evaluation.

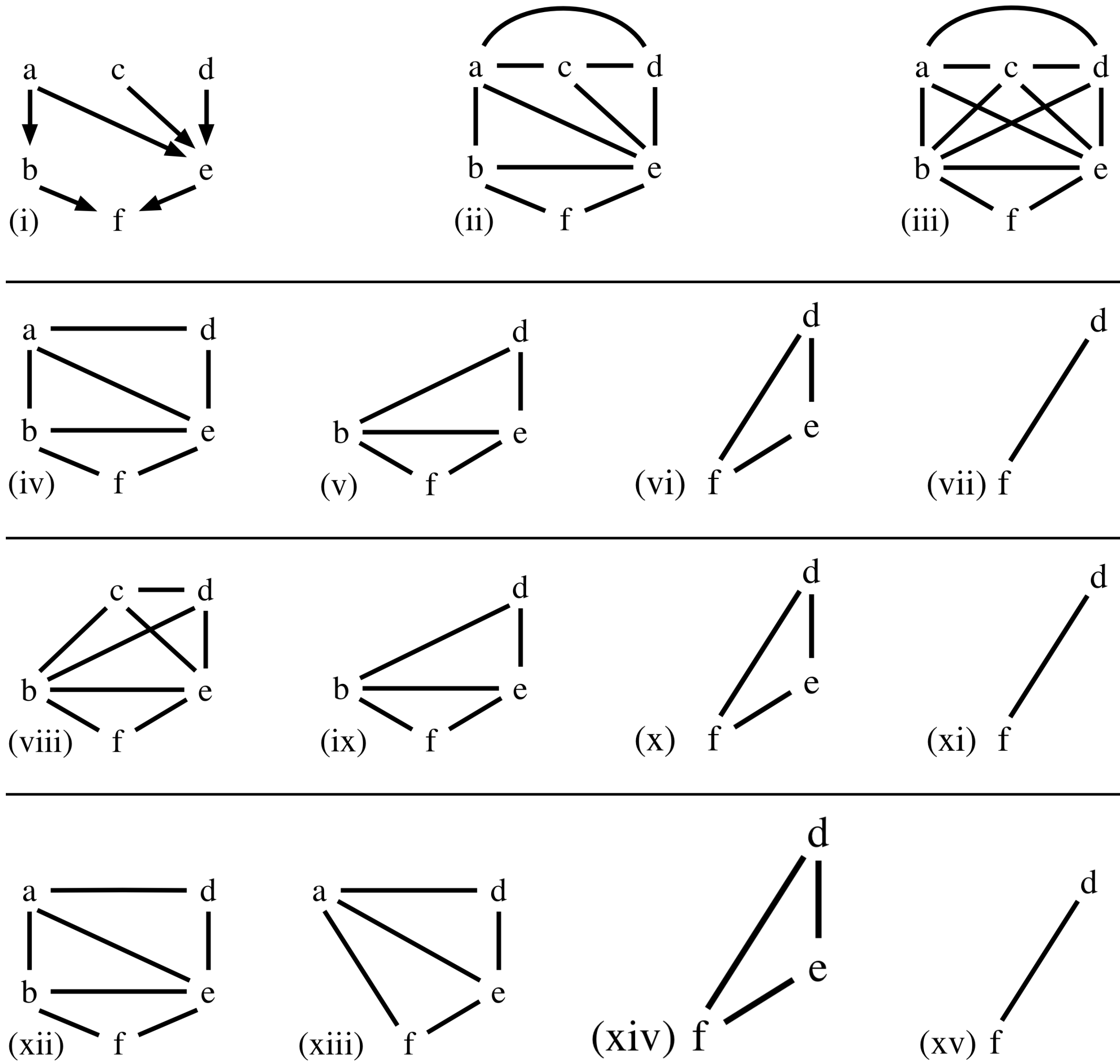


Figure 4.1: (i) a BN $\mathcal{B}$. (ii) the moralization $\mathcal{B}_m$. (iii) adding edges between a's neighbours in $\mathcal{B}_m$. (iv)-(vii) MN and WMF can determine $\sigma = (c, a, b, e)$. (viii)-(xi) MW can determine $\sigma = (a, c, b, e)$. (xii)-(xv) MF can determine $\sigma = (c, b, a, e)$.

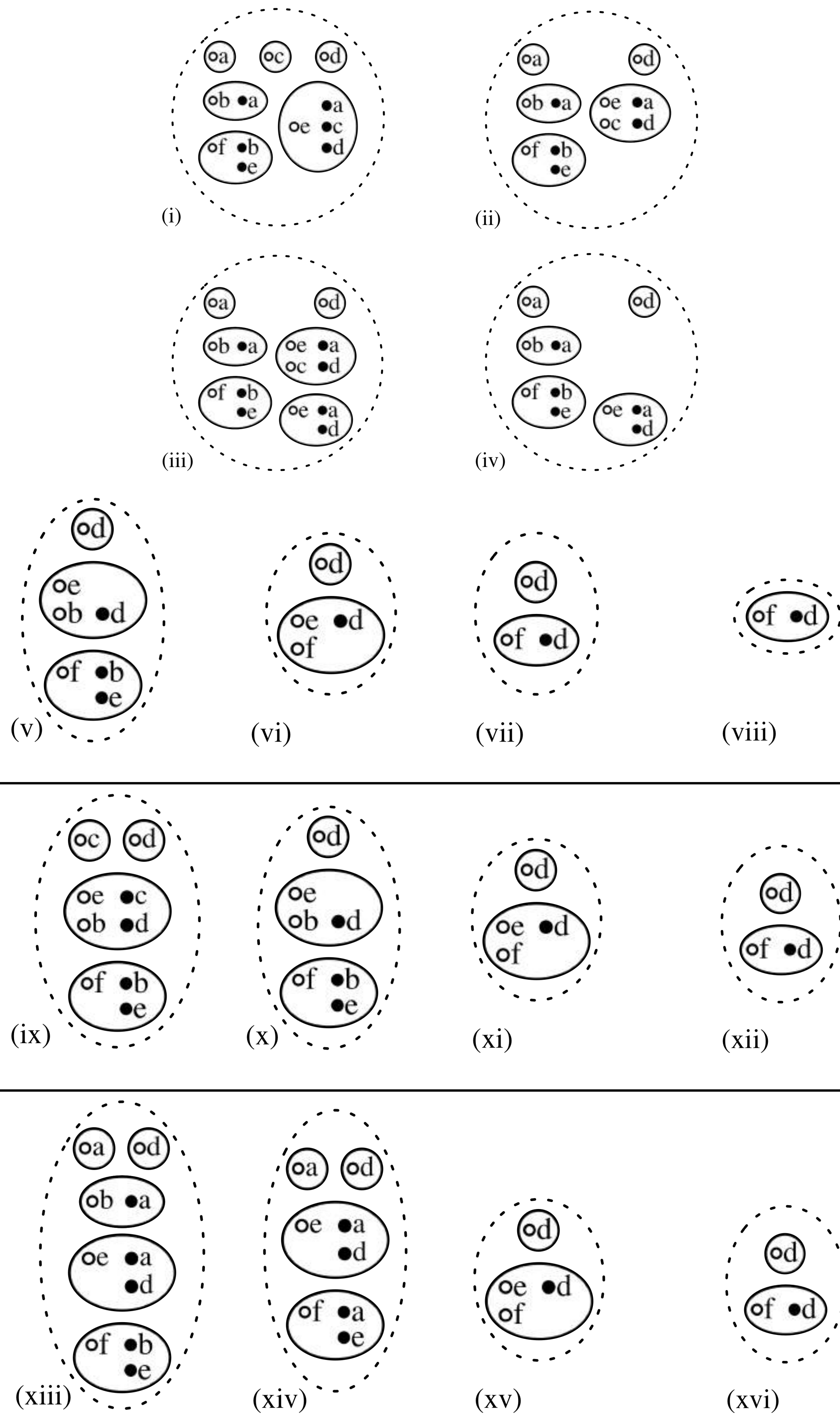


Figure 4.2: DNs $\mathcal{D}$ in (i) and $\mathcal{D}'$ in (viii). (ii)-(iv) eliminating trait c by merging, replication, and natural selection, respectively. (ii)-(vii) determines $\sigma = (c, a, b, e)$. (ix)-(xii) determines $\sigma = (a, c, b, e)$. (xiii)-(xvi) determines $\sigma = (c, b, a, e)$.

Chapter 5

Relevant Path Separation: A Faster Method for Testing Independencies in Bayesian Networks

5.1 Introduction

Directed separation (d-separation) [61] continues to be useful in a wide range of areas, including causal inference in statistics [63], cause and correlation in biology [77], extrapolation across populations [64], handling missing data [55], bioinformatics [56], and deep learning [34]. The d-separation algorithm is a graphical method for determining which *conditional independence* relations are implied by the *directed acyclic graph* (DAG) of a *Bayesian network* (BN) [61]. With respect to a given independence to be tested, current implementations, including Bayes-Ball [74] and Reachable [44], find all nodes reachable along active paths, called the *active* part of a BN. This approach overlooks a crucial property of d-separation.

The work [46] proposed *m-separation* as another method for testing independencies in BNs. In the proof of correctness, it is established that all active paths of interest can only appear in what we call the *relevant* part of a BN. This property warrants attention in itself, since it has both theoretical and practical ramifications.

In this chapter, we propose *relevant path separation* (rp-separation) as a new method for testing independencies in BNs. The salient feature of rp-separation is that it explores the *intersection* between the active and relevant parts of a BN. We introduce the notion of a *relevant* path and establish that *irrelevant* paths are either

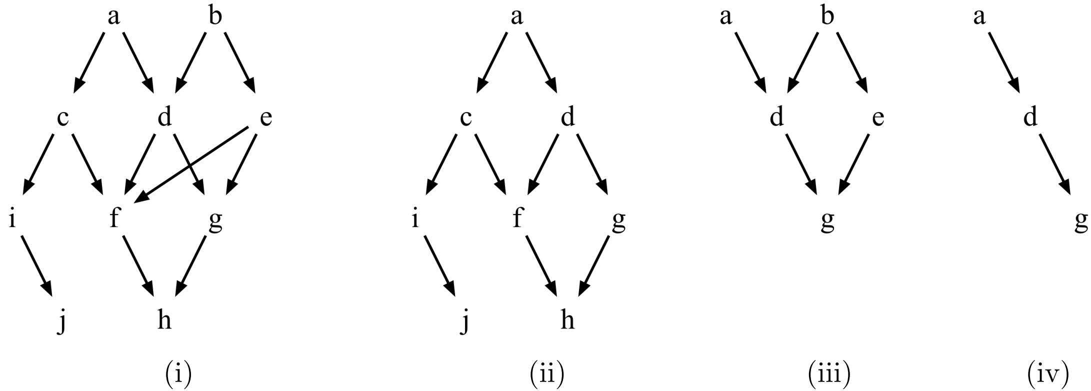


Figure 5.1: (i) testing independence $I(a, e, g)$ in a Bayesian network $\mathcal{B}$; (ii) the active part of $\mathcal{B}$; (iii) the relevant part of $\mathcal{B}$; (iv) the intersection of the active and relevant parts.

active paths that are doomed to become blocked or active paths that terminate before reaching a variable of interest. Rather than exploring all active paths, rp-separation displays impressive performance in practice by only exploring active paths that are relevant. In real-world or benchmark BNs, rp-separation is faster in 17 of 19 cases with an average time savings of 53%, culminating with being nearly twice as fast in the largest BN.

To test whether two sets X and Z of variables are conditionally independent in BN $\mathcal{B}$ given a third set Y of variables, denoted $I(X, Y, Z)$, it is shown that all irrelevant paths include at least one variable not in $XYZ \cup An(XYZ)$, where $An(XYZ)$ are the ancestors of XYZ in $\mathcal{B}$. Thus, the variables considered by rp-separation exactly coincide with those considered by m-separation. However, m-separation builds a secondary structure (an undirected graph), a process that involves adding and deleting undirected edges. We explicitly demonstrate several ways in which this computation is redundant and present a refinement of m-separation. Our experimental results show that the refinement leads to a time savings of 27% on average, but even the refined m-separation is slower than d-separation, let alone rp-separation.

5.2 Background

The work [29] was the first to provide a linear time complexity algorithm for implementing d-separation. Their method, however, always explores the entire DAG. *Bayes-Ball* (BB) [74] only explores the *active* part of the DAG. As will be discussed in Section 5.6, BB works using a hypothetical ball bouncing in a DAG and can be used for purposes outside the scope of our paper. Instead, we use the REACHABLE algorithm [44], since it also explores the active part of a DAG, but with d-separation terminology. To test $I(X, Y, Z)$ in a BN $\mathcal{B}$, REACHABLE takes X, Y, and $\mathcal{B}$ as input and returns the set of all variables reachable from X along active paths. For pedagogical purposes, Example 43 will mention active paths explicitly.

Example 43. *Let us test $I(a, e, g)$ in the BN $\mathcal{B}$ of Figure 5.1 (i) using* REACHABLE. *Given $X = \{a\}$, $Y = \{e\}$, and $\mathcal{B}$, the set of variables reachable from X along active paths is denoted by R and initialized as $R = \{a\}$. As variable c is reachable from variable a along active path (a, c), the set of reachable variables is updated as $R = \{a, c\}$. Similarly, variable f can be reached along the active path $(a, c), (c, f)$. Subsequently, variable h can be reached along active path $(a, c), (c, f), (f, h)$ meaning R is updated as:*

$$R = \{a, c, f, h\}. \tag{5.40}$$

Variables i and j are reachable along active paths $(a, c), (c, i)$ and $(a, c), (c, i), (i, j)$, respectively. Hence, R is updated as:

$$R = \{a, c, f, h, i, j\}. \tag{5.41}$$

As depicted by the active part of $\mathcal{B}$ in Figure 5.1 (ii), it can be verified that the set of all variables reachable from X along active paths is $R = \{a, c, f, h, i, j, d, g\}$. Since at least one variable in Z was reachable from X along an active path, the independence $I(a, e, g)$ does not hold.

Example 43 and Figure 5.1 (ii) illustrate how REACHABLE finds all variables reachable along all active paths. Although linear, this approach can be wasteful.

5.3 Relevant Path Separation

Our purpose here is to give a faster method for testing independencies in BNs in practice. We begin by motivating our study.

m-Separation [46] is another method for testing independencies in BNs. Let X, Y, and Z be pairwise disjoint sets of variables in a BN $\mathcal{B}$. m-Separation tests $I(X,Y,Z)$ with four steps: (i) prune $\mathcal{B}$ onto $XYZ \cup An(XYZ)$; (ii) construct the *moralization* [47] of the sub-DAG constructed in step (i) by adding an undirected edge between each pair of parents of a common child and then dropping directionality; (iii) delete Y and its incident edges from the undirected graph built in (ii); and (iv) if there exists a path from (any variable in) X to (any variable in) Z in (iii), then $I(X,Y,Z)$ does not hold; otherwise, $I(X,Y,Z)$ holds.

Theorem 5.1 shows m-separation is correct by establishing equivalence with d-separation.

Theorem 5.1. *[46] Let X, Y, and Z be disjoint subsets of a DAG $\mathcal{B}$. Then, Y d-separates X from Z in $\mathcal{B}$ if and only if Y separates X from Z in the moralization of the sub-DAG of $\mathcal{B}$ onto $XYZ \cup An(XYZ)$.*

Proof. ($\Leftarrow$) By contraposition, suppose Y does not d-separate X from Z. Then there exists an active path from X to Z. Every sequential variable on this path must be not in Y; otherwise, it would block the path. Similarly, every divergent variable on this path must be not in Y. On the contrary, every convergent variable on this path must either be in Y or have a descendent in Y; otherwise, the path is blocked. It follows that every variable on this active path is in $XYZ \cup An(XYZ)$. Hence, every variable in this path will appear in the moralized graph constructed onto $XYZ \cup An(XYZ)$. The moralization step will add an edge for each convergent variable on the active path, thus creating a path from X to Z not involving Y.

($\Rightarrow$) See [46]. □

A key property of d-separation involving active paths is mentioned in the proof of Theorem 5.1.

Theorem 5.2. *Given $I(X,Y,Z)$ to be tested in a BN $\mathcal{B}$, all active paths between X and Z can only involve variables in $XYZ \cup An(XYZ)$.*

Proof. Follows immediately from the proof of Theorem 5.1. □

Let $I(X,Y,Z)$ be an independence to be tested in a BN $\mathcal{B}$. A path from (any variable in) X to (any variable in) Z is called *relevant*, if it only involves variables in $XYZ \cup An(XYZ)$; otherwise, it is called *irrelevant*. The *relevant part* of $\mathcal{B}$ is its sub-DAG onto $XYZ \cup An(XYZ)$.

Example 44. *Consider testing $I(a,e,g)$ in the BN $\mathcal{B}$ of Figure 5.1 (i). Then, $XYZ \cup An(XYZ) = \{a,e,g\} \cup \{a,b,d,e\} = \{a,b,d,e,g\}$. There are only two relevant paths. The path (a,d), (d,b), (b,e), (e,g) is relevant, since it only involves variables in $\{a,b,d,e,g\}$. Similarly, the path (a,d), (d,g) is relevant. On the contrary, the path (a,c), (c,f), (f,h), (h,g) is irrelevant, since it involves variables $c,f,h \notin \{a,b,d,e,g\}$. Moreover, the path (a,c), (c,i), (i,j) is irrelevant as $c,i,j \notin \{a,b,d,e,g\}$. The relevant part of $\mathcal{B}$ is shown in Figure 5.1 (iii).*

We now formally introduce a new method for testing independencies in BNs.

Definition 5.1. *Let X, Y, and Z be pairwise disjoint sets of variables in a BN $\mathcal{B}$. The independence test $I(X,Y,Z)$ holds if all relevant paths from (any variable in) X to (any variable in) Z are blocked; otherwise, $I(X,Y,Z)$ does not hold.*

We call the method *relevant path separation* (rp-separation) to emphasize that only the relevant part of a BN needs to be considered.

Example 45. *Let us test $I(a,e,g)$ in the BN $\mathcal{B}$ of Figure 5.1 (i) using rp-separation. Given $X = \{a\}$, $Y = \{e\}$, and $Z = \{g\}$, then*

$$XYZ \cup An(XYZ) \quad = \quad \{a,b,d,e,g\}. \tag{5.42}$$

There are only two relevant paths from X to Z to consider. The relevant path $(a,d),(d,g)$ is active, while the relevant path $(a,d),(d,b),(b,e),(e,g)$ is blocked as d is a closed convergent variable. Therefore, $I(a,e,g)$ does not hold in $\mathcal{B}$ by rp-separation.

We now show the correctness of rp-separation.

Theorem 5.3. *Independence $I(X,Y,Z)$ holds in a BN $\mathcal{B}$ by d-separation if and only if $I(X,Y,Z)$ holds in $\mathcal{B}$ by rp-separation.*

Proof. Any path involving a variable not in $XYZ \cup An(XYZ)$ is ignored by rp-separation. Thereby, rp-separation can be seen as applying d-separation on $\mathcal{B}'$, where $\mathcal{B}'$ is the sub-DAG of $\mathcal{B}$ onto variables $XYZ \cup An(XYZ)$. However, testing $I(X, Y, Z)$ in $\mathcal{B}$ is equivalent to testing $I(X, Y, Z)$ in $\mathcal{B}'$ [46]. Therefore, the claim holds. □

Although rp-separation only considers the relevant part of a BN, an efficient implementation, called RP-Reachable and given as Algorithm 4, traverses only the active paths within the relevant part. It is based upon the Reachable algorithm in [44], which instead traverses the active part of a BN. More specifically, given $I(X, Y, Z)$ in a BN $\mathcal{B}$, Algorithm 4 takes X, Y, Z, and $\mathcal{B}$ as input and first marks those variables in $XYZ \cup An(XYZ)$. The algorithm keeps track of whether a variable v is visited from a child, denoted $(\uparrow, v)$, or visited from a parent, denoted $(\downarrow, v)$. In Algorithm 4, L is the set of variables to be visited, R is the set of reachable variables via active paths, and V is the set of variables that have been visited. Active paths from X are explored as long as they only involve variables in $XYZ \cup An(XYZ)$.

Example 46. *Let us test $I(a, e, g)$ in the BN $\mathcal{B}$ of Figure 5.1 (i) using* RP-Reachable. *The initialization sets $A = \{b, e\}$, $XYZ^{up} = \{a, b, d, e, g\}$, $L = \{(\uparrow, a)\}$, $V = \emptyset$, and $R = \emptyset$. The first iteration of the* while *loop in line 8 considers $(\uparrow, a)$. Next, $R = \{a\}$ and $V = \{(\uparrow, a)\}$. Line 19 considers children c and d of a. However, the check for relevant paths on line 20 results in $L = \{(\downarrow, d)\}$. That is, traversing from a to c is ignored because (a, c) is an irrelevant path. Similarly, when considering $(\downarrow, d)$ on the second iteration, line 19 considers children f and g of d. However, line 20 only adds $(\downarrow, g)$ and not $(\downarrow, f)$. The remainder follows similarly, yielding $R = \{a, d, g\}$.*

Example 46 shows that RP-Reachable only explores the intersection depicted in Figure 5.1 (iv).

Theorem 5.4. *Algorithm 4 is linear in the number of directed edges in a BN $\mathcal{B}$.*

Proof. Consider $I(X, Y, Z)$ to be tested in a BN $\mathcal{B}$. Algorithm 4 extends Reachable by computing $XYZ \cup An(XYZ)$ in line 4. Algorithm 4 also differs from Reachable in lines 19, 22, 27, and 31 by checking whether a variable is in $XYZ \cup An(XYZ)$. As Reachable computes ancestral sets and tests set membership [44], the claim follows. □

5.4 Experimental Results and Analysis

We report on an empirical comparison of Reachable and RP-Reachable. Both methods were implemented in Python using the *NetworkX* library (see `networkx.github.io`). The experiments were conducted on a 2.3 GHz Inter Core i7 with 8 GB RAM. The evaluation was carried out on 19 real-world or benchmark BNs listed in the first column of Table 5.1. The second column of Table 5.1 reports the number of variables in each BN. For each BN, 1000 independencies $I(X, Y, Z)$ were randomly generated. Following Reachable in [44], where X is a singleton set, in our experiments X, Y, and Z are kept to being singleton sets. Then $I(X, Y, Z)$ is tested by Reachable and RP-Reachable. The average time in seconds required by Reachable and RP-Reachable are reported in the third and fourth columns, respectively. Time savings by RP-Reachable over Reachable is shown in the fifth column. Table 5.1 shows that RP-Reachable was faster than Reachable on average by 53%.

Lemma 8 and Theorem 5.5 analyze why RP-Reachable tends to be faster in Table 5.1.

Lemma 8. *Given an independence $I(X, Y, Z)$ in a BN $\mathcal{B}$, the set of variables reachable from X along active paths in* RP-Reachable *is a subset of those determined in* Reachable*.*

In Example 46, RP-Reachable determined $R = \{a, d, g\}$, as depicted in Figure 5.1 (iv), whereas, in Example 43, Reachable determined $R = \{a, c, f, h, i, j, d, g\}$, as shown in Figure 5.1 (ii).

Theorem 5.5. *Consider testing $I(X, Y, Z)$ in a BN $\mathcal{B}$. If* Reachable *traverses an irrelevant path, then either the path will remain active and never reach a variable in $XYZ \cup An(XYZ)$ or it will necessarily become blocked by a closed convergent variable not in $XYZ \cup An(XYZ)$.*

The proofs of Lemma 8 and Theorem 5.5 will be provided in an extended paper.

Example 47. *Recall Example 43 where $I(a, e, g)$ is tested using* Reachable*. The active path (a, c), (c, f), (f, h) is explored, giving (5.40). However, spending time exploring this irrelevant path is wasteful, since (a, c), (c, f), (f, h), (h, g) is blocked*

Table 5.1: Comparison of REACHABLE and RP-REACHABLE with 1000 randomly generated independencies $I(X, Y, Z)$ in each BN. Average times are reported in seconds and winners in bold.

BN	Vars	REACHABLE	RP-REACHABLE	savings over REACHABLE
child	20	1.27E-04	**1.14E-04**	10%
insurance	27	1.93E-04	**1.77E-04**	8%
water	32	1.54E-04	**1.38E-04**	10%
mildew	35	**1.62E-04**	1.70E-04	-5%
alarm	37	**1.39E-04**	1.42E-04	-2%
barley	48	2.51E-04	**2.40E-04**	4%
hailfinder	56	1.70E-04	**1.52E-04**	11%
hepar2	70	3.14E-04	**1.95E-04**	38%
win95pts	76	1.21E-04	**9.90E-05**	18%
pathfinder	109	4.84E-04	**1.19E-04**	75%
munin1	186	5.70E-04	**2.60E-04**	54%
andes	223	8.21E-04	**6.81E-04**	17%
diabetes	413	2.38E-03	**2.23E-03**	6%
pigs	441	3.67E-04	**1.11E-04**	70%
link	724	1.20E-03	**3.70E-04**	69%
munin2	1003	7.66E-04	**1.95E-04**	75%
munin4	1038	1.57E-03	**2.47E-04**	84%
munin	1041	1.53E-03	**2.44E-04**	84%
munin3	1041	1.77E-03	**2.45E-04**	86%
Average Time		6.89E-04	**3.23E-04**	53%

by h, a closed convergent variable outside of $\{a, b, d, e, g\}$, as guaranteed by Theorem 5.5. Furthermore, REACHABLE *explores the active path $(a, c), (c, i), (i, j)$ as indicated by (5.41). Again, time is wasted exploring this irrelevant path, since Theorem 5.5 ensures that such an active path must terminate before reaching g.*

The important point when testing $I(X, Y, Z)$ is that no active path from X to Z can involve a variable outside of $XYZ \cup An(XYZ)$. For instance, in Example 47, no active path from a to g involves variables c, f, h, i or j. Therefore, variables that are reachable via active paths, but that are not in $XYZ \cup An(XYZ)$, can be safely ignored.

Besides a practical advantage, Theorems 5.2 and 5.5 provide a clearer description of d-separation. When explaining the test of independence $I(X, Y, Z)$, the textbooks [44, 17, 63, 77, 43, 39] all discuss traversing at least one irrelevant path from

X to Z without mentioning that this path will necessarily become blocked. This is undesirable, since the d-separation test of $I(X, Y, Z)$ is to decide whether or not there exists an active path from X to Z. Testing irrelevant paths from X to Z does not aid in answering this question. For improved clarity, only relevant paths should be tested. For instance, in Example 45, the relevant path $(a, d), (d, g)$ is active, while the relevant path $(a, d), (d, b), (b, e), (e, g)$ is blocked.

5.4.1 Variations of rp-Separation

There are two main ways in which rp-separation can be implemented in practice. One approach is to mark and ignore those variables in the BN $\mathcal{B}$ not contained in $XYZ \cup An(XYZ)$. A second approach is to prune $\mathcal{B}$ onto $XYZ \cup An(XYZ)$, yielding a sub-BN $\mathcal{B}'$.

The third and fourth columns of Table 5.2 show the running times of both approaches to implementing rp-separation following the same setup as described for Table 5.1. The third column of Table 5.2 is for the implementation of rp-separation by marking and ignoring variables not in $XYZ \cup An(XYZ)$. The fourth column is for the implementation of rp-separation on a sub-BN $\mathcal{B}'$ built by pruning the BN $\mathcal{B}$ onto $XYZ \cup An(XYZ)$. Without exception, the marking approach was faster than the pruning approach, as highlighted by winning times in bold. The average time savings of marking over pruning is 20%.

5.5 Refining m-Separation

Here, we analyze and refine m-separation. We begin with an example of m-separation.

Example 48. *Consider testing $I(a, e, g)$ using m-separation in the BN $\mathcal{B}$ of Figure 5.1 (i). In step (i), the sub-DAG constructed onto $\{a, e, g\} \cup An(\{a, e, g\}) = \{a, b, d, e, g\}$ is depicted in Figure 5.2 (i). In step (ii), the moralization of the sub-DAG is depicted in Figure 5.2 (ii), where undirected edges (a, b) and (d, e) were added and then directionality was dropped. Edge (a, b) was added as a and b share common child d. Similarly, d and e share common child g. In step (iii), variable e and its incident edges (b, e), (d, e), and (e, g) are deleted, yielding the undirected graph in Figure 5.2*

Table 5.2: In each BN, 1000 independencies are randomly generated. The third and forth columns indicate whether rp-separation is faster in practice using marking or pruning. The fifth and sixth columns compare m-separation and refined m-separation. Average times are reported in seconds and winners in bold.

BN	Vars	rp-separation by marking	rp-separation by pruning	m-separation	refined m-separation
child	20	**1.14E-04**	1.37E-04	2.33E-04	**2.11E-04**
insurance	27	**1.77E-04**	2.12E-04	4.65E-04	**3.61E-04**
water	32	**1.38E-04**	1.77E-04	5.29E-04	**3.56E-04**
mildew	35	**1.70E-04**	2.16E-04	5.57E-04	**3.98E-04**
alarm	37	**1.42E-04**	1.82E-04	4.25E-04	**3.33E-04**
barley	48	**2.40E-04**	2.94E-04	7.17E-04	**5.25E-04**
hailfinder	56	**1.52E-04**	1.98E-04	4.94E-04	**3.76E-04**
hepar2	70	**1.95E-04**	2.40E-04	5.53E-04	**4.19E-04**
win95pts	76	**9.90E-05**	1.35E-04	4.07E-04	**2.90E-04**
pathfinder	109	**1.19E-04**	1.43E-04	2.86E-04	**2.42E-04**
munin1	186	**2.60E-04**	3.29E-04	7.92E-04	**6.00E-04**
andes	223	**6.81E-04**	8.54E-04	2.49E-03	**1.59E-03**
diabetes	413	**2.23E-03**	2.71E-03	5.93E-03	**4.33E-03**
pigs	441	**1.11E-04**	1.51E-04	3.75E-04	**3.06E-04**
link	724	**3.70E-04**	4.92E-04	1.46E-03	**9.73E-04**
munin2	1003	**1.95E-04**	2.68E-04	7.19E-04	**5.57E-04**
munin4	1038	**2.47E-04**	3.26E-04	8.35E-04	**6.29E-04**
munin	1041	**2.44E-04**	3.22E-04	8.19E-04	**6.19E-04**
munin3	1041	**2.45E-04**	3.22E-04	7.84E-04	**6.04E-04**
Average Time		**3.23E-04**	4.06E-04	9.93E-04	**7.22E-04**

(iii). Since there exists a path from a to g in Figure 5.2 (iii), $I(a, e, g)$ does not hold in $\mathcal{B}$ by m-separation.

The complexity of m-separation is $O(|E|^2)$, where E is the number of directed edges in the BN [29]. Although m-separation only considers relevant paths, it is lacking for two other reasons. First, it builds a sub-DAG in step (i) by pruning the given BN. The results in the third and fourth columns of Table 5.2 suggest that a pruning step is slower than a marking step. Second, steps (ii) and (iii) can be excessive.

Example 49. *Adding edge (a, b) is wasteful in Figure 5.2 (ii), since there already exists a path (a, d), (d, b) between a and b in the undirected graph in Figure 5.2 (iii).*

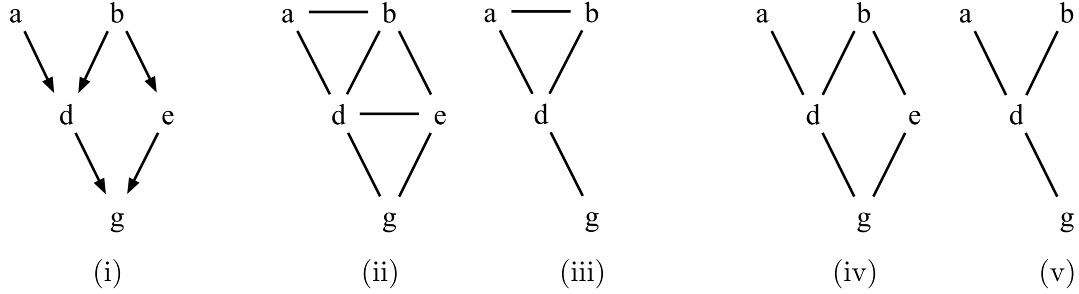


Figure 5.2: Testing $I(a, e, g)$ in the BN $\mathcal{B}$ in Figure 5.1 (i) with m-separation in (i)-(iii) and with refined m-separation in (i), (iv), and (v).

Example 49 shows that m-separation can add edges that do not change connectivity.

Example 50. *Adding edge (d, e) is wasteful in Figure 5.2 (ii), since (d, e) is deleted in Figure 5.2 (iii).*

Example 50 demonstrates that m-separation can add an edge in step (ii) only to delete it in step (iii). We now put forth a refinement of m-separation.

Let X, Y, and Z be pairwise disjoint sets of variables in a BN $\mathcal{B}$. Then, refined m-separation tests $I(X, Y, Z)$ with four steps: (i) mark and ignore variables of $\mathcal{B}$ not in $XYZ \cup An(XYZ)$; (ii) add undirected edge (v_i, v_j) between parents v_i and v_j of a common child v_k, only if $v_k \in Y$ and $v_i, v_j \notin Y$; (iii) delete Y and its incident edges from the undirected graph built in (ii); and (iv) if there exists a path from (any variable in) X to (any variable in) Z in (iii), then $I(X, Y, Z)$ does not hold; otherwise, $I(X, Y, Z)$ holds.

Example 51. *Consider testing $I(a, e, g)$ in the BN $\mathcal{B}$ of Figure 5.1 (i) using refined m-separation. In step (i), the variables under consideration are $\{a, e, g\} \cup \{a, b, d, e\} = \{a, b, d, e, g\}$ as illustrated in Figure 5.2 (i). In step (ii), variables a and b have variable d as common child, but $d \notin Y$. Similarly, variables d and e have common child $g \notin Y$. Thus, no undirected edges are added in step (ii), as depicted in Figure 5.2 (iv), where directionality has been dropped. Variable e and its incident edges (b, e) and (e, g) are deleted in step (iii), yielding Figure 5.2 (v). Since there exists a path from a to g in Figure 5.2 (v), $I(a, e, g)$ does not hold in $\mathcal{B}$ by refined m-separation.*

Example 51 underscores the advantage of the refined m-separation algorithm. It only adds required edges and it never adds an edge in step (ii) that will be subsequently deleted in step (iii).

The fifth and sixth columns of Table 5.2 provide the results of an empirical comparison between m-separation and refined m-separation. The experiments were conducted in the same manner as those in Table 5.1. The refined m-separation algorithm was faster than m-separation in all cases. The time savings were 27% on average. Furthermore, as the same randomly generated independencies were used in Tables 5.1 and 5.2, it can be seen that refined m-separation is slower than using REACHABLE, let alone RP-REACHABLE.

It is worth mentioning here that moralization is ideally suited when applied for building a *join tree* [47] for a given DAG. For this purpose, moralization ensures that a clique containing all variables in each CPT of the BN is formed in the undirected graph. On the contrary, moralization can be overkill when applied for testing independencies in BNs.

5.6 Related Work

Although the motivation and focus of this chapter is on m-separation and d-separation, other methods for testing independencies also consider irrelevant active paths.

Bayes-Ball (BB) [74] is a simple algorithm that can be applied for testing independencies and for determining requisite information in decision problems. Whereas [29] runs in time linear in the size of the DAG, BB runs in time linear in the size of the *active* part of the DAG. Although the efficiency gain is modest (sub-linear versus linear), the computation of requisite information is performed at the same time as irrelevance is determined. However, when applied for testing independencies, we will show that BB can explore irrelevant active paths.

Due to space limitations, we overview BB and refer the reader to [74] for details. To test $I(X, Y, Z)$ in a BN $\mathcal{B}$, BB takes X and Y as input. It sends a bouncing ball to visit variables in $\mathcal{B}$ starting from X. Depending on the type of variable with respect to Y and the direction from which the ball came (from parents or from children), the ball can *pass through* the variable, *bounce back*, or be *blocked.* The ball behaviour

corresponds to reachability along active paths.

Example 52. *Let us test $I(a, e, g)$ in the BN $\mathcal{B}$ of Figure 5.1 (i) using BB. Starting at $X = \{a\}$, the ball can pass through c, pass through i, and bounce back at j. Moreover, the ball can pass through f, and then is blocked at h. The remainder of the example is immaterial to our discussion.*

Example 52 highlights the fact that BB explores irrelevant active paths. Although the path (a, c),(c, i),(i, j) is active, it is irrelevant as active paths from a to g can only involve variables in $\{a, e, g\} \cup An(\{a, e, g\})$. In addition, BB explores the irrelevant active path (a, c), (c, f), (f, h), (h, g), which is guaranteed by Theorem 5.5 to become blocked, by closed convergent variable h in this case.

The function dSep in the *ggm* library [52], written in the R programming language, is based upon an alternative method of testing independencies given in [81]. Although the test does not use Reachable per se, it can be seen as exploring irrelevant paths.

An empirical evaluation of RP-Reachable with BB and dSep remains as future work.

5.7 Conclusion

When testing independence $I(X, Y, Z)$ in a BN $\mathcal{B}$, the definition of d-separation considers the entire BN, namely, checking whether every path from X to Z is active or blocked. The method in [29] always checks every directed edge in $\mathcal{B}$, but still decides all active paths in linear time. The Bayes-Ball [74] and Reachable [44] algorithms do not necessarily explore the entire BN, i.e., they find all variables reachable from X along active paths. Here, we emphasize that $I(X, Y, Z)$ can be tested by considering only the *relevant* part of $\mathcal{B}$. The relevant part of $\mathcal{B}$ is its sub-DAG onto $XYZ \cup An(XYZ)$. Finally, we suggest RP-Reachable that only traverses the active paths within the relevant part. Figure **??** reflects these ideas using the test of independence $I(a, e, g)$. The experimental results in Table 5.1 show that RP-Reachable is 53% faster on average than Reachable in real-world or benchmark BNs. The work presented in this chapter was inspired by Darwinian networks [9].

Algorithm 2 Given an independence $I(X, Y, Z)$, RP-REACHABLE traverses all active paths from X within the relevant part of a BN $\mathcal{B}$.

```
 1: procedure RP-REACHABLE(X,Y,Z,B)
 2:     ▷ Initialization
 3:     A ← Y ∪ An(Y)
 4:     XYZ^up ← XYZ ∪ An(XYZ)                      ▷ The relevant part of B
 5:     for v ∈ X do                         ▷ (Variable,direction) to be visited
 6:         L ← L ∪ {(↑, v)}
 7:     V ← ∅                              ▷ (Variable,direction) marked as visited
 8:     R ← ∅                              ▷ Variables reachable via an active path
 9:     ▷ Starting from X traverse relevant paths that are active
10:     while L ≠ ∅ do                         ▷ While variables to be checked
11:         Select (d, v) in L
12:         L ← L − {(d, v)}
13:         if (d, v) ∉ V then          ▷ If v has not been visited from direction d
14:             if v ∉ Y then
15:                 R ← R ∪ {v}                                ▷ v is reachable
16:             V ← V ∪ {(d, v)}             ▷ Mark v as visited from direction d
17:             if d =↑ and v ∉ Y then
18:                 for v_i ∈ Pa(v) do                        ▷ v is open serial
19:                     if v_i ∈ XYZ^up then          ▷ Only explore relevant paths
20:                         L ← L ∪ {(↑, v_i)}
21:                 for v_i ∈ Ch(v) do                     ▷ v is open divergent
22:                     if v_i ∈ XYZ^up then          ▷ Only explore relevant paths
23:                         L ← L ∪ {(↓, v_i)}
24:             else if d =↓ then
25:                 if v ∉ Y then
26:                     for v_i ∈ Ch(v) do                    ▷ v is open serial
27:                         if v_i ∈ XYZ^up then      ▷ Only explore relevant paths
28:                             L ← L ∪ {(↓, v_i)}
29:                 if v ∈ A then
30:                     for v_i ∈ Pa(v) do                ▷ v is open convergent
31:                         if v_i ∈ XYZ^up then      ▷ Only explore relevant paths
32:                             L ← L ∪ {(↑, v_i)}
33:     return R       ▷ All variables reachable from X via active paths within the
    relevant part of B
```

Chapter 6

Exploiting Symmetry of Independence in d-Separation

6.1 Introduction

d-Separation [61] is perhaps the greatest contribution made in the founding of *Bayesian networks* (BNs) [62]. d-Separation is a graphical test of those probabilistic conditional independencies encoded in the *directed acyclic graph* (DAG) of a BN. One striking feature is that the independencies read from the DAG are guaranteed to hold in the joint probability distribution defined by the BN. Another salient characteristic is that d-separation can be implemented with linear complexity in the number of edges in the DAG. Consequently, d-separation continues to be useful in a wide range of areas, including causal inference in statistics [63], cause and correlation in biology [77], extrapolation across populations [64], cognition [57], handling missing data [55], bioinformatics [56], and deep learning [34]. Many variations have been studied for d-separation [17, 29, 44, 60, 46, 74]. Given disjoint pairwise sets X, Y, and Z of variables (nodes), the conditional independence of X and Z given Y, denoted $I(X,Y,Z)$, holds in a DAG, if there does not exist an *active* path from X to Z with respect to Y [61]. It is known that if $I(X,Y,Z)$ holds by d-separation in the DAG, then $I(X,Y,Z)$ holds in joint probability distribution defined by the BN.

Lauritzen et al. [46] have established that said active paths can only involve nodes in XYZ and their ancestors. Hence, one fundamental optimization step is to restrict attention to the sub-DAG defined on these variables. However, all the above implementations find R_X, the reachable nodes from X along active paths with respect to

Y, and then test whether

$$R_X \cap Z = \emptyset. \tag{6.43}$$

Thereby, all previous implementations ignore the fact that probabilistic conditional independence is *symmetric* [19].

In this chapter, we propose SYMMETRIC D-SEPARATION as the first d-separation implementation that exploits the symmetry of probabilistic conditional independence. Since $I(X, Y, Z)$ is equivalent to $I(Z, Y, X)$ by symmetry, the test can be answered by computing R_Z, the nodes reachable from Z along active paths with respect to Y, and then testing whether

$$R_Z \cap X = \emptyset. \tag{6.44}$$

It is important to realize that R_X and R_Z can contain different nodes and may have distinct cardinalities. Our analysis of reachability highlights that *v-structure* [44] nodes play an important role in this regard. More specifically, it may be better to approach observed v-structure nodes from the bottom. Approaching an observed v-structure node from a child blocks an active path, whereas an active path can continue when approaching an observed v-structure node from a parent. Hence, a measure, called *depth*, is suggested to decide whether the search should run from start to goal or from goal to start. One salient feature is that depth can be computed during the pruning optimization step of finding the ancestor set of XYZ. An empirical comparison is conducted against a clever implementation of d-separation suggested by [17], which we refer to as DARWICHE. The experimental results are promising in two aspects. The effectiveness increases with network size, as well as with the amount of observed evidence, culminating with an average time savings of 9% in the 9 largest BNs used in our experiments.

6.2 Darwiche's Implementation of d-Separation

Definition 6.1. *[61] A variable v_k is called a* v-structure *[44] in a DAG $\mathcal{D}$, if $\mathcal{D}$ contains directed edges (v_i, v_k) and (v_j, v_k), but not a directed edge between variables v_i and v_j.*

For example, D, G, and H are the 3 v-structure nodes in Figure 6.1i.

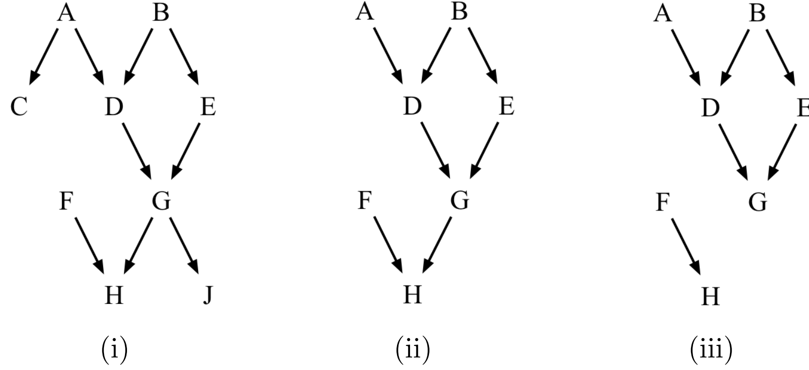


Figure 6.1: When testing $I(D, G, H)$ in Figure 6.1i, active paths between D and H with respect to G can only involve variables in the sub-DAG in Figure 6.1ii defined over $DGH \cup An(DGH)$. Darwiche's sub-DAG in Figure 6.1iii.

d-Separation [61] tests independencies in BNs and can be presented as follows [17]. Let X, Y, and Z be pairwise disjoint sets of variables in a BN $\mathcal{D}$. We say X and Z are *d-separated* by Y, denoted $I(X, Y, Z)$, if at least one variable on every undirected path from (any variable in) X to (any variable in) Z is closed. On a path, there are three kinds of variable v: (i) a *sequential* variable means v is a parent of one of its neighbours and a child of the other; (ii) a *divergent* variable is when v is a parent of both neighbours; and (iii) a *convergent* variable is when v is a child of both neighbours. A variable v is either open or closed. A sequential or divergent variable is *closed*, if $v \in Y$. A convergent variable is *closed*, if $(v \cup De(v)) \cap Y = \emptyset$. A path with a closed variable is *blocked*; otherwise, it is *active*.

A fundamental optimization step stems from Lemma 9.

Lemma 9. *[46] When testing $I(X, Y, Z)$ in a BN $\mathcal{D}$, an active path from X to Z with respect to Y can only involve nodes in $XYZ \cup An(XYZ)$.*

For example, consider testing $I(D, G, H)$ in the DAG shown in Figure 6.1i. As

$$\{D, G, H\} \cup An(D, G, H) = \{A, B, D, E, F, G, H\}, \tag{6.45}$$

the independence $I(D, G, H)$ can be safely tested in the sub-DAG shown in Figure 6.1ii.

Given an independence statement $I(X, Y, Z)$ to be tested in a BN $\mathcal{D}$. By R_X, we denote the set of nodes reachable from X along active paths with respect to Y. For

instance, when testing $I(D, G, H)$ in the pruned DAG in Figure 6.1ii, the reachable nodes from D are $R_D = \{A, B, E\}$.

Here, we review a clever implementation of d-separation as suggested by [17], which we will henceforth refer to as DARWICHE.

Theorem 6.1. *[17] Testing whether X and Z are d-separated by Y in DAG $\mathcal{D}$ is equivalent to testing whether X and Z are disconnected in a new DAG $\mathcal{D}'$, which is obtained by pruning $\mathcal{D}$ as follows:*

- *delete any leaf node W from $\mathcal{D}$ as long as W does not belong to $X \cup Y \cup Z$. This process is repeated until no more nodes can be deleted.*
- *delete all edges outgoing from nodes in Y.*

This implementation is clever in that its connectivity test ignores edge directions, yet is equivalent to testing active paths [17].

Example 53. *Consider testing $I(D, G, H)$ in the DAG $\mathcal{D}$ in Figure 6.1i using* DARWICHE. *Leaf nodes C and J are removed in the first pruning step of Theorem 6.1. Directed edge (G, H) is deleted in the second pruning step, yielding the sub-DAG in Figure 6.1iii. The nodes reachable from node D are*

$$R_D = \{A, B, D, E\}.$$

Therefore, $I(D, G, H)$ holds, since node H is not reachable from D.

6.3 d-Separation Exploiting Symmetry

In this section, we present a novel implementation of d-separation that exploits the symmetry of probabilistic conditional independence.

Symmetry is a known property of probabilistic conditional independence [19] and is included as one inference axiom in the semi-graphoid axioms [61]. Thus, $I(X, Y, Z)$ holds in $P(U)$ if and only if $I(Z, Y, X)$ holds in $P(U)$.

Given $I(X, Y, Z)$, the nodes R_X reachable from X will often be different from the nodes R_Z reachable from Z, since X and Z are disjoint sets of nodes.

Example 54. *Recall testing $I(D, G, H)$ in Figure 6.1i, where the pruning optimization step yields the sub-DAG on $\{D, G, H\} \cup An(DGH)$ in Figure 6.1ii. Notice that $R_D = \{A, B, D, E\}$, while $R_H = \{F, H\}$.*

Figure 6.2: Consider v-structure node $C \in Y$ in $I(X, Y, Z)$. (6.2i) Approaching C from the top, say from A, renders B reachable. (6.2ii) Approaching C from the bottom, makes B unreachable.

We now analyze factors that affect reachability.

The set of reachable nodes can depend on the direction a node is approached from and on the kind of node. Consider a v-structure node v_i with parent set P_i and such that $v_i \in Y$ in $I(X, Y, Z)$. Then reaching any node in P_i makes all other non-evidence nodes in P_i reachable, since v_i is an open, convergent variable. On the contrary, P_i is not reachable from the children of v_i, since v_i is a closed, sequential variable.

Example 55. *Consider the v-structure C in the DAG $\mathcal{D}$ in Figure 6.2ii, where $C \in Y$ of $I(X, Y, Z)$. C does not block the path from A to B in Figure 6.2i, since C is an open, convergent variable. On the contrary, C blocks the path from D to B (and A) in Figure 6.2ii, since C is a closed, sequential variable.*

Consequently, it may be better to compute reachability in DAGs from bottom-to-top rather than top-to-bottom. Given $I(X, Y, Z)$ to be tested in a DAG, the universal approach to implementing d-separation is to test whether

$$R_X \cap Z = \emptyset. \tag{6.46}$$

However, by symmetry of independence, one may equivalently test whether

$$R_Z \cap X = \emptyset. \tag{6.47}$$

The idea is to compute the smaller of R_X and R_Z. It is conjectured that the set with fewer nodes between R_X and R_Z corresponds to the set X or Z appearing "lower" in the DAG. In other words, if X is lower than Z in the DAG, then R_X likely has fewer elements than R_Z.

Algorithm 3 Symmetric d-separation tests whether a given independence $I(X,Y,Z)$ holds in a given BN $\mathcal{D}$.

```
 1: procedure SYMMETRIC D-SEPARATION(X,Y,Z,D)
 2:     ▷ Initialization
 3:     X^up ← X ∪ An(X)
 4:     Y^up ← Y ∪ An(Y)
 5:     Z^up ← Z ∪ An(Z)
 6:     XYZ^up ← X^up ∪ Y^up ∪ Z^up                    ▷ The relevant part of D
 7:     ▷ Exploit symmetry
 8:     if |An(X)| > |An(Z)| then                          ▷ Check depth
 9:         R_X ← SYMMETRIC-REACHABLE(X, Y, Z, D, XYZ^up)
10:         if R_X ∩ Z = ∅ then
11:             return true
12:     else                                     ▷ Test I(X,Y,Z) as I(Z,Y,X)
13:         R_Z ← SYMMETRIC-REACHABLE(Z, Y, X, D, XYZ^up)
14:         if R_Z ∩ X = ∅ then
15:             return true
16:     return false
```

We now propose a measure of depth for a given set of nodes in a DAG.

Definition 6.2. *The* depth *of a set W of nodes in a DAG, denoted $depth(W)$, is defined as $|An(W)|$.*

Example 56. *In Figure 6.1ii, $depth(D) = 2$, since $An(D) = \{A, B\}$.*

Given $I(X,Y,Z)$ to be tested in a DAG, if $depth(X) > depth(Z)$, we test

$$R_X \cap Z = \emptyset;$$

otherwise, test

$$R_Z \cap X = \emptyset.$$

Our main contribution, called Symmetric d-Separation, is formally given in Algorithm 3. For computing reachable nodes in lines 9 and 13, we use Algorithm 4.

Example 57. *Let us test $I(D,G,H)$ in Figure 6.1i using Algorithm 3. In particular, line 3 computes*

$$D^{up} = \{A, B, D\},$$

while line 5 determines

$$H^{up} = \{A, B, D, E, F, G, H\}.$$

The computation of D^{up} *and* H^{up} *is used in the key pruning step in line 6. Line 8 checks depth. Since* $|An(D)| < |An(H)|$*, we exploit symmetry to instead test* $I(H, G, D)$ *in line 12. In line 13,*

$$R_H = \{F, H\}.$$

In line 14,

$$R_H \cap \{D\} = \emptyset.$$

Hence, $I(H, G, D)$ *holds.*

The key point in Example 57 is that it is beneficial to take advantage of symmetry in order to test $I(D, G, H)$ as $I(H, G, D)$. Recall how DARWICHE tests $I(D, G, H)$ in Example 53. First, DARWICHE prunes the DAG. Next, DARWICHE computes the nodes reachable from D,

$$R_D = \{A, B, D, E\}, \tag{6.48}$$

and then tests whether

$$R_D \cap \{H\} = \emptyset. \tag{6.49}$$

In contrast, consider how SYMMETRIC D-SEPARATION tested $I(D, G, H)$ in Example 57. By computing $An(D)$ and $An(H)$ during the pruning optimization step, it estimated that H is lower than D in the DAG of Figure 6.1ii. Since it may be preferable to traverse bottom-up, symmetry is exploited to equivalently test $I(D, G, H)$ as $I(H, G, D)$. Hence, the nodes reachable from H are computed,

$$R_H = \{F, H\}, \tag{6.50}$$

and then it is checked whether

$$R_H \cap \{D\} = \emptyset. \tag{6.51}$$

Thereby, DARWICHE computed $R_D = \{A, B, D, E\}$ in (6.48), whereas SYMMETRIC D-SEPARATION computed $R_H = \{F, H\}$ in (6.50).

6.4 Experimental Results

We report on an empirical comparison of DARWICHE and SYMMETRIC-REACHABLE. Both methods were implemented in Python using the *NetworkX* library (see `networkx.github.io`). The experiments were conducted on a 2.9 GHz Intel Core i7 with 8 GB RAM. The evaluation was carried out on 18 real-world or benchmark BNs listed in the first columns of Table 6.1. The second column reports the number of variables in each BN. For each BN, 1000 independencies $I(X, Y, Z)$ were randomly generated and tested by DARWICHE and SYMMETRIC-REACHABLE. Following REACHABLE in [44], in our experiments X and Z are kept to being singleton sets. The average time in seconds required by DARWICHE and SYMMETRIC-REACHABLE are reported in the fourth and sixth columns, respectively. For completeness, the percentage of time taken by DARWICHE to prune is listed in the third column, while the percentage of time taken to exploit symmetry in SYMMETRIC-REACHABLE is given in the fifth column. In the last column, we show the time savings of SYMMETRIC-REACHABLE over DARWICHE.

Experiments are broken down into separate cases based on the percentage of evidence variables. Table 6.1 reports the case where 1% of the BN variables are randomly instantiated as evidence $Y \in I(X, Y, Z)$. Similarly, Tables 6.2, 6.3, and 6.4 show cases 5%, 10%, and 25%, respectively.

Table 6.1 shows that DARWICHE is faster than SYMMETRIC D-SEPARATION by an average of 6% when 1% evidence is considered. In Table 6.2, for 5% evidence, SYMMETRIC D-SEPARATION is faster in the 10 largest BNs and slower in the 8 smallest BNs. SYMMETRIC D-SEPARATION wins in 15 out of 18 BNs when 10% evidence is considered in Table 6.3. Finally, in Table 6.4, SYMMETRIC D-SEPARATION is faster than DARWICHE in all BNs by an average of 6%, where 25% evidence is examined.

Deeper analysis of experimental results reveals two trends. First note that the time savings offered by SYMMETRIC D-SEPARATION is proportional to the percentage of observed variables. The more evidence that is observed, the greater the time savings of SYMMETRIC D-SEPARATION over DARWICHE. DARWICHE is faster than SYMMETRIC D-SEPARATION by 6% when 1% evidence is considered, and by 1% when 5% evidence is examined. In contrast, SYMMETRIC D-SEPARATION, is faster than DARWICHE by 2% when 10% evidence is considered, and by 6% when 25% evidence is tested.

Table 6.1: With 1% of evidence, comparison of DARWICHE and SYMMETRIC d-SEPARATION with 1000 randomly generated independencies in each BN.

BN	size	time pruning darwiche	time (s) darwiche	time swap	time (s) sym sep	time savings
child	20	4.77%	2.74E-02	1.21%	2.76E-02	-0.58%
insurance	27	4.77%	2.74E-02	1.15%	2.82E-02	-2.96%
water	32	5.17%	2.96E-02	1.16%	3.04E-02	-2.68%
mildew	35	5.04%	2.85E-02	1.08%	3.03E-02	-6.14%
alarm	37	4.87%	2.77E-02	1.25%	2.80E-02	-1.04%
barley	48	4.89%	3.00E-02	1.01%	3.16E-02	-5.48%
hailfinder	56	4.81%	2.91E-02	1.03%	3.05E-02	-4.92%
hepar2	70	4.64%	2.95E-02	1.22%	3.16E-02	-7.21%
win95pts	76	5.36%	3.12E-02	1.00%	3.33E-02	-6.57%
pathfinder	109	5.13%	3.90E-02	0.95%	4.21E-02	-7.91%
munin1	186	5.15%	3.38E-02	0.77%	3.95E-02	-16.70%
andes	223	3.97%	6.64E-02	0.49%	7.58E-02	-14.25%
pigs	441	5.01%	1.20E-01	0.37%	1.28E-01	-6.50%
link	724	4.26%	2.41E-01	0.20%	2.57E-01	-6.70%
munin2	1003	4.96%	4.44E-01	0.22%	4.76E-01	-7.26%
munin4	1038	4.55%	3.45E-01	0.19%	3.65E-01	-5.80%
munin	1041	3.97%	2.91E-01	0.15%	3.05E-01	-4.59%
munin3	1041	4.55%	3.76E-01	0.21%	4.10E-01	-8.96%
Average		4.77%	1.23E-01	0.76%	1.32E-01	-6.46%

The second trend to note is that SYMMETRIC d-SEPARATION is especially effective on larger BNs. Consider those 9 BNs in Table 6.1 with more than 100 variables. It can be verified that DARWICHE is faster by 9% when 1% evidence is considered. SYMMETRIC d-SEPARATION, on the other hand, is faster in 5% by 2%, in 10% by 3%, and in 25% by 9%, respectively.

The important point in this section is that the experimental results show the usefulness of SYMMETRIC d-SEPARATION when implementing d-separation.

6.5 Conclusion

Although *d-separation* [61] has been extensively studied, all previous implementations, including [17], [29], [44], [60], [74], and [46], test independence $I(X,Y,Z)$ by computing R_X, the nodes reachable from X along active paths with respect to Y,

Table 6.2: With 5% of evidence, comparison of DARWICHE and SYMMETRIC D-SEPARATION with 1000 randomly generated independencies in each BN.

BN	size	time pruning darwiche	time (s) darwiche	time swap	time (s) sym sep	time savings
child	20	5.24%	3.33E-02	1.15%	3.39E-02	-1.67%
insurance	27	5.11%	3.00E-02	1.09%	3.19E-02	-6.29%
water	32	5.12%	3.25E-02	1.19%	3.34E-02	-2.72%
mildew	35	5.04%	3.11E-02	1.03%	3.33E-02	-7.17%
alarm	37	5.32%	2.94E-02	1.18%	3.00E-02	-1.98%
barley	48	5.00%	6.45E-02	0.70%	6.83E-02	-5.97%
hailfinder	56	4.88%	5.82E-02	0.63%	5.98E-02	-2.73%
hepar2	70	4.57%	8.35E-02	0.47%	8.43E-02	-0.89%
win95pts	76	4.52%	8.31E-02	0.48%	8.30E-02	0.12%
pathfinder	109	3.90%	1.27E-01	0.32%	1.26E-01	0.87%
munin1	186	3.59%	2.26E-01	0.18%	2.25E-01	0.37%
andes	223	4.38%	3.62E-01	0.21%	3.59E-01	0.87%
pigs	441	3.56%	5.90E-01	0.09%	5.86E-01	0.56%
link	724	3.93%	1.30E+00	0.10%	1.29E+00	0.76%
munin2	1003	3.44%	1.52E+00	0.05%	1.48E+00	2.58%
munin4	1038	3.43%	1.55E+00	0.05%	1.50E+00	2.94%
munin	1041	3.41%	1.57E+00	0.05%	1.53E+00	2.91%
munin3	1041	3.34%	1.53E+00	0.05%	1.49E+00	2.31%
Average		4.32%	5.12E-01	0.50%	5.03E-01	-0.84%

and then test whether $R_X \cap Z = \emptyset$. Thereby, no previous study has examined exploiting the *symmetry* [19] of probabilistic conditional independence when implementing d-separation.

In this chapter, we consider exploiting symmetry to answer $I(X, Y, Z)$ by testing $I(Z, X, Y)$. This involves computing R_Z, the nodes reachable from Z along active paths with respect to Y, and then checking whether $R_Z \cap X = \emptyset$. It is important to realize that R_X and R_Z can contain different variables and have distinct cardinalities. Our analysis in Section 3 reveals that v-structure nodes play a critical role in this respect. As shown in Example 55, it is better to approach an observed v-structure node from the bottom (from a child) rather than from the top (from a parent). When approached from the bottom, the parents are not reachable, since the v-structure node is a closed, sequential valve. In contrast, when approached from the top, all

Table 6.3: With 10% of evidence, comparison of Darwiche and Symmetric d-Separation with 1000 randomly generated independencies in each BN.

BN	size	time pruning darwiche	time(s) darwiche	time swap	time(s) sym sep	time savings
child	20	4.33%	5.20E-02	0.64%	5.27E-02	-1.52%
insurance	27	4.21%	5.21E-02	0.62%	5.21E-02	0.10%
water	32	4.45%	9.13E-02	0.50%	9.18E-02	-0.45%
mildew	35	4.10%	7.52E-02	0.45%	7.48E-02	0.60%
alarm	37	4.11%	7.45E-02	0.44%	7.35E-02	1.36%
barley	48	4.11%	1.06E-01	0.37%	1.05E-01	1.15%
hailfinder	56	3.83%	1.29E-01	0.34%	1.27E-01	2.27%
hepar2	70	3.66%	1.81E-01	0.24%	1.76E-01	2.93%
win95pts	76	4.30%	2.24E-01	0.28%	2.18E-01	2.66%
pathfinder	109	4.36%	3.00E-01	0.23%	2.95E-01	1.69%
munin1	186	3.98%	5.38E-01	0.14%	5.27E-01	2.14%
andes	223	3.41%	5.91E-01	0.09%	5.86E-01	0.90%
pigs	441	3.82%	1.46E+00	0.09%	1.48E+00	-1.88%
link	724	3.14%	2.22E+00	0.03%	2.21E+00	0.51%
munin2	1003	3.28%	3.94E+00	0.05%	3.73E+00	5.29%
munin4	1038	3.29%	3.90E+00	0.04%	3.65E+00	6.28%
munin	1041	3.08%	3.91E+00	0.03%	3.59E+00	8.11%
munin3	1041	3.20%	4.13E+00	0.04%	3.84E+00	6.87%
Average		3.81%	1.22E+00	0.26%	1.16E+00	2.17%

other non-evidence parents are reachable, since the v-structure node is an open, convergent valve. Then, more generally, it may be better to find reachable nodes from whichever of X and Z is "lower" in the DAG. We suggest a measure of depth in Definition 6.2. We incorporate symmetry into our implementation of d-separation, given in Algorithm 4, and called Symmetric d-Separation. We empirically compare our approach with the implementation of d-separation suggested by [17] with favourable results. The experimental results are promising in two aspects. The effectiveness increases with network size, as well as with the amount of observed evidence, culminating with an average time savings of 9% in the 9 largest networks.

This work was motivated in part by recent advances in graph search [38]. Classical search algorithms such as A^* are unidirectional and have been extended to search from goal to start and to search from the start and goal simultaneously, meeting somewhere in the middle. Future work can investigate a meet in the middle variation

of d-separation. Other applications of our work may include *influence diagram* [44] and inference algorithms that test independencies, such as *Lazy Propagation* [50].

Algorithm 4 Given an independence $I(X, Y, Z)$, SYMMETRIC-REACHABLE traverses all active paths from X within the relevant part of a BN $\mathcal{D}$.

```
1: procedure SYMMETRIC-REACHABLE(X,Y,Z,D, XYZ^up)
2:     V ← ∅                                   ▷ (Variable,direction) marked as visited
3:     R ← ∅                                   ▷ Variables reachable via an active path
4:     L ← ∅                                   ▷ (Variable,direction) to be visited
5:     for v ∈ X do
6:         L ← L ∪ {(↑, v)}
7:     ▷ Starting from X traverse relevant paths that are active
8:     while L ≠ ∅ do                          ▷ While variables to be checked
9:         Select (d, v) in L
10:        L ← L − {(d, v)}
11:        if (d, v) ∉ V then                  ▷ If v has not been visited from direction d
12:            if v ∉ Y then
13:                R ← R ∪ {v}                 ▷ v is reachable
14:            V ← V ∪ {(d, v)}                ▷ Mark v as visited from direction d
15:            if d =↑ and v ∉ Y then
16:                for v_i ∈ Pa(v) do          ▷ v is open sequential
17:                    if v_i ∈ XYZ^up then    ▷ Only explore relevant paths
18:                        L ← L ∪ {(↑, v_i)}
19:                for v_i ∈ Ch(v) do          ▷ v is open divergent
20:                    if v_i ∈ XYZ^up then    ▷ Only explore relevant paths
21:                        L ← L ∪ {(↓, v_i)}
22:            else if d =↓ then
23:                if v ∉ Y then
24:                    for v_i ∈ Ch(v) do      ▷ v is open sequential
25:                        if v_i ∈ XYZ^up then    ▷ Only explore relevant paths
26:                            L ← L ∪ {(↓, v_i)}
27:                if v ∈ A then
28:                    for v_i ∈ Pa(v) do      ▷ v is open convergent
29:                        if v_i ∈ XYZ^up then    ▷ Only explore relevant paths
30:                            L ← L ∪ {(↑, v_i)}
31:    return R            ▷ All variables reachable from X via active paths within the
   relevant part of D
```

Table 6.4: With 25% of evidence, comparison of Darwiche and Symmetric d-Separation with 1000 randomly generated independencies in each BN.

BN	size	time pruning darwiche	time(s) darwiche	time swap	time(s) rp_sep	time savings
child	20	4.09%	1.65E-01	0.35%	1.62E-01	1.74%
insurance	27	4.02%	1.69E-01	0.31%	1.62E-01	3.98%
water	32	4.14%	2.57E-01	0.29%	2.52E-01	1.85%
mildew	35	4.27%	2.41E-01	0.27%	2.37E-01	1.75%
alarm	37	4.01%	2.54E-01	0.24%	2.45E-01	3.62%
barley	48	3.85%	3.47E-01	0.20%	3.32E-01	4.15%
hailfinder	56	3.91%	4.19E-01	0.18%	4.05E-01	3.34%
hepar2	70	3.90%	4.97E-01	0.16%	4.82E-01	3.04%
win95pts	76	3.90%	5.66E-01	0.15%	5.62E-01	0.84%
pathfinder	109	3.78%	8.20E-01	0.12%	7.99E-01	2.57%
munin1	186	3.64%	1.49E+00	0.08%	1.44E+00	3.21%
andes	223	3.59%	1.80E+00	0.07%	1.74E+00	3.61%
pigs	441	3.09%	4.33E+00	0.05%	4.22E+00	2.53%
link	724	2.64%	8.76E+00	0.02%	8.38E+00	4.32%
munin2	1003	2.24%	1.36E+01	0.02%	1.12E+01	18.08%
munin4	1038	2.25%	1.41E+01	0.03%	1.20E+01	14.88%
munin	1041	2.26%	1.41E+01	0.02%	1.20E+01	15.04%
munin3	1041	2.26%	1.45E+01	0.02%	1.23E+01	15.02%
Average		3.44%	4.25E+00	0.14%	3.71E+00	5.76%

Chapter 7

Sum-Product Networks

Here we give pertinent definitions and review two deep learning networks: arithmetic circuits and sum-product networks.

We use uppercase letters X to denote variables and lowercase letters x to denote their values. We denote sets of variables with boldface uppercase letters $\mathbf{X}$ and their instantiations with boldface lowercase letters $\mathbf{x}$. For a boolean variable X, x denotes when $X = true$ and $\bar{x}$ for $X = false$. Also, we use $P(x)$ instead of $P(X = x)$ and $P(\mathbf{x})$ to mean $P(\mathbf{X} = \mathbf{x})$. Lastly, we denote graphs with calligraphic letters.

7.0.1 Arithmetic Circuits

Bayesian networks [61] are probabilistic graphical models composed of a *directed acyclic graph* (DAG) and a set of *conditional probability tables* (CPTs) corresponding to the structure of the DAG. For example, Figure 7.1 shows the DAG of a Bayesian network $\mathcal{B}$ on binary variables A, B, and C, where CPTs $P(A)$, $P(B|A)$, and $P(C|A)$ are not illustrated.

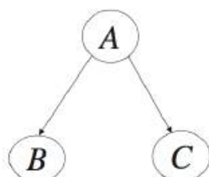


Figure 7.1: A Bayesian network.

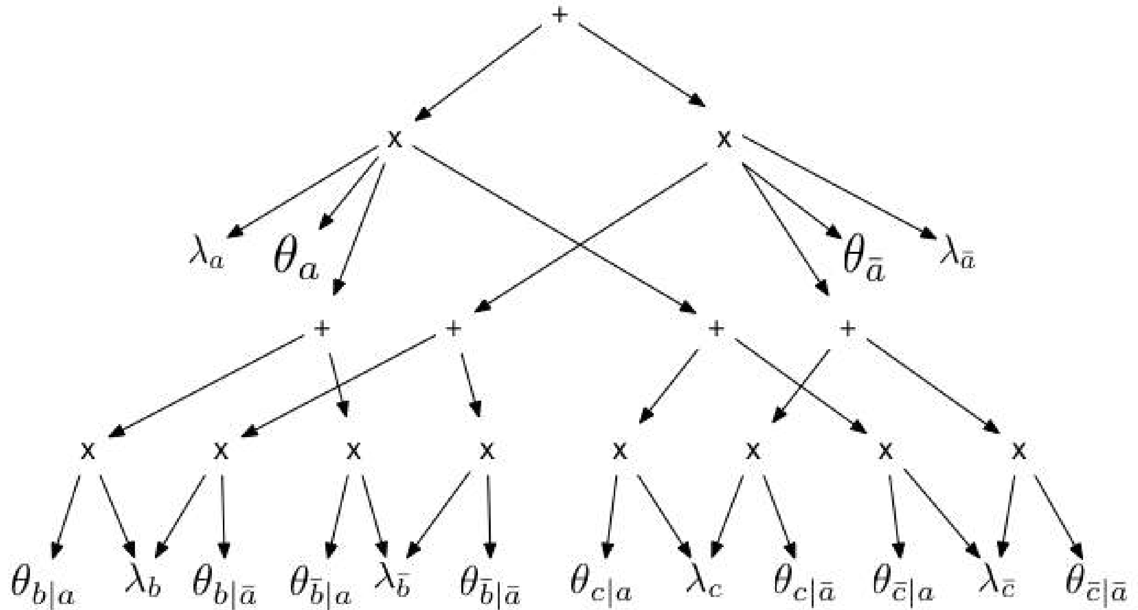


Figure 7.2: An AC for the Bayesian network in Figure 7.1 following (58).

Bayesian networks are compiled into *Arithmetic Circuits* (ACs) [16] by mapping the operations performed when marginalizing all variables from the BN into a graph.

Example 58. *Consider marginalizing variables A, B, and C from the Bayesian network in Figure 7.1:*

$$\sum_{A} P(A) \cdot \sum_{B} P(B|A) \cdot \sum_{C} P(C|A). \tag{7.52}$$

The computational process in Example 58 can be represented as the AC in Figure 7.2 and formalized next.

Definition 7.1. *[16] An arithmetic circuit (AC) over variables* $\mathbf{U}$ *is a rooted, DAG whose leaf nodes are labeled with numeric constants, called parameters, or λ variables, called indicators, and whose other nodes are labeled with multiplication and addition operations.*

Notice that parameter variables are set according to the Bayesian network CPTs, while indicator variables are set according to any observed evidence.

Example 59. *Figure 7.2 shows an AC. Here,* $P(a) = \theta_a$ *and* $P(b|a) = \theta_{b|a}$*, the parameters are the numeric constant variables in the leaf nodes. The indicators are the* λ *variables in the leaf nodes, including* λ_a*, the indicator for* a*, and* $\lambda_{\bar{a}}$ *the indicator for* $\bar{a}$*.*

In practical terms, an AC graphically represents a *network polynomial* over variables $\mathbf{U}$, where for each $X \in \mathbf{U}$, we have a set of indicator variable λ_x and for each network parameter for CPT $P(x|\mathbf{u})$, we have a set of parameters $\theta_{x|\mathbf{u}}$.

Example 60. *The AC in Figure 7.2 represents the following network polynomial* f*:*

$$f = (\lambda_a \cdot \theta_a) \cdot (\lambda_b \cdot \theta_{b|a} + \lambda_{\bar{b}} \cdot \theta_{\bar{b}|a}) \cdot (\lambda_c \cdot \theta_{c|a} + \lambda_{\bar{c}} \cdot \theta_{\bar{c}|a}) \tag{7.53}$$

$$+ (\lambda_{\bar{a}} \cdot \theta_{\bar{a}}) \cdot (\lambda_b \cdot \theta_{b|\bar{a}} + \lambda_{\bar{b}} \cdot \theta_{\bar{b}|\bar{a}}) \cdot (\lambda_c \cdot \theta_{c|\bar{a}} + \lambda_{\bar{c}} \cdot \theta_{\bar{c}|\bar{a}}) \tag{7.54}$$

A query can be answered in an AC in time and space linear in the circuit size [16], where *size* means the number of edges. In order to evaluate an AC, we first set the indicator variables to 0 or 1 accordingly to any observed evidence. For example, when computing $P(a, \bar{c})$, indicators λ_a, λ_b, $\lambda_{\bar{b}}$, and $\lambda_{\bar{c}}$ are set to 1, while indicators $\lambda_{\bar{a}}$ and λ_c are set to 0. Next, we traverse the AC upward computing the value of a node after having computed the values of its children. A downward pass can be performed to compute the posterior of all variables given evidence [16].

7.0.2 Sum-Product Networks

Sum-Product Networks (SPNs) [70] are a deep probabilistic model. One striking feature of SPNs is that they can efficiently represent tractable probability distributions [71].

Definition 7.2. *[70] A* Sum-product network *(SPN) over Boolean variables* $\mathbf{X}$ *is a rooted DAG whose leaves are indicators* $\lambda_{x_1}, \ldots, \lambda_{x_N}$ *and* $\lambda_{\bar{x}_1}, \ldots, \lambda_{\bar{x}_N}$ *and whose internal nodes are sums and products. Each edge* (v_i, v_j) *emanating from a sum node* v_i *has a non-negative weight* w_{ij}*. The value of a product node is the product of the values of its children. The value of a sum node is* $\sum_{v_j \in Ch(v_i)} w_{ij} val(v_j)$*, where* $Ch(v_i)$ *are the children of* v_i *and* $val(v_j)$ *is the value of node* v_j*. The value of an SPN* $\mathcal{S}$ *is the value of its root.*

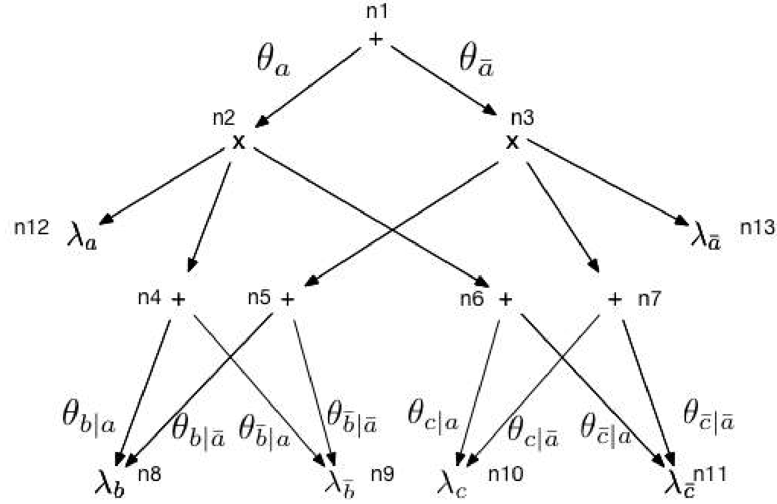


Figure 7.3: A Sum-product network.

Example 61. *An SPN is shown in Figure 11.1. Here, for instance, λ_a is an indicator, $n2$ is a product node, and $n4$ is a sum node.*

An SPN graph compactly represents a polynomial over variables U.

Example 62. *The SPN in Figure 11.1 represents the polynomial f:*

$$f = (\lambda_a \cdot \theta_a) \cdot (\lambda_b \cdot \theta_{b|a} + \lambda_{\bar{b}} \cdot \theta_{\bar{b}|a}) \cdot (\lambda_c \cdot \theta_{c|a} + \lambda_{\bar{c}} \cdot \theta_{\bar{c}|a}) \tag{7.55}$$

$$+ (\lambda_{\bar{a}} \cdot \theta_{\bar{a}}) \cdot (\lambda_b \cdot \theta_{b|\bar{a}} + \lambda_{\bar{b}} \cdot \theta_{\bar{b}|\bar{a}}) \cdot (\lambda_c \cdot \theta_{c|\bar{a}} + \lambda_{\bar{c}} \cdot \theta_{\bar{c}|\bar{a}}) \tag{7.56}$$

The important notion of scope is now reviewed.

Definition 7.3. *[85] For any node n in an SPN, if n is a leaf node then $scope(n) = \{X\}$, else $scope(n) = \bigcup_{c \in Ch(n)} scope(c)$.*

Example 63. *Recall Figure 11.1. The scope of leaf node $n10$ is $scope(n10) = \{C\}$, since $n10$ is an indicator. Sum node $n6$ has two children, nodes $n10$ and $n11$. By definition, the scope of $n6$ is $scope(n6) = \{C\}$. Product node $n3$ has $scope(n3) =$*

$\{A, B, C\}$, *which is the union of* $n3$ *children's scope. Similarly, for instance,*

$$scope(n8) = \{B\}, \tag{7.57}$$

$$scope(n4) = \{B\}, \tag{7.58}$$

$$scope(n2) = \{A, B, C\}, \text{ and} \tag{7.59}$$

$$scope(n1) = \{A, B, C\}. \tag{7.60}$$

The follow three properties have practical usefulness in SPNs.

Definition 7.4. *[70] An SPN is* complete *iff each sum node has children with the same scope.*

Definition 7.5. *[70] An SPN is* consistent *iff no variable appears negated in one child of a product node and non-negated in another.*

Definition 7.6. *[70] An SPN is* decomposable *iff for every product node* n, $scope(n_i) \cap scope(n_j) = \emptyset$, *where* $n_i, n_j \in Ch(n), i \neq j$.

In SPNs, decomposability implies consistency [70]. A complete and consistent SPN computes a query in time linear in its size (number of edges) [70].

Example 64. *The SPN in Figure 11.1 is complete, since all sum nodes have children with the same scope. For instance, the children of sum node* $n4$*, namely leaf nodes* $n8$ *and* $n9$*, have the same scope, that is,* $scope(n8) = scope(n9) = \{b\}$. *The same can be verified for sum nodes* $n1$, $n2$, $n3$, $n5$, $n6$, *and* $n7$. *Also, the SPN in Figure 11.1 is decomposable, since all children have disjoint scopes for every product node. For example, the children* $n4$ *and* $n6$ *of product node* $n2$ *have disjoint scopes. Since decomposability implies consistency, the SPN is also consistent.*

In our paper, we relate SPNs with *convolutional neural networks* (CNNs) [34]. In general, CNNs are formed by convolutional and pooling layers. A *convolutional layer* can be constructed by applying a convolutional operation with a *filter* on a previous layer output. A *pooling layer* can be built by applying a max or average operation over some elements of the previous layer with respect to a sliding *window*. A sum-pooling layer can be easily obtained from an average-pooling layer.

7.0.3 Mixture Model Interpretation

SPNs can be viewed as generalized DAGs of *mixture models*, where sum nodes correspond to mixtures over subsets of variables and product nodes correspond to features or mixture components [70]. In general, mixture models can be interpreted as a distribution over random variables described by the marginalization of *latent variables* (LVs). Under this interpretation, each sum node from an SPN has an associated LV [66].

Definition 7.3 agrees with the mixture model interpretation, since it takes into consideration the scope of nodes involved in previous computation. That is, a hierarchical mixture model is considered as a hierarchical structured LV model. Henceforth, by $scope_{mm}$, we mean scope as given in Definition 7.3.

Chapter 8

Resolving Inconsistencies of Scope Interpretations in Sum-Product Networks

8.1 Introduction

Sum-product networks (SPNs) [70] are a class of tractable deep learning models for probabilistic inference. This is an attractive feature when compared to probabilistic graphical models in general, including *Bayesian networks* [61], where inference is NP-hard [13]. The tractability of SPN can be seen as follows [70]: the partition function of a probabilistic graphical model is intractable because it is the sum of an exponential number of terms. All marginals are sums of subsets of these terms, meaning that if the partition function can be computed efficiently, so can they. SPNs have been successfully applied in a variety of practical applications, including image completion [25, 65, 70], computer vision [1], classification [30], and speech and language modeling [12, 67]. An SPN is a rooted, directed acyclic graph in which leaf nodes are indicator variables and internal nodes are either sum or multiplication operations. The practical usefulness of SPNs relies on three properties: *completeness*, *consistency*, and *decomposability* [70]. The satisfaction of these properties can be verified using the scope definition. The *scope* [70] of a node are the random variables appearing as descendants of that node.

There are probabilistic and non-probabilistic interpretations of SPNs. From a non-probabilistic perspective, SPNs can be interpreted as deep neural networks with the following three special properties [79]. First, SPNs are labeled, meaning that all nodes in the network are associated to random variables by the means of a scope definition.

Next, they are constrained, that is, subjected to the completeness, consistency, and decomposability properties. Lastly, they are fully probabilistic neural networks, where each node (or neuron) is a valid SPN and still models a correct distribution over its scope.

Our discussion focuses on the probabilistic perspective, where SPNs have the capacity to generalize a variety of approaches to representing and learning tractable models, including *mixture models* [66] and *arithmetic circuits* (ACs) [16]. We compare these two interpretations by means of the scope definition. The scope definition is ideally suited for mixture model interpretation of an SPN, since the scope of a node grows larger the closer the node is to the root. This reflects the fact that higher nodes are mixtures of more random variables. On the other hand, this interpretation of scope is direct opposition with ACs, where sum nodes are marginalization operations that *reduce* scope.

In this chapter, our first contribution is a new scope definition that suits the AC interpretation of an SPN. This definition considers the fact that ACs are compiled from Bayesian networks. Thus, the random variables at the leaf nodes come from probabilities in Bayesian network conditional probability tables. The scope of the leaf nodes then are labeled according to the variables in the conditional probability tables. The scope of a sum node removes one random variable from the union of the scope of its children. This reflects the interpretation of sum nodes as being marginalization. The scope of a multiplication node is the union of its children's scope. We establish that the completeness property of SPN can be defined under the AC interpretation of scope. Unfortunately, consistency and decomposability can not be applied using this scope definition, meaning that the definition can not be used for verifying tractability of an SPN. Moreover, the proposed scope definition for ACs can be non-intuitive for leaf nodes under a mixture model interpretation. For example, the scope of a leaf node can consider variables that are not random variables for an SPN leaf node, yet appear in a conditional probability table from the Bayesian network.

The second contribution of this chapter is a novel scope definition with two advantages. First, it can be used when defining all three SPN properties. Second, it addresses the counter-intuitive points in both the mixture model and AC interpretations. Here, we define the scope of sum nodes more intuitively from an AC point of view by treating them as marginalization. Hence, the scope of sum nodes removes one

random variable from the union of the scope of its children. Moreover, we make the scope of leaf nodes more intuitive from a mixture model point of view by only considering variables that are random variables for the SPN leaf node. Since both mixture models and ACs can be compiled into SPNs, the new scope definition improves the understanding of SPNs for different research communities, while still guaranteeing tractable inference during learning.

8.2 Scope Under an AC Interpretation

In order to propose a scope definition under an AC interpretation, we first need to formalize the equivalence between ACs and SPNs. ACs are equivalent to SPNs for discrete domains [71]. The representational differences are that ACs use indicator nodes and parameter nodes as leaves, while SPNs use univariate distributions as leaves and attach all parameters to the outgoing edges of sum nodes [71]. For example, the AC in Figure 7.2 is equivalent to the SPN in Figure 11.1. Indeed, the polynomial represented by the AC in (7.54) for Example 60 is the same as the polynomial represented by the SPN in (7.56) for Example 62.

We now propose a novel scope definition taking into consideration the AC interpretation of an SPN. First, notice that all the weights in an SPN built from an AC correspond to probabilities in the given Bayesian network. Thus, we can associate labels from the CPTs to each edge weight. Outgoing edges from a product node have the same labels as its incoming edges.

Example 65. *In the SPN of Figure 11.1, sum node $n4$ has weights $\theta_{b|a}$ and $\theta_{\bar{b}|a}$. Both weights are probabilities $P(b|a)$ and $P(\bar{b}|a)$ from the CPTs in the Bayesian network in Figure 7.1. Thus, the set $\{A, B\}$ of variables are associated labels of both edges $(n4, n8)$ and $(n4, n9)$. Moreover, outgoing edges $(n2, n12)$, $(n2, n4)$, and $(n2, n6)$ from product node $n2$ have the set $\{A\}$ as the associated label, since θ_a is the weight attached to $n2$'s incoming edge.*

Next, let us consider the layers of an SPN, where we will consider the bottom layer being the leaf nodes and the top layer being the root. Here, sum or product nodes are considered to be in the same *layer* if they have the same length (number of nodes) in the longest path from the root.

Example 66. *The SPN in Figure 11.1 contains 4 layers. The bottom layer is formed by leaf nodes* $n8$, $n9$, $n10$, $n11$, $n12$, *and* $n13$. *The next layer consisting of sum nodes* $n4$, $n5$, $n6$, *and* $n7$ *followed by a layer formed by multiplication nodes* $n2$ *and* $n3$. *Lastly, the top layer is sum node* $n1$.

We now give the first contribution of this chapter.

Definition 8.1. *The scope of a leaf node is the set of all variables involved in the labeling of its incoming edges. The scope of a product node is the union of its children's scope. The scope of a sum node is the intersection between the union of its children's scope in layer* k *and the scope of all other nodes in layer* k.

The scope of a node under the AC interpretation will be denoted $scope_{ac}$.

Example 67. *Recall the SPN in Figure 11.1. The scope of leaf node* $n10$ *is* $scope_{ac}(n10) = \{A, C\}$, *since* $n10$ *is a leaf node with labeled incoming edges* $(n6, n10)$ *and* $(n7, n10)$ *both over variables* $\{A, C\}$. *Product node* $n3$ *has* $scope(n3) = \{A\}$, *which is the union of* $n3$ *children's scope. Now, consider sum node* $n6$. *It has two children,* $n10$ *and* $n11$. *The other nodes on the same layer as* $n10$ *and* $n11$ *are* $n8$ *and* $n9$. *Then, by definition, the scope of* $n6$ *is:*

$$\begin{aligned} scope_{ac}(n6) &= (scope_{ac}(n10) \cup scope_{ac}(n11)) \cap (scope_{ac}(n8) \cup scope_{ac}(n9)) \\ &= (\{A, C\} \cup \{A, C\}) \cap (\{A, B\} \cup \{A, B\}) \\ &= \{A, C\} \cap \{A, B\} \\ &= \{A\}. \end{aligned}$$

Similarly, it can be verified that:

$$scope_{ac}(n8) = \{A, B\}, \tag{8.61}$$

$$scope_{ac}(n4) = \{A\}, \tag{8.62}$$

$$scope_{ac}(n2) = \{A\}, \text{ and} \tag{8.63}$$

$$scope_{ac}(n1) = \{\}. \tag{8.64}$$

This scope definition agrees with the AC interpretation for two reasons. First, leaf nodes consider all variables involved in the Bayesian network CPTs. For example, in

Example 67, $scope_{ac}(n8) = \{A, B\}$, since the CPT $P(b|a)$ from where $\theta_{b|a}$ comes from, involves variables A and B. Second, sum nodes reduce the scope of previous layers. This is consistent with marginalization. In Example 67, for instance, $scope_{ac}(n4) = \{A\}$ reduced the scope of $n8$ by removing A from $\{A, B\}$.

8.3 A Scope Definition for Both SPN Interpretations

We first motivate the introduction of an unified interpretation of scope that works for both mixture model and AC interpretations of an SPN.

Notice that the scope definition under the AC interpretation in Definition 8.1 can consider variables that are not in the domain of indicator variables.

Example 68. *In Figure 11.1, node* $n8$ *has indicator* λ_b*, that is, an indicator for variable* b*. Yet,* $scope_{ac}(n8) = \{A, B\}$ *under the AC interpretation. Notice that variable* A *is considered part of the scope even though it is not in the domain of* λ_b*.*

On the other hand, the scope definition under the mixture model interpretation does not consider a sum node as marginalization, namely, a domain reduction operation.

Example 69. *Consider* $n4$ *and* $n8$ *in Figure 11.1. For instance,* $scope_{mm}(n8) = \{B\}$ *under the mixture model interpretation. Yet* $scope_{mm}(n4) = \{B\}$ *even though* $n4$ *is a sum node. That is, the scope was not reduced after a sum node as would be expected by marginalization.*

The scope definition under the AC interpretation can be counterintuitive in the leaf nodes of a mixture model point of view. On the other hand, the scope definition under a mixture model interpretation can be counterintuitive in the sum nodes from an AC point of view. In general, one interpretation does not cover the other one with their respective scope definition.

Our second main contribution is the proposal of a scope definition that agrees with both the mixture model and AC interpretations of SPN.

Definition 8.2. *The scope of a leaf node* n *with indicator* λ *over* X *is* $scope(v) = \{X\}$*. The scope of a sum node is the union of its children's scope, except those variables not appearing in the scope of any non-descendant nodes in the child layer. The scope of a product node is the union of its children's scope.*

Henceforth, we will denote the scope of a node under both the AC and mixture model interpretations as $scope_{am}$.

Example 70. *Recall Figure 11.1. The scope of leaf node* $n10$ *with indicator* λ_c *over* C *is* $scope_{am}(n10) = \{C\}$. *Sum node* $n6$ *has scope being the union of its children's scope, that is,* $\{C\}$, *except to those variables not appearing in the scope of any non-descendant nodes in the layer below, namely, variable* C. *Thus,* $scope_{ac}(n6) = \{\}$. *Product node* $n3$ *has* $scope(n3) = \{A\}$, *which is the union of* $n3$ *children's scope. Similarly, we can obtain:*

$$scope_{am}(n8) = \{B\}, \tag{8.65}$$

$$scope_{am}(n4) = \{\}, \tag{8.66}$$

$$scope_{am}(n2) = \{A\}, \text{ and} \tag{8.67}$$

$$scope_{am}(n1) = \{\}, \tag{8.68}$$

As shown in Example 70, Definition 8.2 addresses both counterintuitive issues by only considering variables that are in the domain of the indicator variables and by reducing the scope after a sum node.

8.4 Analysis

Here, we show that $scope_{ac}$, the scope definition under the AC interpretation, can be used for defining correctness of an SPN, but not for defining consistency and decomposability. On the other hand, we show that $scope_{am}$, the scope definition under both interpretations, can be used for defining all three properties.

First, let us consider the scope definition under the AC interpretation.

Lemma 10. *If an SPN is complete by using the* $scope_{mm}$ *definition, then it is complete by using the* $scope_{ac}$ *definition.*

Proof. Consider a complete SPN by using the $scope_{mm}$ definition. A sum node n can have either product or leaf children.

We first show that leaf children of n also have the same scope as when using $scope_{ac}$. Consider a leaf child c of n. Suppose $scope(c)_{mm} = \{X\}$. By definition,

this means that node c has an indicator λ over X. Under an AC interpretation, the λ over X exist only because it came from a Bayesian network CPT $P(X|Y)$. Hence, incoming edges for c have labels containing variables in X and Y. Thus, by the $scope(c)_{ac}$ definition, $scope(c)_{ac} = \{X, Y\}$. The same argument can be made for the other leaf children of n. Therefore, leaf children of n have the same scope using $scope_{ac}$ and $scope_{mm}$.

We next establish that a product child of n also has the same scope when using $scope_{ac}$. The scope definition for product nodes is the same for $scope_{mm}$ or $scope_{ac}$.

□ □

Example 71. *The SPN in Figure 11.1 is complete, as shown in Example 64, by using* $scope_{mm}$*, since all children of each sum node have the same scope. Consider, for instance, sum node* $n4$ *with leaf children. Both leaf children have the same scope, namely,* $scope_{mm}(n8) = scope_{mm}(n9) = \{B\}$*. Similarly, under the AC interpretation, the scope of* $n4$ *leaf children are the same, that is,* $scope_{ac}(n8) = scope_{ac}(n9) = \{A, B\}$*. Now, consider sum node* $n1$ *with product children. Both product children have the same scope, namely,* $scope_{mm}(n2) = scope_{mm}(n9) = \{A, B, C\}$*. In the same way, under the AC interpretation, the scope of* $n1$*'s product children are the same. That is,* $scope_{ac}(n2) = scope_{ac}(n9) = \{A\}$*.*

Unfortunately, $scope_{ac}$ can not be used for defining consistency or decomposability.

Example 72. *Example 64 shows that the SPN in Figure 11.1 is decomposable by using* $scope_{mm}$*. In contrast, under the AC interpretation,* $scope_{ac}(n12) = \{A\}$*,* $scope_{ac}(n4) = \{A\}$*, and* $scope_{ac}(n6) = \{A\}$*. Therefore, the SPN is not decomposable by using* $scope_{ac}$*.*

Now, let us consider $scope_{mm}$, the scope definition under both mixture models and ACs interpretation. Here, if completeness, consistency, and decomposability hold in an SPN using $scope_{mm}$, then they hold using $scope_{am}$.

Lemma 11. *If an SPN is complete by using* $scope_{mm}$ *definition, then it is also complete when using* $scope_{am}$*.*

Lemma 12. *If an SPN is consistent by using* $scope_{mm}$ *definition, then it is also complete when using* $scope_{am}$*.*

Lemma 13. *If an SPN is decomposable by using* $scope_{mm}$ *definition, then it is also complete when using* $scope_{am}$.

Example 73. *The SPN in Figure 11.1 is complete and decomposable by using* $scope_{mm}$ *as shown in Example 64. Similarly, using* $scope_{am}$, *the SPN in Figure 11.1 is complete. Consider, for example, sum node* $n4$, *which has both children with the same scope, namely,* $scope_{am}(n8) = \{B\}$ *and* $scope_{am}(n9) = \{B\}$. *Moreover, the SPN in Figure 11.1 is complete using* $scope_{am}$. *Consider, for instance, product node* $n2$, *which has different scopes for all of its three children, namely,* $scope_{mm}(n12) = \{A\}$, $scope_{mm}\ (n4) = \{B\}$, *and* $scope_{mm}(n6) = \{C\}$.

Since mixture models and ACs can be compiled to SPNs, one advantage of using the scope definition under both interpretations is that it can enhance the SPN understanding from both point of views, while still being useful for testing tractability of the network.

8.5 Conclusions

In this chapter, we identify and resolve inconsistencies regarding scope in the SPN literature. We considered two probabilistic interpretations of SPNs. The scope definition given in [70] fits nicely in the mixture models interpretation. On the other hand, that scope definition for sum nodes can be non-intuitive when under the AC interpretation of an SPN. We proposed a scope definition that suits the ACs interpretation of an SPN. Unfortunately, the scope definition of leaf nodes can be non-intuitive when under the mixture models interpretation. Thus, we proposed yet another scope definition that addresses the non-intuitive points for both mixture model and AC interpretations.

The latter scope definition can be used for checking the completeness, consistency, and decomposability properties of an SPN. Therefore, it is helpful for understanding SPNs originating from different research communities, while guaranteeing tractability when learning or compiling SPNs. Future work includes analyzing this scope definition when considering independence among LVs [66].

Chapter 9

On Learning the Structure of Sum-Product Networks

9.1 Introduction

Goodfellow, Bengio and Courville [34] contend that machine learning is the only viable approach to building artificial intelligence systems that can operate in complicated real-world environments. In particular, they show that *deep learning* is a powerful and robust framework which represents the real-world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. While it has been conjectured that deeper models are more expressive than shallow ones, *sum-product networks* (SPNs) [70] are the only deep learning model where this is provably the case [24, 53]. Delalleau and Bengio [24] explicitly write that these results contribute to the motivation of learning deep SPNs.

Gens and Domingos [31] introduced LearnSPN, which has become the standard unsupervised learning algorithm for SPNs. LearnSPN applies two general steps, namely, a "chop" operation for splitting features (columns) in a dataset and a "slice" operation for clustering instances (rows) in a dataset. *G-test* [82] and *mutual information* [14] are two methods commonly used for the chop operation, while *k-means* [54] and expectation–maximization for *Gaussian mixture models* (GMM) [26] are frequently used methods for the slice operation. To the best of our knowledge, no study has investigated which pairs of methods yield deeper SPNs.

In this chapter, we perform an empirical study of LearnSPN. We consider g-test and mutual information for chopping and k-means and GMMs for slicing. Our experiments, conducted on 20 real-world datasets, suggest that the deepest SPNs tend to be learned when using mutual information for chopping and k-means for slicing. Second, our results show that the pair of g-test and GMM tend to yield the most accurate SPNs, especially on larger datasets. These results suggest that the particular combination of mutual information and k-means may suffer especially from *overfitting* [80]. Lastly, we examine the sparseness of the learned SPN. Our experiments show that the pair of g-test and GMM often yield SPNs with fewer edges. This knowledge is beneficial to SPN learning algorithms that penalize networks with more edges [48]. Our findings can have far reaching influence, since SPNs have been applied in image completion [25, 65, 70], computer vision [1], classification [30], and speech and language modeling [12, 67]. Thereby, this study extends the deep learning literature in both practical and theoretical directions.

9.2 Background Knowledge

As our focus is on structural learning of SPNs, SPNs will be depicted with univariate distributions and sum node edge weights understood.

Example 74. *An SPN over four features X_0, X_1, X_2, and X_3 is illustrated in Figure 11.1.*

Gens and Domingos [31] introduced LearnSPN, given as Algorithm 5. LearnSPN has become the standard unsupervised learning algorithm for SPNs. LearnSPN applies two general steps, namely, a “chop” operation in line 4 for splitting features (columns) in the dataset and a “slice” operation in line 8 for clustering instances (rows) in the dataset.

Our focus in LearnSPN is on the chop operation examined in Section 9.2.1 and on the slice operation discussed in Section 9.2.2.

9.2.1 Chopping Methods

The chopping method of LearnSPN splits a dataset vertically. Here we consider two chopping methods, namely, *g-test* [82] and *mutual information* [14]. Although the

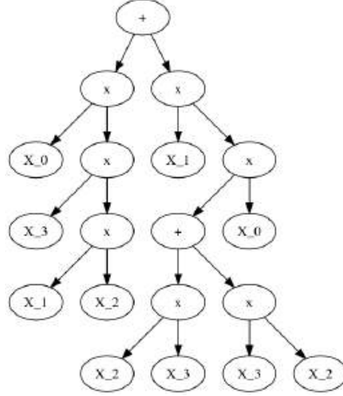


Figure 9.1: An SPN over four features X_0, X_1, X_2, and X_3.

Algorithm 5 LearnSPN(T, V)

Input: a set of instances T over features V
Output: an SPN S
Main:

1: **if** $|V| == 1$ **then**
2: $\quad S \leftarrow \textsc{leaf}(T, V)$ ▷ Univariate distribution
3: **else**
4: $\quad V_1, \ldots, V_k \leftarrow chop_m(T, V_i)$ ▷ Chop using m
5: $\quad$ **if** $k > 1$ **then**
6: $\quad\quad S \leftarrow \prod_{i=1,\ldots,k} \textsc{Learn}\text{SPN}(T, V_i)$
7: $\quad$ **else**
8: $\quad\quad T_1, \ldots, T_k \leftarrow slice_n(T_i, V)$ ▷ Slice using n
9: $\quad\quad S \leftarrow \sum_{i=1,\ldots,k} \frac{|T_i|}{|T|} \textsc{Learn}\text{SPN}(T_i, V)$
10: **return** S

---

scoring metric is different, both methods adhere to the following scoring procedure.

One feature $X_i \in V$ is chosen at random. Then the score $s(X_i, X_j)$ is computed, for all other features X_j. If $s(X_i, X_j)$ is greater than a threshold, then X_j is considered to be similar to X_i and they are grouped together. Next, $s(X_j, X_k)$ is computed, for all X_k not previously grouped with X_i. Again, if $s(X_j, X_k)$ is greater than a threshold, then X_k is considered similar to X_j, which means X_k is similar to X_i, and X_k is grouped with X_i and X_j. This process repeats until no more variables can be grouped with X_i.

G-test tests pairwise independence of features as follows:

$$G(X_i, X_j) = 2 \sum_{x_i} \sum_{x_j} c(x_i, x_j) log \frac{c(x_i, x_j)|T|}{c(x_i)c(x_j)}, \tag{9.69}$$

where $c(\cdot)$ counts the occurrences of a setting of a variable pair or singleton [82], and *log* is the natural logarithm. One of the threshold values used in [80] is 2.

Example 75. *Suppose g-test is used for chopping in* LearnSPN *when applied on the dataset in Table 9.1, which has four features X_0, X_1, X_2, and X_3, and ten instances $I_0, I_1, \ldots, I_9$ meaning $|T| = 10$ in (9.69). Further suppose that X_0 is chosen at random. It can be verified that $c(X_0 = 0, X_1 = 0) = 2$, $c(X_0 = 0) = 5$, and $c(X_1 = 0) = 6$. Then, from (9.69), we obtain:*

$$2 \ log \frac{2 \cdot 10}{5 \cdot 6} = -0.81.$$

After counting the other values of X_0 and X_1, (9.69) gives:

$$\begin{aligned} G(X_0, X_1) &= 2 \ (-0.81 + 1.22 + 1.15 + -0.69) \\ &= 1.74. \end{aligned} \tag{9.70}$$

Similarly,

$$G(X_0, X_2) = 0.40, \tag{9.71}$$

$$G(X_0, X_3) = 0.0. \tag{9.72}$$

Since all three g-test scores in (9.70)-(9.72) were less than the threshold value of 2, no cluster is formed. That is, g-test does not chop the dataset in Table 9.1 into smaller units.

The second method we consider for the chopping operation in Algorithm 5 is mutual information. *Mutual information* (MI) [14] tests pairwise independence of features as follows:

$$MI(X_i, X_j) = \sum_{x_i} \sum_{x_j} c(x_i, x_j) log \frac{c(x_i, x_j)}{c(x_i)c(x_j)}. \tag{9.73}$$

One of the MI threshold values used in [33] is 0.05.

Table 9.1: Dataset with 4 features and 10 instances.

	X_0	X_1	X_2	X_3
I_0	0	0	1	0
I_1	0	0	0	1
I_2	0	1	0	0
I_3	0	1	0	1
I_4	0	1	1	0
I_5	1	0	0	0
I_6	1	0	0	1
I_7	1	0	1	0
I_8	1	0	1	1
I_9	1	1	1	0

Example 76. *Suppose MI is used for chopping in* LearnSPN *when applied on the dataset in Table 9.1. Further suppose that X_0 is chosen at random. By (9.73), we obtain*

$$MI(X_0, X_1) = 0.12,$$
$$MI(X_0, X_2) = 0.03,$$
$$MI(X_0, X_3) = 0.00.$$

Since $MI(X_0, X_1) > 0.05$, we compute

$$MI(X_1, X_2) = 0.00,$$
$$MI(X_1, X_3) = 0.05.$$

As neither $MI(X_1, X_2)$ nor $MI(X_1, X_3)$ is greater than the threshold 0.05, the chopping method stops. Hence, the dataset in Table 9.1 is chopped into $\{X_0, X_1\}$ and $\{X_2, X_3\}$.

9.2.2 Slicing Methods

The slicing method of LearnSPN splits a dataset horizontally. Here we consider two slicing methods, namely, *k-means* [36] and expectation-maximization for fitting *Gaussian mixture models* [26].

K-means is a method for slicing (partitioning) the rows $I_0, \ldots, I_n$ of a dataset into k clusters. Each row I_i is assigned to the cluster k_j that minimizes the Euclidean

distance to the mean μ_j of that cluster:

$$||I_i - \mu_j||.$$

The mean μ_j is recalculated as the mean of those I_i assigned to cluster k_j. Then the above process is repeated using the recalculated μ_j. The process stops after 300 iterations or when all μ_j do not change. As in [80], here we focus on two clusters. For each cluster k_j, the initial mean μ_j can be randomly chosen without affecting the output of k-means.

Example 77. *Suppose k-means is used for slicing in* LearnSPN *when applied on the dataset in Table 9.1. Let the two initial means be*

$$\mu_0 = \langle 0, 0, 0, 0 \rangle,$$

and

$$\mu_1 = \langle 1, 1, 1, 1 \rangle.$$

Consider I_0 in Table 9.1. Its Euclidean distance to μ_0 is, by definition,

$$\begin{aligned} & ||I_0 - \mu_0|| \\ = & |(0-0)|^2 + |(0-0)|^2 + |(1-0)|^2 + |(0-0)|^2 \\ = & 1. \end{aligned} \tag{9.74}$$

Its Euclidean distance to μ_1 is

$$\begin{aligned} & ||I_0 - \mu_1|| \\ = & |(0-1)|^2 + |(0-1)|^2 + |(1-1)|^2 + |(0-1)|^2 \\ = & 3. \end{aligned} \tag{9.75}$$

By (9.74)-(9.75), I_0 is assigned to the first cluster k_0. $I_1, \ldots, I_9$ are assigned similarly. It can be verified that

$$k_0 = \{I_0, I_1, \ldots, I_7\} \tag{9.76}$$

and

$$k_1 = \{I_8, I_9\} \tag{9.77}$$

where ties are broken by assigning the instantiation to the first cluster k_0. The next step is to recalculate the mean μ_0 of (9.76) and μ_1 of (9.77). The latter is

$$\mu_1 = \langle 1, 0.5, 1, 0.5 \rangle.$$

Using the recalculated μ_0 and μ_1, the above process repeats. It can be verified that k-means results in clusters

$$k_0 = \{I_0, I_1, I_2, I_4, I_5, I_6, I_7, I_9\}$$

and

$$k_1 = \{I_3, I_8\}.$$

The second method we consider for the slicing operation in Algorithm 5 is *Gaussian mixture models* (GMM) using expectation-maximization. The process is nearly identical to that for k-means, except that the representation of each cluster and the assignment of instantiation to cluster are more involved. As in [80], we assume there are two clusters. Each cluster is a Gaussian distribution represented by its mean and co-variance matrix. These can be randomly assigned initially without disturbing the end result. Next, the probability of each instantiation I_i being in each cluster is computed. I_i is assigned to the cluster with the higher probability. Do this for every instantiation. The mean and covariance matrix are recalculated for each cluster by following the expectation-maximization algorithm. This process repeats 100 times or until the clusters do not change. More formally, each cluster k_j is represented by a Gaussian distribution, denoted $\mathcal{N}(I|\mu_j, \Sigma_j)$, where μ_j is its vector mean and Σ_j is its covariance matrix. The scoring of each instantiation I_i being in each cluster is the probability $p(k_j|I_i)$, which can be computed as:

$$p(k_j|I_i) = \frac{w_j \, \mathcal{N}(I|\mu_j, \Sigma_j)}{c},$$

where w_j is a cluster weight and c is a normalization function.

Example 78. *Suppose GMM is used for slicing in* LearnSPN *when applied on the dataset in Table 9.1. It can be verified that GMM results in clusters*

$$k_0 = \{I_0, I_1, I_2, I_4, I_5\}$$

and

$$k_1 = \{I_3, I_6, I_7, I_8, I_9\}.$$

9.3 A Variety of Possible SPNs

LearnSPN may yield a wide variety of SPNs from the same dataset, since a number of methods can be applied to chop and slice.

Example 79. *Recall the dataset in Table 9.1. Applying g-test for chopping yields*

$$\{X_0, X_1, X_2, X_3\},$$

meaning g-test did not chop the dataset into smaller units, as discussed in Example 75. On the contrary, Example 76 illustrates that MI will chop the same dataset into

$$\{X_0, X_1\}$$

and

$$\{X_2, X_3\}.$$

Similarly, when considering the slicing operation, k-means slices the dataset in Table 9.1 as

$$\{I_0, I_1, I_2, I_4, I_5, I_6, I_7, I_9\}$$

and

$$\{I_3, I_8\},$$

as shown in Example 77. In contrast, Example 78 slices the same dataset as

$$\{I_0, I_1, I_2, I_4, I_5\}$$

and

$$\{I_3, I_6, I_7, I_8, I_9\}.$$

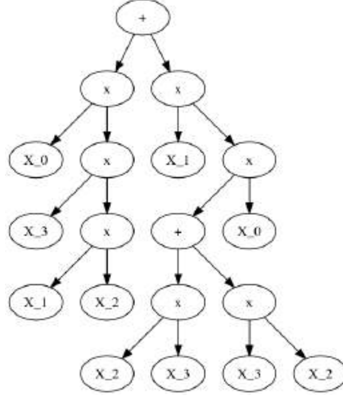


Figure 9.2: SPN $\mathcal{S}_1$ learned from the dataset in Table 9.1 using g-test for chopping and k-means for slicing.

The key point of Example 79 is that the output of LearnSPN is dependent upon the particular methods applied for chopping and slicing. As there are multiple combinations that can be used as methods within LearnSPN, each pair of which potentially gives rise to a different SPN.

Example 80. *Let us apply* LearnSPN *on the dataset in Table 9.1. If g-test is used for chopping in line 4 and k-means is used for slicing in line 8, then the learned SPN* $\mathcal{S}_1$ *is illustrated Figure 9.2. If g-test is used for chopping in line 4 and GMM is used for slicing in line 8, then the learned SPN* $\mathcal{S}_2$ *is shown Figure 9.3. If MI is used for chopping in line 4 and k-means is used for slicing in line 8, then the learned SPN* $\mathcal{S}_3$ *is depicted Figure 9.4. If MI is used for chopping in line 4 and GMM is used for slicing in line 8, then the learned SPN* $\mathcal{S}_4$ *is shown Figure 9.5.*

Example 80 highlights the fact that applying LearnSPN on that dataset in Table 9.1 can yield 4 different SPNs $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$, respectively depicted in Figures 9.2-9.5.

9.4 Experimental Analysis of SPN Depth

We perform an empirical study of LearnSPN by examining the various combinations of methods used to chop and slice. In this section, we focus on SPN depth.

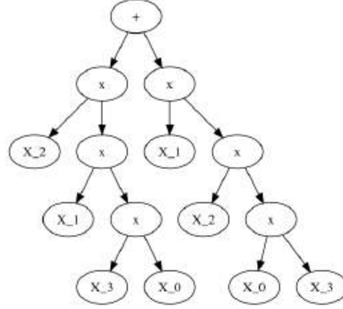


Figure 9.3: SPN $\mathcal{S}_2$ learned from the dataset in Table 9.1 using g-test for chopping and GMM for slicing.

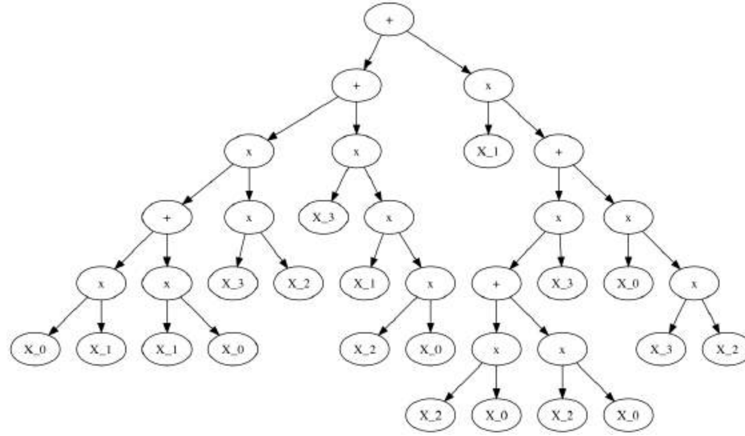


Figure 9.4: SPN $\mathcal{S}_3$ learned from the dataset in Table 9.1 using MI for chopping and k-means for slicing.

The *depth* [34] of an SPN $\mathcal{S}$, denoted $depth(\mathcal{S})$, is the number of nodes in the longest path in $\mathcal{S}$ from the root to a leaf.

Example 81. *Let us determine the depth of SPN $\mathcal{S}_1$ in Figure 9.2. Since the number of nodes in the longest path from a leaf node to the root node is 6, by definition,*

$$depth(\mathcal{S}_1) = 6.$$

Similarly, the depth of the SPNs in Figures 9.3, 9.4, and 9.5 are:

$$depth(\mathcal{S}_2) = 5,$$
$$depth(\mathcal{S}_3) = 7,$$

and

$$depth(\mathcal{S}_4) = 9.$$

We empirically evaluate each LearnSPN combination in 20 real-world benchmark datasets [48, 78]. The dataset characteristics are described in Table 9.2.

Our chopping thresholds are set as follows. The g-test threshold is set to 5, a value used in [80]. The MI threshold is set to 0.05 based upon [33]. The slicing thresholds are set by the default values in [80].

The implementation in [80] is a refinement of LearnSPN. More precisely, line 2 of Algorithm 5 is refined using two parameters α and m. The Laplace smoothing parameter α is set to 2.0 as done in [80]. The parameter m is a second stopping condition. It represents the minimum number of instances where chopping or slicing will still be applied and is set to 50 as in [80]. The refinement of Algorithm 5 is formally given in Algorithm 6, the version which we implemented.

Table 9.3 reports the depth of the SPN obtained from each of the four combinations. Winners are shown in boldface. Analysis of Table 9.3 reveals that MI and k-means is usually the pair that produces the deepest SPN. This pair yielded the

Figure 9.5: SPN $\mathcal{S}_4$ learned from the dataset in Table 9.1 using MI for chopping and GMM for slicing.

Algorithm 6 LearnSPN(T, V, α, m)

Input: a set of instances T over features V
Output: an SPN S
Main:

1: **if** $|V| == 1$ **then**
2: $S \leftarrow$ UNIVARIATEDISTRIBUTION(T, V, α)
3: **else if** $|T| < m$ **then**
4: $S \leftarrow$ NAIVEFACTORIZATION(T, V, α)
5: **else**
6: $\{V_j\}_{j=1}^{C} \leftarrow$ SPLITFEATURES(V, T) ▷ Chop
7: **if** $C > 1$ **then**
8: $S \leftarrow \prod_{j=1}^{C}$ LEARNSPN(T, V, α, m) ▷ Slice
9: **else**
10: $\{T_i\}_{i=1}^{R} \leftarrow$ CLUSTERINSTANCES(T, V)
11: $S \leftarrow \prod_{i=1}^{R} \frac{|T_i|}{|T|}$LEARNSPN$(T_i, V, \alpha, m)$
12: **return** S

deepest SPN in 16 out of 20 datasets. The other three pairs won only rarely.

These empirical findings are significant, since deeper networks have a larger *capacity* [34], i.e, they can encode a larger set of functions [24]. In other words, with each additional layer of depth, the set of distributions which can be efficiently captured grows in size [53]. While it has been conjectured that deeper models are more expressive than shallow ones, SPNs are the only deep learning model where this is provably the case [53]. For instance, the *network polynomial* [17] defined by the SPN $\mathcal{S}_1$ in Figure 9.2 cannot be efficiently represented by any of the other SPNs $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$ in Figures 9.3-9.5.

Let us focus in on the individual methods in Table 9.3. It can be seen that g-test for chopping yielded the deepest SPN in 3 out of 20 cases and tied for the deepest SPN in 3 other cases. That is, using MI for chopping yielded or tied for the deepest SPN in 17 out of 20 cases. With respect to slicing, GMM outperformed k-means in only 2 of 20 cases.

These results, when considering either individual methods or pairs of methods, suggest that MI and k-means yield the deepest SPNs.

Table 9.2: 20 Benchmark datasets used in unsupervised deep learning.

Dataset name	# features	# instances training	# instances validation	# instances testing
NLTCS	16	16181	2157	3236
MSNBC	17	291326	38843	58265
KDDCup2K	65	180092	19907	34955
Plants	69	17412	2321	3482
Audio	100	15000	2000	3000
Jester	100	9000	1000	4116
Netflix	100	15000	2000	3000
Accidents	111	12758	1700	2551
Retail	135	22041	2938	4408
Pumb-star	163	12262	1635	2452
DNA	180	1600	400	1186
Kosarek	190	33375	4450	6675
MSWeb	294	29441	3270	5000
Book	500	8700	1159	1739
EachMovie	500	4525	1002	591
WebKB	839	2803	558	838
Reuters-52	889	6532	1028	1540
20Newsgroup	910	11293	3764	1540
BBC	1058	1670	225	330
Ad	1556	2461	327	491

9.5 Experimental Analysis of SPN Accuracy

In this section, we study the accuracy of the SPNs learned from datasets.

Gens and Domingos [31] measure the quality of an SPN $\mathcal{S}$ using likelihood. The *likelihood* of an SPN $\mathcal{S}$, denoted $LL(\mathcal{S})$, is defined as the product of the values of $\mathcal{S}$, for each instantiation of the dataset. The higher the product, the better the quality.

Example 82. *Let us compute $LL(\mathcal{S}_1)$, where $\mathcal{S}_1$ is the SPN in Figure 9.2 learned from the dataset $I_0, \ldots, I_9$ in Table 9.1. Consider instantiation I_0, where $X_0 = 0$, $X_1 = 0$, $X_2 = 1$, and $X_3 = 0$. In Figure 9.2, consider all leaf nodes X_0. For each node X_0, in the univariate distribution over that X_0, use the probability value for $X_0 = 0$. Similarly, choose the probability value for $X_1 = 0$ in the univariate distributions for leaf nodes X_1, select the probability values for $X_2 = 1$ for leaf nodes X_2, and use the probability values for $X_3 = 0$ in the univariate distribution at leaf nodes X_3. Using*

Table 9.3: SPN depth for each combination.

Dataset name	(g-test, k-means)	(MI, k-means)	(g-test, GMM)	(MI, GMM)
NLTCS	15	**26**	17	19
MSNBC	**31**	**31**	19	21
KDDCup2K	69	**117**	23	39
Plants	31	**35**	21	23
Audio	25	**37**	21	23
Jester	13	**33**	19	21
Netflix	15	9	**25**	**25**
Accidents	**29**	25	27	27
Retail	31	**163**	27	25
Pumb-star	**27**	**27**	**27**	**27**
DNA	11	7	**15**	13
Kosarek	65	**137**	27	27
MSWeb	35	**99**	29	35
Book	63	**327**	17	71
EachMovie	27	**55**	21	27
WebKB	23	**65**	19	23
Reuters-52	31	**45**	27	27
20Newsgroup	33	**89**	31	29
BBC	**39**	27	19	17
Ad	79	**137**	47	77

g-test for chopping with threshold 2 and k-means for slicing with the default thresholds in [80] on the dataset in Table 9.1, and with respect to the structure of $\mathcal{S}_1$ *in Figure 9.2, the reader can verify that propagating to the root yields 0.053. After doing this for* $I_1, \ldots, I_9$*, the product of all ten root probabilities is*

$$LL(\mathcal{S}_1) = 0.081.$$

Similarly, we can score the quality of the other 3 SPNs $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$ in Figures 9.3-9.5 learned from Table 9.1. It can be verified that the scores of $\mathcal{S}_2$ in Figure 9.3, $\mathcal{S}_3$ in Figure 9.5, and $\mathcal{S}_4$ in Figure 9.4 are

$$LL(\mathcal{S}_2) = 0.073,$$

$$LL(\mathcal{S}_3) = 0.089,$$

and

$$LL(\mathcal{S}_4) = 0.094.$$

It follows that $\mathcal{S}_4$ is the best of the four possible SPNs. That is, the combination of MI for chopping and GMM for slicing yields the best SPN from the dataset in Table 9.1.

We now focus on real-world datasets. Using the same four combinations, we learn SPNs as described in the last section. Table 9.4 shows the accuracy of LearnSPN of each combination in 20 datasets. Winners are shown in boldface. Analysis of Table 9.4 is more involved than that of depth. Here, the pair for yielding the most accurate SPN seems to depend upon the dataset characteristics. For very small datasets with less than 20 features, MI and GMM produces the most accurate SPN. However, once the dataset has around 65 features, Table 9.4 shows that g-test and GMM is the most accurate pair. Continuing on, if the dataset has 100 - 135 features, our results imply that g-test and k-means is the pair yielding the most accurate SPNs. Finally, for the largest datasets with 160 - 1556 features, Table 9.4 demonstrates that g-test and GMM wins in 10 out of 11 cases. Thus, it seems that using g-test for chopping and GMM for slicing in LearnSPN will produce the most accurate SPNs from large datasets.

Let us scrutinize the individual roles of chopping and slicing in LearnSPN. Consider chopping in Table 9.4. Here, g-test was the chopping method used in 18 out of 20 winners, while MI was the chopping method used in 2 out of 20 winners.

Now consider slicing in Table 9.4. Here, k-means was the slicing method used in 6 out of 20 winners, while GMM was the slicing method used in 14 out of 20 winners. GMM won in the extreme datasets, namely, the least number of features and the most number of features.

9.6 Experimental Analysis of SPN Sparseness

In this section, we perform an empirical study of LearnSPN by examining SPN sparseness.

The sparseness of an SPN $\mathcal{S}$, denoted $sparseness(\mathcal{S})$, is defined as the number of edges in $\mathcal{S}$. An SPN will be called sparse if it has relatively few edges. In contrast, an SPN with many edges is called *dense* [70].

Table 9.4: SPN accuracy for each combination.

	(g-test, k-means)	(MI, k-means)	(g-test, GMM)	(MI, GMM)
NLTCS	-6.053	-6.070	-6.053	**-6.052**
MSNBC	-6.043	-6.046	-6.041	**-6.039**
KDDCup2K	-2.141	-2.173	**-2.139**	-2.148
Plants	-13.104	-13.489	**-13.022**	-13.107
Audio	**-40.103**	-40.580	-40.493	-40.623
Jester	**-52.967**	-53.097	-53.558	-53.585
Netflix	**-56.776**	-56.806	-57.685	-57.626
Accidents	**-31.062**	-36.035	-31.282	-35.588
Retail	**-10.967**	-12.092	-11.012	-11.417
Pumb-star	-25.885	-30.379	**-25.718**	-29.883
DNA	**-92.247**	-97.876	-93.458	-97.851
Kosarek	-10.966	-12.054	**-10.736**	-11.145
MSWeb	-10.115	-11.245	**-9.876**	-10.089
Book	-34.947	-38.934	**-34.372**	-37.275
EachMovie	-53.501	-55.520	**-51.527**	-51.902
WebKB	-156.319	-162.569	**-155.109**	-161.229
Reuters-52	-85.756	-91.971	**-84.600**	-89.083
20Newsgroup	-153.914	-161.183	**-153.373**	-158.472
BBC	-252.195	-257.876	**-251.165**	-256.474
Ad	-33.664	-45.920	**-30.638**	-41.751

Example 83. *Let us compute the sparseness of the SPN $\mathcal{S}_1$ in Figure 9.2. By definition,*

$$sparseness(\mathcal{S}_1) = 18,$$

since $\mathcal{S}_1$ has 18 edges. Similarly, the sparseness of $\mathcal{S}_2$ in Figure 9.3, $\mathcal{S}_3$ in Figure 9.4, and $\mathcal{S}_4$ in Figure 9.5 are

$$sparseness(\mathcal{S}_2) = 14,$$
$$sparseness(\mathcal{S}_3) = 36,$$

and

$$sparseness(\mathcal{S}_4) = 44.$$

Table 9.5: SPN sparseness for each combination.

Dataset name	(g-test, k-means)	(MI, k-means)	(g-test, GMM)	(MI, GMM)
NLTCS	**1090**	3993	1110	2007
MSNBC	5357	14790	**4356**	9132
KDDCup2K	6036	25022	**4161**	8744
Plants	12003	20487	**9410**	11148
Audio	11082	47617	**2511**	33215
Jester	**6782**	28784	10787	23977
Netflix	**14591**	45451	24945	40333
Accidents	24691	36015	**22007**	28227
Retail	3507	81985	**3040**	18162
Pumb-star	22505	30384	**20651**	27100
DNA	10059	9045	9715	**8689**
Kosarek	15297	92044	**7517**	26146
MSWeb	13608	64951	**8611**	19946
Book	18558	175609	**9668**	77018
EachMovie	22851	35560	**14461**	19502
WebKB	42333	95546	**32943**	57950
Reuters-52	70166	124696	**58552**	92590
20Newsgroup	148986	493183	**131822**	272316
BBC	50411	79519	**42361**	53200
Ad	31344	56023	**21447**	35242

In Example 83, $\mathcal{S}_2$ is the sparsest SPN of $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$, while $\mathcal{S}_4$ is the densest SPN. We now turn our attention to real-world datasets.

We consider the 20 benchmark datasets described in Table 9.2. An SPN is learned from each dataset using each of the 4 pairs of chopping and slicing method as mentioned in Section 9.4.

Table 9.5 reports our findings on SPN sparseness. The sparsest SPN, the one with fewest edges, is shown in boldface.

Analysis of Table 9.5 shows that using g-test for chopping and GMM for slicing very often yields the sparsest SPN. More precisely, this pair won for best in 16 out of 20 datasets. The second best pair, namely, g-test and k-means, won for best only in 3 of 20 datasets.

Let us scrutinize the individual methods in Table 9.5. It can be seen that MI for chopping yielded the sparsest SPN in 1 our of 20 cases. That is, using g-test for

chopping yielded for the sparsest SPN in 19 out of 20 cases. With respect to slicing, k-means outperformed GMM in only 3 of 20 cases.

These results, when considering either individual methods or pairs of methods, suggest that g-test and GMM yield the sparsest SPNs, which tend to be the most accurate SPNs in Table 9.4.

9.7 Conclusions

LearnSPN, the standard unsupervised learning algorithm for SPNs, can learn a wide variety of SPNs from the same dataset, as illustrated in Example 80. The reason for this is that there is more than one method for chopping and more than one method for slicing. To the best of our knowledge, we give the first empirical analysis of SPN depth, SPN accuracy, and SPN sparseness using 20 deep learning benchmark datasets. Our results in Section 9.4 show that the deepest SPNs tend to be obtained when using MI for chopping and k-means for slicing. This is significant, since deeper networks have a larger *capacity* [34], i.e, they can encode a larger set of functions [24]. On the other hand, the pair of g-test and GMM tend to yield the most accurate SPNs, as discussed in Section 9.5. These results suggest that MI and k-means may lead to *overfitting* [80]. Lastly, g-test for chopping and GMM for slicing often produce the sparsest SPN, as seen in Section 9.6. This is important as some SPN learning algorithms penalize SPNs containing more edges [49]. Future work can study the roles of slicing and chopping in extensions of LearnSPN, including *ID-SPN* [71], which incorporates direct and indirect variable interactions.

Chapter 10

Deep Convolutional Sum-Product Networks

10.1 Introduction

Generative models are of current interest in the deep learning community, including *generative adversarial networks* (GANs) [35], variational auto-encoders [41], neural autoregressive distribution estimators [45], pixel recurrent neural networks [59], and convolutional arithmetic circuits [76]. *Convolutional neural networks* (CNNs) [34] can be used in GANs. *Sum-product networks* (SPNs) [70] are a generative model that have received limited attention from the deep learning community [68]. An SPN is a *directed acyclic graph* (DAG), where leaf nodes are tractable distributions and each internal node is either a sum or product operation. A *valid* SPN defines a joint probability distribution and allows for efficient inference [70].

Conditions are given as to when subclasses of CNNs define valid SPNs, including convolutional layer filters of certain sizes and non-overlapping windows in sum-pooling layers. Satisfaction of these conditions yields a subclass of CNNs, called *convolutional* SPNs (CSPNs). CSPNs permit a vectorized representation allowing for exploitation of tensor libraries such as Tensorflow, but they also can suffer from being too shallow, and it is known that deep SPNs are more expressive than shallow SPNs [24].

We introduce *deep convolutional sum-product networks* (DCSPNs). DCSPNs permit the convolutional and sum-pooling layers to form rich DAG structures by augmenting layer tensors under conditions that maintain *decomposability* and *completeness*. As a decomposable and complete SPN is a valid SPN, our main result is that

DCSPNs are a larger subclass of CNNs that define valid SPNs. DCSPNs are a rigorous probabilistic model. As such, they can exploit probabilistic reasoning, including *marginal* inference and *most probable explanation* (MPE) inference. This allows an alternative method for learning DCSPNs using vectorized differentiable MPE. We show how to vectorize MPE using a mask algorithm and how it plays a role similar to the GANs generator. Image sampling is yet another application demonstrating the robustness of DCSPNs. This involves a minor modification to the mask algorithm. Our preliminary results on image sampling are promising, since the DCSPN sampled images exhibit variability. Experimental results on left- and bottom-completion like those in Table 10.1 show DCSPNs achieve results competitive to state-of-the-art by building deeper structures using both vertical and horizontal sum-pooling windows, which leverage local structure in the image data in both directions. Applying a simple low pass filter as a post-processing smoothing operation lowers the *mean squared error* (MSE) score from **910** to **802** for left-completion in Olivetti.

10.2 Sum-Product Networks

Nodes of an SPN can be organized as layers for a vectorized implementation. The discussion here draws from [79]. The SPN *input layer* is formed by the leaf distribution nodes. Let $\mathbf{L}(\mathbf{x}) \in \mathbb{R}^{\mathbf{s}}$ be the output of a generic layer with s nodes and input $\mathbf{x}$. The value of a *sum layer* with an input $\mathbf{x}$ from r input nodes is

$$\mathbf{L}(\mathbf{x}) = log(\mathbf{w} \times \mathbf{x}), \tag{10.78}$$

where $\mathbf{w} \in \mathbb{R}^{s \times r}$ is a matrix of weights defining sparse connections:

$$\mathbf{w}_{ij} = \begin{cases} w_{ij} & \text{if edge } (i,j) \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

Table 10.1: Mean squared error (MSE) scores in Olivetti Face.

	left	bottom
P&D [70]	942	918
ICNN [3]	833	-
D&V [25]	779	782
DCGAN [83]	1870	1414
DCSPN	**910**	**1006**

and $\mathbf{x} \in [0,1]^r$ represents the input probability values. Similarly, the value of a *product layer* with input $\mathbf{x}$ is

$$\mathbf{L}(\mathbf{x}) = exp(\mathbf{p} \times \mathbf{x}), \tag{10.79}$$

where $\mathbf{p} \in \{0,1\}^{s \times r}$ is a matrix of sparse connections:

$$\mathbf{p}_{ij} = \begin{cases} 1 & \text{if edge } (i,j) \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

and $\mathbf{x} \in [0,1]^r$ represents the input probability values in log-space. In this formulation, the exponential and logarithmic functions act as nonlinearities.

In this chapter, we relate SPNs with *convolutional neural networks* (CNNs) [34]. In general, CNNs are formed by convolutional and pooling layers. A *convolutional layer* can be constructed by applying a convolutional operation with a *filter* on a previous layer output. A *pooling layer* can be built by applying a max or average operation over some elements of the previous layer with respect to a sliding *window*. A sum-pooling layer can be easily obtained from an average-pooling layer.

10.3 The Building Blocks of DCSPNs

The discussion here is drawn from [5].

First, note that a vectorized SPN can represent a CNN. SPN sum layers correspond to CNN convolutional layers. Here, sum layer weights are convolutional filters and the sum layer value computes the convolution operation. Similarly, SPN product layers correspond to CNN sum-pooling layers. The sum-pooling window computes the product layer value in log-space.

Example 84. *Figure 10.1 illustrates an SPN with 3 layers being represented by a CNN with 3 layers, where colours represent node scopes. The input layer is the same for both, while the sum and product layers in the SPN are convolutional and sum-pooling layers in the CNN, respectively.*

Conversely, we show how a CNN can represent a vectorized SPN in log-space. Representational, convolutional, and sum-pooling layers in CNNs can represent the input, sum, and product layers in SPNs, respectively.

A *representational layer* [76] is formed by applying n representation functions $f_1, \ldots, f_n : \mathbb{R}^s \rightarrow \mathbb{R}$ over s-dimensional local patches of the dataset. We apply the logarithmic function in representational layers so as to correspond to SPN input layers in log-space. For instance, n Gaussian distributions can be used as representation functions to map patches of dataset instances to n values in the representational layer.

The value of a *convolutional layer* with input $\mathbf{x}$ is computed element-wise depending on filter $\mathbf{w}$'s size with depth c being either m-by-n (top) or height-by-width (bottom):

$$\mathbf{L}_{ij}(\mathbf{x}) = \begin{cases} \sum_{q=0}^{m-1} \sum_{l=0}^{n-1} log(\mathbf{w}_{ql}) + \mathbf{x}_{(i+q)(j+l)} \\ \sum_{k=0}^{c-1} log(\mathbf{w}_{ijk}) + \mathbf{x}_{ijk}. \end{cases} \tag{10.80}$$

Convolutional layers in (10.80) relate to sum layers in (10.78).

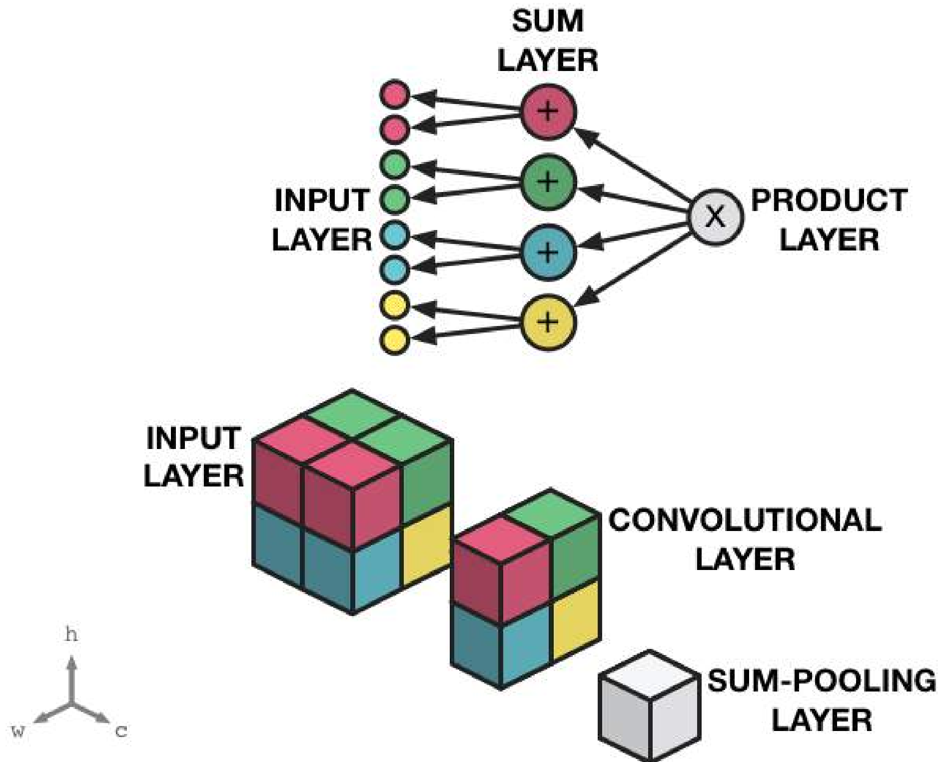


Figure 10.1: Input, convolutional, and sum-pooling layers in a CNN representing input, sum, and product layers in an SPN.

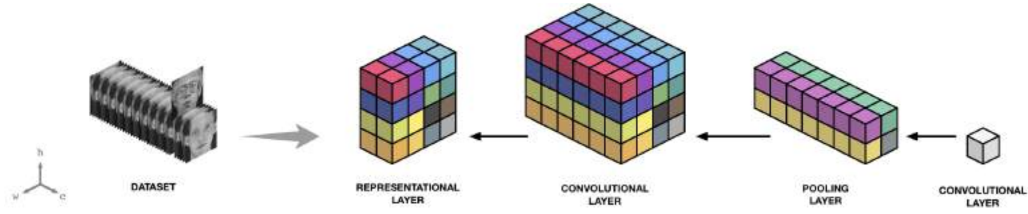


Figure 10.2: A CSPN represents a valid SPN and is vectorized, but also can suffer from being shallow.

The value of a *sum-pooling layer* with sliding window size m-by-n and input $\mathbf{x}$ is computed element-wise as:

$$\mathbf{L}_{ij}(\mathbf{x}) = \sum_{q=0}^{m-1} \sum_{l=0}^{n-1} \mathbf{x}_{(i+q)(j+l)}. \tag{10.81}$$

Sum-pooling layers in (10.81) relate to product layers in (10.79).

We now define a subclass of CNNs that represent SPNs by restricting convolutional and sum-pooling layers.

Definition 10.1. *A* convolutional SPN *(CSPN) is a CNN formed under the following two restrictions: (i) convolutional layer filters must have height and width of 1-by-1 or h-by-w, where h and w are the height and width of the convolutional layer, respectively; (ii) in sum-pooling layers, the horizontal and vertical stride of the sliding window must be at least the width and the height of the window itself, respectively.*

Example 85. *Consider the CSPN illustrated in Figure 10.2, where colours represent node scopes. The dataset is mapped to a 4-by-4 representational layer using 2 representation functions. Next, a convolutional layer is formed with 6 filters of size 4-by-4. A 2-by-2 sum-pooling layer is then built with a 2-by-2 sliding window. Finally, the root node is a 1-by-1 convolutional layer obtained using one 2-by-2 filter.*

In [5], it is shown that CSPNs represent valid SPNs.

Theorem 10.1. *Let $\mathcal{C}$ be a CSPN formed with respect to a CNN $\mathcal{C}'$ in Definition 10.1. Then, $\mathcal{C}$ is a* valid *SPN.*

CSPNs can struggle with depth, since the sum-pooling window size quickly reduces the size of the layers. For example, as depicted by the CSPN in Figure 10.2, the 4-by-4

representational layer reduced to the 1-by-1 root layer after two convolutions and one sum-pooling operation. As deep SPNs are more expressive than shallow SPNs [24], we now turn our attention to introducing deep CSPNs.

We build deep CSPNs by considering two related issues. First, we seek a DAG structure rather than a chain structure. Second, since each node is implemented as a tensor, each combination of tensors must be done such that a valid SPN is maintained. The next section formalizes these ideas.

10.4 Deep Convolutional SPNs

In this section, we introduce a tractable generative model, called *deep convolutional SPNs* (DCSPNs).

We denote by $\mathcal{T}$ a tensor of rank (order) n and dimension m in each mode. That is, $\mathcal{T}$ is a multi-dimensional array, specified by a shape with n indexes $[d_1, \ldots, d_n]$, each ranging in $[m] \equiv \{1, \ldots, m\}$. Without loss of generality, a layer is represented as a rank 4 tensor with shape $[b, h, w, c]$, where b is the batch (number of instances) being considered, and h, w, and c are the height, width, and channel (depth), respectively. Tensor elements are SPN nodes. In a sum layer tensor, elements are sum nodes, while in a product layer, tensor elements are product nodes.

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two tensors with the same height and the same width. The *channel augmentation* of $\mathcal{T}_1$ and $\mathcal{T}_2$ is the tensor with the same height and the same width formed by concatenating $\mathcal{T}_1$ and $\mathcal{T}_2$ with respect to the channel axis.

Example 86. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the two tensors in Figure 10.3 (i) (left, right), respectively. Then, the channel augmentation of $\mathcal{T}_1$ and $\mathcal{T}_2$ is the tensor depicted in Figure 10.3 (ii).*

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two tensors with the same depth and height. The *width augmentation* of $\mathcal{T}_1$ and $\mathcal{T}_2$ is the tensor with the same depth and the same height formed by concatenating $\mathcal{T}_1$ and $\mathcal{T}_2$ with respect to the width axis. The *height augmentation* is defined similarly.

Example 87. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the two tensors in Figure 10.3 (iii) (left, right), respectively. Then, the width augmentation of $\mathcal{T}_1$ and $\mathcal{T}_2$ is the tensor depicted in Figure 10.3 (iv).*

Definition 10.2. *A* deep convolutional sum-product network *(DCSPN) $\mathcal{D}$ over n variables $X_1, \ldots, X_n$ is a rooted DAG whose leaves are representational layers and whose internal nodes are convolutional and sum-pooling layers. A convolutional layer with more than one child is formed by recursively applying channel augmentation on all its children. A sum-pooling layer with more than one child is formed by recursively applying either height or width augmentation on all its children.*

Example 88. *Consider the DCSPN in Figure 10.4. The dataset is mapped to 2 representational layers, each using 2 representation functions. In the upper branches, after a convolutional and a sum-pooling layer, a channel augmentation is used to form a convolutional layer. In the lower branch, after a convolutional and sum-pooling layer, a width augmentation is used to form a sum-pooling layer. Lastly, a sum-pooling layer is followed by the root convolutional layer.*

By construction, DCSPNs represent SPNs. We now provide structural conditions under which DCSPNs represent valid SPNs.

Lemma 14. *Consider a DCSPN $\mathcal{D}$ where every convolutional layer has children over the same element-wise scope. Connect the children of each convolutional layer using channel augmentation. Then, $\mathcal{D}$ is a complete SPN.*

Proof. Consider the SPN defined by the DCSPN $\mathcal{D}$. Channel augmentation over all children of each convolutional layer yields tensors (layers) with elements of the same scope aligned along the channel axis. By construction, summation in the convolutional operation in (10.80) occurs over the channel axis. Hence, every summation involves operands of the same scope. By definition, $\mathcal{D}$ is complete. □

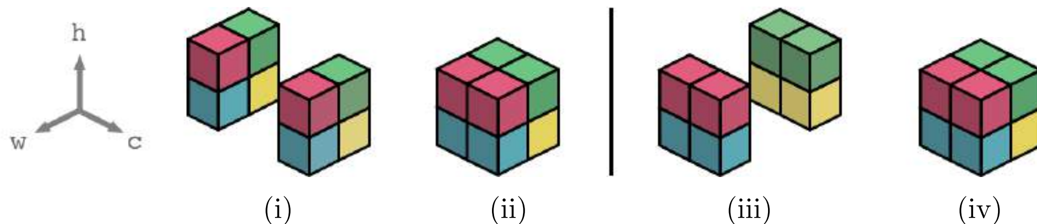


Figure 10.3: *Channel* (a)-(b) and *width* (c)-(d) augmentations.

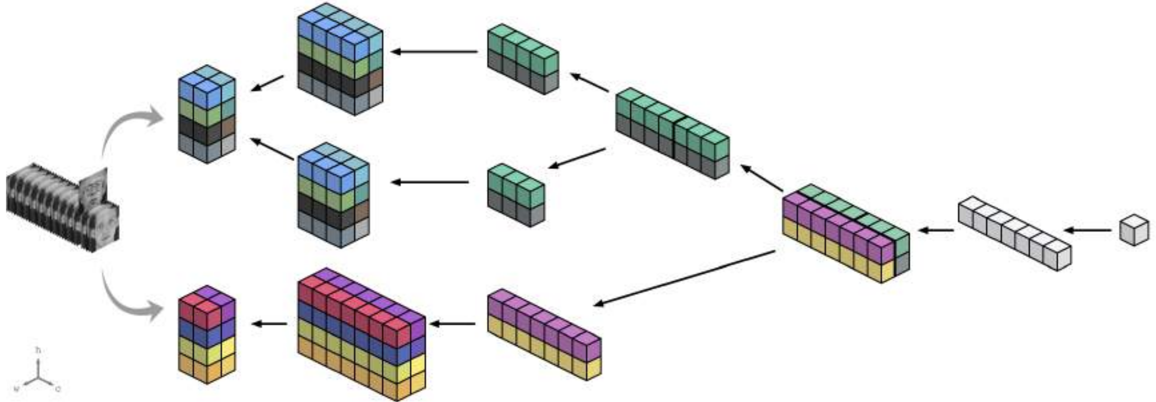


Figure 10.4: A DCSPN, depicting a rich DAG structure of convolutional and sum-pooling layers, while still being a valid SPN.

Lemma 14 is important as it establishes one practical method for combining tensors such that the resulting DCSPN is complete. Lemma 15, given next, provides a practical method for combining tensors such that the obtained DCSPN is decomposable.

Lemma 15. *Consider a DCSPN $\mathcal{D}$ where every sum-pooling layer has children with element-wise disjoint scopes. Connect the children of each sum-pooling layer using either height or width augmentation. Then, $\mathcal{D}$ is a decomposable SPN.*

Proof. Consider the SPN defined by the DCSPN $\mathcal{D}$. Width augmentation over all children of each sum-pooling layer yields tensors (layers) with elements of disjoint scopes at corresponding positions in the width axis. A similar property holds for height augmentation and the height axis. By construction, multiplication in the log-space sum-pooling operation in (10.81) occurs over either height or width or both axes. Thus, every multiplication involves operands of disjoint scopes. By definition, $\mathcal{D}$ is decomposable. $\square$

We now show the desired result.

Theorem 10.2. *Let $\mathcal{D}$ be a DCSPN in which the tensors of each convolutional and sum-pooling layer are connected in accordance to Lemmas 14 and 15. Then, $\mathcal{D}$ is a valid SPN.*

Proof. Consider the SPN defined by the DCSPN $\mathcal{D}$. By Lemma 14, $\mathcal{D}$ is necessarily a complete SPN. Moreover, $\mathcal{D}$ is guaranteed to be a decomposable SPN, by Lemma 15. By definition, since $\mathcal{D}$ is both complete and decomposable, $\mathcal{D}$ is a valid SPN. □

Theorem 10.2 is significant for two reasons. First, as DCSPNs define joint probability distributions, they are tractable deep generative models. Second, DCSPNs are robust in that there are many possible DAG structures in theory satisfying Theorem 10.2. We will examine later a specific DAG structure that performed exceptionally well in practice.

The DCSPN parameters can be learned with methods such as *expectation maximization* and *gradient descent*, similar to SPNs [70]. More specifically, given a DAG structure, we consider learning its parameters Θ (weights of sum nodes) with the maximum likelihood principle. This optimization problem can be equivalently seen as minimizing the *negative log-likelihood* (NLL) loss function $\mathcal{L}$ [76]:

$$\mathcal{L}(\Theta) = \mathbb{E}[-log\ \mathcal{S}(\mathbf{x})]. \tag{10.82}$$

Given a DCSPN DAG structure coupled with parameters, we turn our attention to applications demonstrating the effectiveness of DCSPNs.

10.5 Experiments

As promised, we now describe a DCSPN DAG structure that performed exceptionally well in practice. A convolutional layer follows every representational layer and every sum-pooling layer. All convolutional layers have filter sizes height-by-width matching the layer size. Two sum-pooling layers follow each convolutional layer: one with a window size of 1-by-2 and the other 2-by-1. Alternate the window sizes of 1-by-2 and 2-by-1 with 2-by-2 and 2-by-2 every n layers. This hyperparameter n is tuned per dataset and varied between 70 and 100 in our experiments. For each dataset, we randomly set aside one third (up to 50 images) for testing. For training, we use *ADAM* [40] with a learning rate of 0.005. Four Gaussian representation functions are used per pixel (variable), where the mean and variance are computed from equal quantiles of pixel intensities. In practice, we observed better accuracy when maintaining a sum operation in convolutional layers rather than a max operation during MPE. [70] made a similar observation.

Table 10.2: MSE scores in Caltech datasets.

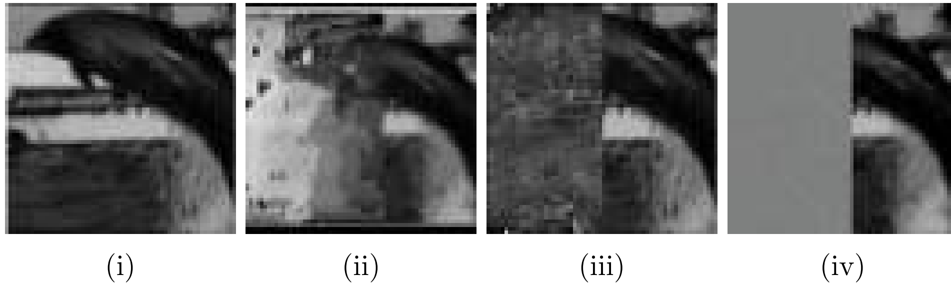
left	P&D	D&V	DCGAN	DCSPN
Face	1815	1657	1334	2356
Dolphin	3096	-	4096	4004
Helicopter	2749	-	3925	3404
bottom				
Face	1924	1517	1046	2298
Dolphin	2767	-	4016	4204
Helicopter	3064	-	3811	4206

We compare DCSPNs with *deep convolutional generative adversarial networks* (DCGANs) [83] using the publicly available code from [2]. Here, we use the hyperparameter values suggested in [2] and 100 epochs during training. The Poisson blending [69] post-processing technique for DCGANs is not implemented in [2].

Table 10.1 gives the *mean squared error* (MSE) scores for left-completion and bottom-completion in the *Olivetti* Face dataset [72]. We also compare competing methods in [70, 3, 25] with DCSPNs. For left-completion, DCSPNs score **910**, while for bottom-completion DCSPNs score (**1006**).

Table 10.2 shows left-completion and bottom-completion MSE scores in the *Caltech* datasets [28]. [25] did not report MSE scores for the Dolphin and Helicopter datasets. For bottom-completion, DCSPNs score **4204** and **4206** in Dolphin and Helicopter, respectively. Representative completions are illustrated in Figure 10.5.

Figure 10.5: Columns show original, DCSPN, DCGAN, and P&D. The first and second rows show left-completion and bottom-completions, respectively. Left and right pictures are from datasets Olivetti Face and Caltech Face, respectively.



(i) (ii) (iii) (iv)

Figure 10.6: DCSPNs performed well on a dataset with 65 images. Left-completion of the original image in (i) by DCSPN (ii), P&D (iii), and DCGAN (iv).

Analysis suggests several reasons why DCSPNs can reach competitive to state-of-the-art results. First, deeper structures are created by simultaneously deriving 1-by-2 and 2-by-1 sum-pooling window sizes. Second, alternating every n layers with window sizes 2-by-2 and 2-by-2 serves as a regularization technique, since larger windows tend to yield shallower DAGs. Third, the known vanishing gradient problem in SPNs [70] seems to be alleviated by alternating window sizes as above, since this has the effect of creating branches of different lengths. This is similar to how shortcuts work in residual networks [37]. Fourth, the vertical and horizontal windows of 1-by-2 and 2-by-1 leverage local structure in the image data in both directions. Fifth, accuracy was improved by using height-by-width instead of a 1-by-1 filter, that is, no sharing of parameters was more effective than sharing parameters. A similar finding was observed in [76].

DCSPNs left-complete well on a small dataset. The Caltech Dolphin dataset only contains 65 images. Nevertheless, DCSPNs performed quite well, as exemplified by the MSE scores in Table 10.2 and by the left-completions illustrated in Figure 10.6.

On the contrary, DCSPN completions admittedly look as though they are sometimes composed of random blocks of high frequency. To mitigate this, a low pass Gaussian filter can be applied as a smoothing post-processing step. This simple technique lowers the DCSPN MSE score in Olivetti left-completion in Table 10.1 from 910 to **802**.

We also tried other common CNN techniques, such as average-pooling instead of sum-pooling layers, sharing parameters in convolutional layers, batch normalization, and dropout, but did not observe any significant improvement. It is noted that

dropout was successfully applied in SPNs for image classification [68].

10.6 Conclusion

Deep convolutional sum-product networks (DCSPNs) can form rich DAG structures of convolutional and sum-pooling layers, while still being valid SPNs. As a tractable generative model, DCSPNs can perform efficient probabilistic reasoning, including marginal inference and approximate MPE inference. On the other hand, as a CNN, DCSPNs can build deeper structures using both vertical and horizontal sum-pooling windows, which leverage local structure in the image data. Practical applications of DCSPNs include image completion and image sampling. DCSPNs are flexible in that they allow for an alternative learning method based on differentiable MPE. Relationships between this learning approach and learning in GANs are discussed.

Experimental results show that DCSPNs results are competitive to state-of-the-art. For example, Table 10.1 reports the MSE scores for image left-completion in the benchmark Olivetti Face dataset. Applying a simple low pass filter as a post-processing step lowers the DCSPN MSE score down to 802.

Chapter 11

Future Work: Focused Learning in Sum-Product Networks

11.1 Introduction

We suggest a general two-phase method for further improving the accuracy of a learned SPN by detecting underperforming input variables and focusing learning only in the marginal formed by these variables. Consider an SPN learned over a dataset by maximizing the likelihood of the dataset over the network parameters. Moreover, consider an accuracy metric which uses a probabilistic inference task. We suggest two main phases: (i) detecting the underperforming variables; and, (ii) learning on them. For (i), we suggest using a *performance threshold* to partition the SPN variables into low- and high-performing groups. Here, the low- or high-performing variables define the low- or high-performing marginal, respectively. Using only the threshold might yield marginals with variables that are not correlated to each other, which goes against our goal of focused learning. To mitigate this scenario, we cluster together underperforming variables to create multiple size groups. Next, we introduce a *cluster size threshold* to consider only groups of that size or larger. For (ii), we perform marginal likelihood maximization. Here, the underperforming variables are considered missing from the dataset. In SPNs, missing variables are marginalized out of the underlying distribution.

We demonstrate the practicality of our two-phase method in image completion. An image completion is obtained from partial evidence by computing the MPE. Thus, the mean squared error between the MPE completion and the original image is the

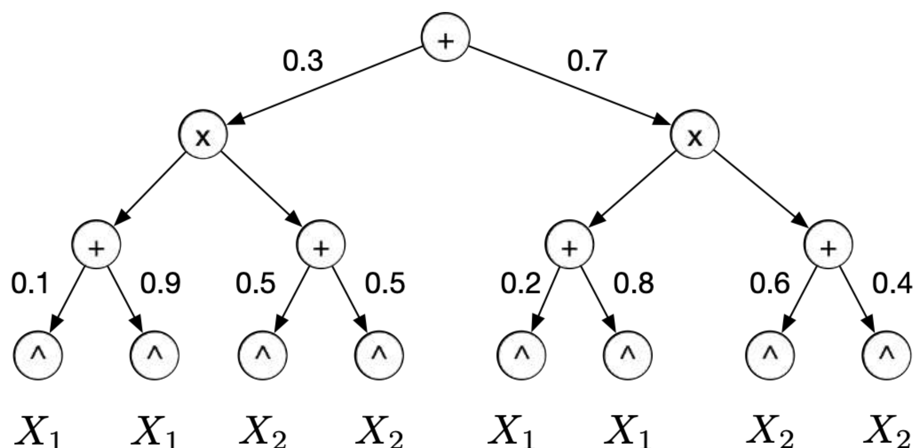


Figure 11.1: An SPN $\mathcal{S}$ over RVs $\mathbf{X} = \{X_1, X_2\}$.

accuracy metric. Low- or high-performing variables are good or poorly completed pixels accordingly to the performance threshold. Clusters of good and poor pixels can be formed using a spatial distance algorithm. Then, the cluster size threshold is applied leaving only large clusters of poor pixels. Lastly, by considering the poor pixels missing from the dataset instances, we perform marginal likelihood maximization.

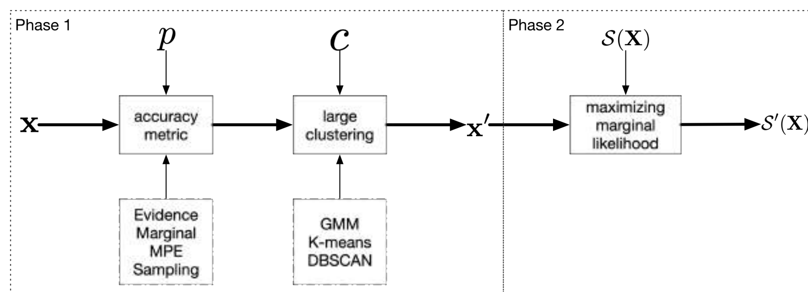


Figure 11.2: Two-phase method for focused in learning. Let $\mathbf{x}$ be an instance from the dataset. Phase 1 detects underperforming variables by using an accuracy metric with a performance threshold p and clustering them in large groups as determined by a cluster-size threshold c. Phase 2 maximizes the marginal likelihood from Phase 1.

11.2 A Two-Phase Method for Focused Learning

First, we recall that valid SPNs can be learned by maximizing the likelihood of a dataset over the network parameters. Given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L\}$ with L i.i.d. instances and an SPN $\mathcal{S}$ over variables $\mathbf{X}$, learning the sum node weights $\mathbf{W}$ of $\mathcal{S}$ corresponds to maximizing the likelihood $\mathcal{L}$:

$$max\ \mathcal{L}(\mathbf{W}; \mathbf{X}) = \prod_{l=1}^{L} \mathcal{S}(\mathbf{x}_l | \mathbf{W}). \tag{11.83}$$

Finding derivatives in SPNs is straightforward using backpropagation [70, 16]. Therefore, gradient methods such as gradient descent or expectation-maximization can be used to solve (11.83). Moreover, the structure of an SPN can be learned by using a divide-and-conquer approach [31, 80].

Various types of probabilistic inference query can be answered in an SPN, including computing the probability of evidence, marginals, most probable explanation (MPE), and generating samples. Given an SPN $\mathcal{S}$ over variables $\mathbf{X}$, we consider four types of SPN queries:

- **Evidence.** Input: $\mathbf{x} \in \mathbf{X}$. Output: $\mathcal{S}(\mathbf{x})$;
- **Marginal.** Input: $\mathbf{Y} \subseteq \mathbf{X}$. Output: $\mathcal{S}(\mathbf{y})$;
- **MPE.** $argmax_{\mathbf{x} \in \mathbf{X}} \mathcal{S}(\mathbf{x})$; and
- **Samples.** $\mathbf{x} \sim \mathcal{S}(\mathbf{X})$.

These queries can be used in various applications to measure specific task accuracy. For instance, in image completion, accuracy can be measured by the mean squared error between a given image and a completion obtained with MPE.

If we have a method for detecting underperforming input variables, then we could improve the SPN accuracy for an application by focusing learning in the marginal defined by these variables.

We formulate a two-phase method for detecting underperforming variables and focusing learning on them. This method is summarized in Figure 11.2.

We now describe the first phase. Detecting underperforming variables involves using an accuracy metric and clustering. First, given an SPN $\mathcal{S}$ and a dataset instance

$\mathbf{x}$, the accuracy metric should compute a discrete measurement of quality, for each input variable $X \in \mathbf{X}$. The metric can involve one or more inference queries in its computation. Then, a performance threshold p is applied to create a bipartite map of the input variables. Here, variables with an accuracy measurement below p are considered underperforming; otherwise, they are high-performing.

We want to cluster the underperforming variables together in order to focus learning on them. We use a clustering algorithm appropriate for the application. For instance, in image applications, clustering by spatial distance might be beneficial. Let the size of a cluster be the number of variables within its group. A cluster-size threshold c is used to select large clusters. Thus, the output of the first phase are large clusters of underperforming variables. These clusters together form a marginal $P(\mathbf{Y})$, with $\mathbf{Y} \subset \mathbf{X}$, of the SPN joint probability distribution $\mathcal{S}(\mathbf{X})$. We call $P(\mathbf{Y})$ a underperforming marginal.

The second phase performs focused learning. Here, the underperforming marginal $P(\mathbf{Y})$ from the first phase is the learning goal. The optimization problem is similar to (11.83), but instead of maximizing the likelihood, we use marginal likelihood:

$$max\ \mathcal{L}(\mathbf{W}; \mathbf{Y}) = \prod_{l=1}^{L} \mathcal{S}(\mathbf{y}_l | \mathbf{W}). \tag{11.84}$$

In SPNs, missing variables are marginalized out of the underlying distribution [70]. We assume that variables $\mathbf{Y}$ are missing, yielding partial evidence $\mathbf{y}_l$, even though the dataset instances specify complete evidence $\mathbf{x}_l$.

Algorithm 7 formalizes this two-phase learning method. The function $accuracyMetric$ uses a probabilistic inference query to compute a performance measurement, for each input variable $X \in \mathbf{X}$. Moreover, $accuracyMetric$ outputs the bipartite map with low- and high-performing variables based on the performance threshold p. The function $largeClusters$ uses a clustering algorithm to group underperforming variables. Then, $largeClusters$ outputs the large clusters based on the cluster-size threshold c. Lastly, Algorithm 7 learns new parameters $\mathbf{W}'$ and updates the SPN $\mathcal{S}$ with $\mathbf{W}'$.

11.3 Improving Image Completion

In this section, we show how the two-phase method can be used to improve SPN performance in image completion.

Algorithm 7 Focused Learning

Input: an SPN $\mathcal{S}(\mathbf{X};\mathbf{W})$, dataset $\mathcal{D}$, performance threshold p, cluster-size threshold c
for $\mathbf{x} \in \mathcal{D}$ **do**
▷ Phase 1
  $\mathbf{M} = accuracyMetric(\mathcal{S}(\mathbf{x}), \mathcal{D}, p)$
  $\mathbf{C} = largeClusters(\mathbf{M}, c)$
▷ Phase 2
  $\mathbf{W}' = argmax_{\mathbf{w}\in\mathbf{W}}\mathcal{S}(\mathbf{C}|\mathbf{w})$
  Update $\mathcal{S}$ with $\mathbf{W}'$
Return $\mathcal{S}$

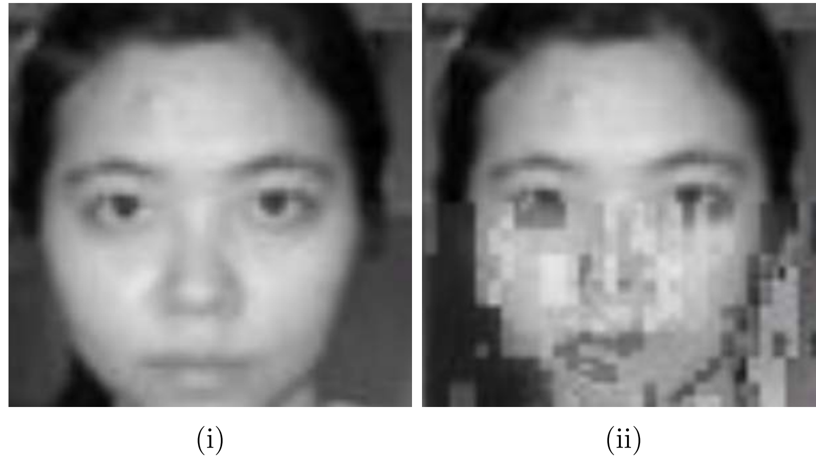
The motivation for further improvement comes from completions with local visual imperfections.

Example 89. *Consider the task of image completion illustrated in Figure 11.3 taken from [5]. Figure 11.3i is the original image. Consider the bottom-half missing. We use MPE to complete the missing part as shown in Figure 11.3ii. Although it seems that MPE is correctly completing the face, the completion itself looks blocky.*

These local imperfections are mostly due to structural characteristics of the learned SPN [70, 5]. In image completion, SPN graphs are usually designed in a way to capture local structure in the image [70, 25, 5]. While capturing local structure helps with generalization, it can also lead to local imperfections when learning challenges are observed, such as local noise or over- and under-fitting.

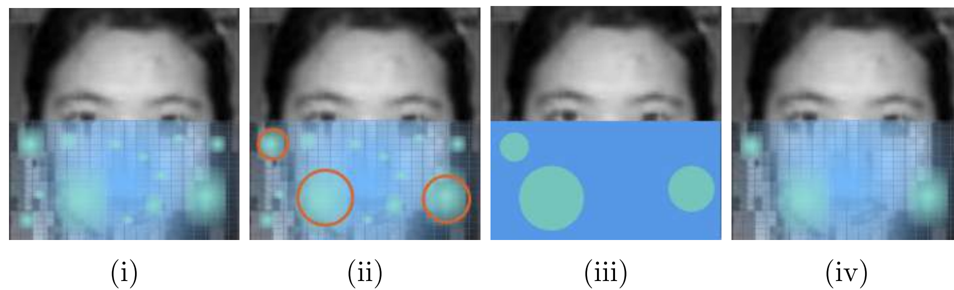
Given a dataset $\mathcal{D}$ of L images $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l\}$, consider each pixel being a RV $X \in \mathbf{X}$ and each RV having an univariate leaf distribution. We learn the parameters $\mathbf{W}$ of an SPN $\mathcal{S}(\mathbf{X})$ by maximizing the likelihood of $\mathcal{D}$ over $\mathbf{W}$ using (11.83). Image completion can be performed with MPE on $\mathcal{S}$. First, an image with missing pixels is provided. Missing pixels are seen as variables to be marginalized. In practice, this corresponds to leaf distributions assuming value 1. An upward phase in the SPN is performed with max functions instead of sum nodes [70]. Then, a downward phase is performed by choosing all children for product nodes and the maximizing child for max nodes. The maximizing value (mode) of the leaf distributions in the missing pixels form the MPE assignment for those variables.

We now show how to use the proposed two-phase method to improve local imperfections in image completion. In the first phase, we use the mean squared error



(i) (ii)

Figure 11.3: The MPE bottom-half completion of (i) yields a blocky texture in (ii).



(i) (ii) (iii) (iv)

Figure 11.4: Detecting an underperforming marginal involves (i) clustering underperforming variables (colour green), (ii) focusing on the large clusters by ignoring small clusters, as depicted in (iii). The underperforming marginal is given by the remaining clusters, as shown in (iv).

between the original image and the completion as the accuracy metric. In this way, we can measure the performance of each pixel individually. Assuming a performance threshold p applied with the accuracy metric to each pixel, a bipartite map is generated with low- and high-performing variables. This map is illustrated in Figure 11.4i, where the colours green and blue are low and high-performing variables, respectively. A clustering algorithm based on spatial distance is more appropriate to gather groups of pixels in two-dimensional images, such as DBSCAN [27]. In order to focus learning in general underperforming areas, our goal is to consider only large clusters, as highlighted in Figure 11.4ii. For that, we use a cluster-size threshold c to select only

large clusters, as shown in Figure 11.4iii. Thus, the final underperforming marginal, illustrated in Figure 11.4iv, is formed by the underperforming variables from the large clusters.

In the second phase, we perform marginal likelihood maximization as shown in (11.84). Here, the marginal considered is the output of the first phase. The underperforming pixels are considered missing from the input image. On the other hand, the high-performing pixels are considered evidence. As similarly done in MPE, missing variables are marginalized out of the SPN distribution, yielding value 1 for each of the missing variable's leaf distribution. Lastly, the SPN weights are updated with, for instance, gradient descent.

11.4 Discussion

In this section, we discuss one striking feature of the proposed method and as well as one drawback.

The two-phase method for focused learning is favorable in some practical aspects such as controlling overfitting with thresholds. Detecting underperforming variables involves using two threshold values, namely, the performance and the cluster-size thresholds. The performance threshold directly impacts the number of initial clusters since more underperforming variables facilitates more groups. The cluster size threshold influence the generality of focused learning procedure. The more small clusters that are avoided, the less specific that learning pass will be, since it will focus on fewer variables. Both thresholds act as a regularization parameters which might avoid overfitting. Future work will report on an empirical investigation on choosing the values of both thresholds.

On the other hand, the two-phase method can negatively disturb the distribution of the original SPN $\mathcal{S}$. Maximizing the marginal likelihood does not necessarily achieve the same solution as maximizing the SPN likelihood. In fact, this scenario is very unlikely. On the contrary, [76] has observed that maximizing the marginal can yield favorable results, compared to when the SPN likelihood is being maximized.

11.5 Conclusions

In this chapter, we suggested a two-phase method for focused learning. The main idea is to improve the performance of a previous learned SPN on a specific task. The first phase of the method detects underperforming variables by using a accuracy metric involving one or more probabilistic queries, and by applying a performance-threshold. The output of the first phase is a underperforming marginal. Then, a second phase updates the SPN weights by maximizing the marginal likelihood. Here, missing variables are marginalized out during learning. A characteristic of the two-phase method is the ability to control overfitting using its two thresholds. On the other hand, there is a risk of negatively disturbing the given SPN when maximizing the marginal instead of the SPN likelihood. Future work will empirically evaluate the two-phase method.

Chapter 12

Conclusion

Bayesian networks (BNs) and *sum-product networks* (SPNs) are two probabilistic graphical models for reasoning under uncertainty. In BNs, independencies among random variables are clearly defined by a directed acyclic graph, but inference is, in general, NP-hard. On the other hand, SPNs allows for tractable inference, while incorporating latent random variables that makes the model less clear.

We make several contributions in this thesis culminating with a powerful variation of SPNs, called *deep convolutional sum-product networks*. We start by introducing *Darwinian networks*, an alternative representation to BNs, which led to a new algorithm for testing independencies, namely *rp-separation*. Note that another practical application of DNs was *simple propagation* [10], which is the current state-of-the-art join tree inference algorithm in BNs.

Next, we suggest a clarification for scope inconsistencies in an SPN compiled from a BN. We also empirically explore learning SPN structure from data. Finally, we introduce DCSPNs as a deep tractable model for probabilistic inference. DCSPNs exploit the commonly used tensor libraries for neural networks, while still guaranteeing correctness as a PGM. Experimental results show that DCSPNs are comparable to state-of-the-art methods in image completion tasks.

References

[1] M. R. Amer and S. Todorovic. Sum-product networks for modeling activities with stochastic structure. In *Proceedings of the Twenty-Fifth IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1314–1321, 2012.

[2] B. Amos. Image Completion with Deep Learning in TensorFlow. `http://bamos.github.io/2016/08/09/deep-completion`, 2016. Accessed: July 1st, 2018.

[3] B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML)*, pages 146–155, 2017.

[4] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the Association for Computing Machinery*, 30(3):479–513, 1983.

[5] C. Butz, J. S. Oliveira, A. dos Santos, and A. L. Teixeira. Deep convolutional sum-product networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)*, pages 3248–3255, 2019.

[6] C. J. Butz, A. E. dos Santos, J. S. Oliveira, and C. Gonzales. Testing independencies in Bayesian networks with i-separation. In *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 644–649, 2016.

[7] C. J. Butz, J. S. Oliveira, and A. E. dos Santos. Darwinian networks. In *Proceedings of the Twenty-Eighth Canadian Artificial Intelligence Conference (AI)*, pages 16–29, 2015.

[8] C. J. Butz, J. S. Oliveira, and A. E. dos Santos. Determining good elimination orderings with darwinian networks. In *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 600–603, 2015.

[9] C. J. Butz, J. S. Oliveira, and A. E. dos Santos. On Darwinian networks. *Computational Intelligence*, 33(4):629–655, 2017.

[10] C. J. Butz, J. S. Oliveira, A. E. dos Santos, and A. L. Madsen. Bayesian network inference with simple propagation. In *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 650–655, 2016.

[11] N. A. Campbell and J. B. Reece. *Biology*. Pearson Benjamin Cummings, Berkeley, CA, 2009.

[12] Wei Chen Cheng, Stanley Kok, Hoai Vu Pham, Hai Leong Chieu, and Kian Ming Adam Chai. Language modeling with sum-product networks. In *Proceedings of the Fifteenth International Speech Communication Association*, pages 2098–2102, 2014.

[13] G. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

[14] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, NY, 2nd edition, 2012.

[15] J. A. Coyne. *Why Evolution is True*. Oxford University Press, New York, NY, 2009.

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

[17] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, Los Angeles, CA, 2009.

[18] Charles Darwin. *On the Origin of Species*. John Murray, London, UK, 1859.

[19] A. P. Dawid. Conditional independence in statistical theory. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(1):1–15, 1979.

[20] R. Dawkins. *The Selfish Gene*. Oxford University Press, New York, NY, 1976.

[21] R. Dawkins. *The Greatest Show on Earth: The Evidence for Evolution*. Free Press, New York, NY, 2009.

[22] R. Dechter. Bucket elimination: A unifying framework for reasoning. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 211–219, 1996.

[23] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41–85, 1999.

[24] O. Delalleau and Y. Bengio. Shallow vs. deep sum-product networks. In *Proceedings of the Twenty-Fourth Conference on Neural Information Processing Systems (NIPS)*, pages 666–674, 2011.

[25] A. Dennis and D. Ventura. Learning the architecture of sum-product networks using clustering on variables. In *Proceedings of the Twenty-Fifth Conference on Neural Information Processing Systems (NIPS)*, pages 2033–2041, 2012.

[26] Richard Duda, Peter Hart, and David Stork. *Pattern Classification*. John Wiley & Sons, New York, NY, 2012.

[27] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

[28] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[29] D. Geiger, T. S. Verma, and J. Pearl. d-separation: From theorems to algorithms. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 139–148, 1989.

[30] R. Gens and P. Domingos. Discriminative learning of sum-product networks. In *Proceedings of the Twenty-Fifth Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 3248–3256, 2012.

[31] R. Gens and P. Domingos. Learning the structure of sum-product networks. In *Proceedings of the Thirtieth International Conference on Machine Learning (ICML)*, pages 873–880, 2013.

[32] V. Gogate and P. Domingos. Structured message passing. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 252–261, 2013.

[33] V. Gogate, W. Webb, and P. Domingos. Learning efficient markov networks. In *Proceedings of the Twenty-Third Advances in Neural Information Processing Systems (NIPS)*, pages 748–756, 2010.

[34] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, Cambridge, MA, 2016.

[35] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proceedings of the Twenty-Seventh Conference on Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[36] T. Hastie, R. Tibshirani, and J. Friedman. *Overview of supervised learning.* Springer, New York, NY, 2009.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, pages 770–778, 2016.

[38] R. C. Holte, A. Felner, G. Sharon, N. R. Sturtevant, and J. Chen. MM: A bidirectional search algorithm that is guaranteed to meet in the middle. *Artificial Intelligence*, 252:232–266, 2017.

[39] F. V. Jensen and T. D. Nielsen. *Bayesian Networks and Decision Graphs.* Springer, New York, NY, 2007.

[40] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, pages 1–13, 2015.

[41] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[42] U. Kjærulff. Triangulation of graphs - algorithms giving small total state space. Technical Report R90-09, Aalborg University, Denmark, March 1990.

[43] U. B. Kjærulff and A. L. Madsen. *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*. Springer, New York, NY, 2nd edition, 2013.

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

[45] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 29–37, 2011.

[46] S. L. Lauritzen, A. P. Dawid, B. N. Larsen, and H. G. Leimer. Independence properties of directed Markov fields. *Networks*, 20(5):491–505, 1990.

[47] S. L. Lauritzen and D. J. Spiegelhalter. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of Royal Statistical Society*, 50(2):157–244, 1988.

[48] D. Lowd and J. Davis. Learning markov network structure with decision trees. In *Proceedings of the Tenth IEEE International Conference on Data Mining (ICDM)*, pages 334–343, 2010.

[49] Daniel Lowd and Pedro Domingos. Learning arithmetic circuits. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 383–392, 2008.

[50] A. L. Madsen and F. V. Jensen. Lazy propagation: A junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1-2):203–245, 1999.

[51] D. Maier. *The Theory of Relational Databases*. Computer Science Press, Orange, CA, 1983.

[52] G. Marchetti. Independencies induced from a graphical markov model after marginalization and conditioning: The R package ggm. *Journal of Statistical Software*, 15(1):1–15, 2006.

[53] James Martens and Venkatesh Medabalimi. On the expressive efficiency of sum product networks. *arXiv preprint arXiv:1411.7717*, 2014.

[54] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, 1997.

[55] K. Mohan and J. Pearl. On the testability of models with missing data. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 643–650, 2014.

[56] R. E. Neapolitan. *Probabilistic Methods for Bioinformatics: with an introduction to Bayesian networks*. Morgan Kaufmann, San Francisco, CA, 2009.

[57] A. S. Nobandegani and I. N. Psaromiligkos. A rational distributed process-level account of independence judgment. 2018.

[58] S. Olmsted. *On Representing and Solving Decision Problems*. PhD thesis, Stanford University, 1983.

[59] A. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1747–1756, 2016.

[60] J. Pearl. Fusion, propagation and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288, 1986.

[61] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, CA, 1988.

[62] J. Pearl. Belief networks revisited. *Artificial Intelligence*, 59(1):49–56, 1993.

[63] J. Pearl. *Causal inference in statistics: a primer*. John Wiley & Sons Ltd, Hoboken, NJ, 2016.

[64] J. Pearl and E. Bareinboim. External validity: From do-calculus to transportability across populations. *Statistical Science*, 29(4):579–595, 2014.

[65] R Peharz, B. C Geiger, and F. Pernkopf. Greedy part-wise learning of sum-product networks. In *Proceedings of the Joint European Conference on Machine Learning and and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 612–627, 2013.

[66] R. Peharz, R. Gens, F. Pernkopf, and P. Domingos. On the latent variable interpretation in sum-product networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(10):2030–2044, 2017.

[67] R. Peharz, G. Kapeller, P. Mowlaee, and F. Pernkopf. Modeling speech with sum-product networks: Application to bandwidth extension. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3699–3703, 2014.

[68] R. Peharz, A. Vergari, K. Stelzner, A. Molina, X. Shao, M. Trapp, K. Kersting, and Z. Ghahramani. Random sum-product networks: A simple but effective approach to probabilistic deep learning. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

[69] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on Graphics*, 22(3):313–318, 2003.

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

[71] A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *Proceedings of the Thirty-First International Conference on Machine Learning (ICML)*, pages 710–718, 2014.

[72] F. S. Samaria and A. C. Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of the Second IEEE Workshop on Applications of Computer Vision (WACV)*, pages 138–142, 1994.

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

[74] R. D. Shachter. Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 480–487, 1998.

[75] G. Shafer. *Probabilistic Expert Systems.* Philadelphia: Society for Industrial and Applied Mathematics, Philadelphia, PA, 1996.

[76] O. Sharir, R. Tamari, N. Cohen, and A. Shashua. Tensorial mixture models. *arXiv preprint arXiv:1610.04167*, 2018.

[77] B. Shipley. *Cause and Correlation in Biology.* Cambridge University Press, Cambridge, MA, 2016.

[78] Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1148–1154, 2012.

[79] A. Vergari, N. Di Mauro, and F. Esposito. Visualizing and understanding sum-product networks. *Machine Learning*, 108(4):551–573, 2019.

[80] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358, 2015.

[81] N. Wermuth and D. R. Cox. Joint response graphs and separation induced by triangular systems. *Journal of the Royal Statistical Society*, 66(3):687–717, 2004.

[82] Barnet Woolf. The log likelihood ratio test (the g-test). *Annals of Human Genetics*, 21(4):397–409, 1957.

[83] Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, pages 6882–6890, 2017.

[84] N. L. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Artificial Intelligence Conference*, pages 171–178, 1994.

[85] H. Zhao, M. Melibari, and P. Poupart. On the relationship between sum-product networks and Bayesian networks. In *Proceedings of Thirty-Second International Conference on Machine Learning*, pages 116–124, 2015.