

© Copyright by Ole Jakob Mengshoel, 1999

EFFICIENT BAYESIAN NETWORK INFERENCE: GENETIC ALGORITHMS,
STOCHASTIC LOCAL SEARCH, AND ABSTRACTION

BY

OLE JAKOB MENGSHOEL

S. Ing., University of Trondheim, 1989

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1999

Urbana, Illinois

Abstract

Bayesian networks are a core method for uncertainty representation and reasoning in artificial intelligence. This dissertation focuses on efficient approximate Bayesian network inference for computing the most probable explanation. New algorithms for efficient approximate inference are presented, and new techniques are described for creating hard synthetic Bayesian networks.

Three major research results related to speeding up Bayesian network inference are presented in this dissertation. First, improvements are made in the use of genetic algorithms. Local optima is an important problem in Bayesian network inference. Classical genetic algorithms converge, like hill-climbing algorithms, to one local optimum, while niching genetic algorithms converge to multiple local optima. This dissertation introduces the Probabilistic Crowding niching genetic algorithm, and presents theoretical and empirical results showing that Probabilistic Crowding gives predictable convergence, which at equilibrium is proportional to the utility function, which for Bayesian networks is the probability of an explanation.

Second, improvements are made to stochastic local search algorithms for efficient Bayesian network inference. This dissertation presents the Stochastic Greedy Search algorithm, which introduces noisy steps that allows local search to escape local optima. We also introduce different measures of gain (or gradient) and an operator-based approach, giving several ways to search locally. Comparisons to the state-of-the-art inference algorithm Hugin show that Stochastic Greedy Search performs significantly better for satisfiability Bayesian networks as well as for certain Bayesian networks from applications. In application networks, initialization algorithms, which compute the most probable explanation in bounding cases, prove to be very valuable, and we introduce two novel initialization algorithms denoted forward and backward dynamic programming. Initialization starts the local search at points closer to local optima than when search starts from an explanation created uniformly at random.

Lastly, improvements are made to the use and measurement of abstraction and aggregation to improve Bayesian network inference. A criterion is introduced that quantifies how different methods of abstraction impact the quality of inference. The criterion regards abstraction as noise and uses variance as a measure of quality. Results are presented that quantify how different methods and levels of abstraction effect accuracy.

Two major research results are presented that relate to creating hard synthetic Bayesian networks for empirical research on inference algorithms. One method translates deceptive problems studied in genetic algorithms to a Bayesian network setting. We show that Bayesian networks can be deceptive, and this is important since genetic algorithm performance suffers on deceptive problems. The other result is based on translating satisfiability problems into a Bayesian network setting, in a way that allows control over the level of difficulty of the Bayesian network inference problem. In particular, we show how graph connectivity, value of conditional probability tables as well as the degree of regularity of the underlying graph affect the speed of inference in Hugin and Stochastic Greedy Search.

To my family.

Acknowledgments

This thesis would not have been possible without my committee: David C. Wilkins, David E. Goldberg, Mehdi Harandi, and Sylvian Ray. Thanks to my advisor David C. Wilkins for getting me interested in uncertainty reasoning using Bayesian networks, and for supporting this research in so many ways, both intellectually and financially. Thanks to my co-advisor David E. Goldberg for sharing his expertise in and enthusiasm for genetic algorithms. Like David C. Wilkins and David E. Goldberg, Mehdi Harandi and Sylvian Ray have given comments and asked important questions at the right times, and in that way shaped the contents and form of the dissertation in significant ways. Last, but not least, many thanks to Dan Roth, who arrived to Illinois late in my graduate school career but still gave a lot of assistance and inspiration.

Thanks to former and present KBS and IlliGAL group members and visitors for comments and suggestions at various stages of this research: Mike Bobak, Vadim Bulitko, Erick Cantu-Paz, David Fried, Song Han, Tim Harahan, William Hsu, Ernest Kim, Fernando Lobo, Sam Mahfoud, Masaharu Munetomo, Martin Pelikan, Surya Ramachandran, Satish Shankarappa, Brent Spillner, and Misha Voloshin. Special thanks go to David Fried, Song Han, Tim Harahan, and Misha Voloshin for their help in developing the RAVEN tool-set, which has been used in much of the empirical part of this dissertation, as well as to Vadim Bulitko for providing comments to an earlier version of this dissertation.

Researchers who have developed and made their application Bayesian networks available for experimentation are also gratefully acknowledged, along with Nir Friedman who has collected these networks in a Bayesian Network Repository. At the time of this writing, the repository may be found at http://www-nt.cs.berkeley.edu/home/nir/public_html/Repository/index.htm. Also thanks to fellow members of the CoRAVEN team at UIUC and elsewhere, and in particular Peter Asaro, Robin Bargar, Caroline Hayes, Patricia Jones, Stu Schlabach, and numerous domain

experts, for co-developing the Pir Bayesian networks used in parts of this research. Kristian Kristensen is also gratefully acknowledged for giving permission to use his Barley Bayesian network for experimentation.

Members of the Hugin team, and in particular Marianne Bangsø, Frank Jensen, and Lars M. Nielsen are acknowledged for their prompt answers to my questions regarding the proper use of Hugin. In addition, some of the software used in this work exploited the GALib genetic algorithm package written by Matthew Wall at the Massachusetts Institute of Technology.

On a more personal note, I want to thank my family and friends in Norway and in the U.S. for their support over these years. In particular, I first want to thank my parents Knut and Astrid and my siblings Anne Torunn, Tore, and Per for their support and encouragement. In addition, thanks goes to Gunnar Blix who helped me when I started my graduate studies here at UIUC. Finally, thanks to my very special friend Lisa Kaufman, and the rest of the Kaufman family, for support and hospitality.

This research has partly been sponsored by ONR Grant N00014-95-1-0749, ARL Grant DAAL01-96-2-0003, and NRL Grant N00014-97-C-2061.

Table of Contents

Chapter 1	Introduction	1
1.1	Reasoning and Learning Under Uncertainty	2
1.2	Bayesian Networks	4
1.3	Bayesian Network Inference	11
1.4	Genetic Algorithms	13
1.4.1	The Simple Genetic Algorithm	13
1.4.2	Genetic Algorithm Theory	15
1.4.3	Genetic Algorithm Practice	16
1.5	Contributions of Dissertation	17
1.5.1	Problem Hardness: Deceptive and Satisfiability Bayesian Networks	17
1.5.2	Genetic Algorithm Niching: Probabilistic Crowding	18
1.5.3	Local Search: Stochastic Greedy Search	18
1.5.4	Problem Approximation: Abstraction in Bayesian Networks	18
1.6	Reader's Guide	19
Chapter 2	Bayesian Networks and Inference Algorithms	20
2.1	Bayesian Network Preliminaries	20
2.2	Exact Inference Methods	21
2.2.1	Inference by Network Propagation	22
2.2.2	Inference by Clustering	34
2.3	Inexact Inference Methods	42
2.3.1	Lin et al.	42
2.3.2	Rojas-Guzman and Kramer	45
2.3.3	Gelsema	47
2.3.4	Borghetti et al.	49
2.3.5	Welch	50
2.3.6	Miscellaneous	52
2.4	Summary	54
Chapter 3	The Probabilistic Crowding Genetic Algorithm	58
3.1	Integrated Tournament Algorithms	59
3.2	Probabilistic Crowding Algorithm	63
3.3	Analysis of Probabilistic Crowding	64
3.3.1	Two Niches, Same Jump Probabilities	64
3.3.2	Two Niches, Different Jump Probabilities	66
3.3.3	Multiple Niches, Different Jump Probabilities	68
3.4	Experiments Using Idealized Operators	70

3.4.1	Two Niches, Same Jump Probabilities	70
3.4.2	Multiple Niches, Same Jump Probabilities	72
3.5	Experiments Using Traditional Operators	72
3.6	Population Sizing	74
3.6.1	Population Sizing Theory	76
3.6.2	Population Sizing Experiments	77
3.7	Conclusion and Future Work	78
Chapter 4	Deceptive Bayesian Networks	80
4.1	Schemata, Deception, and Functions of Unitation	81
4.2	Constructing Bayesian Networks	84
4.3	Onemax Functions	87
4.4	Trap Functions	89
4.5	Hill Functions	93
4.6	General BNs from Deceptive Trap Functions	95
4.6.1	4-node Trap BNs	95
4.6.2	Solving the Equations for 4-node Trap BNs	96
4.6.3	Numerical Solutions	98
4.7	Conclusion and Future Work	100
Chapter 5	The Stochastic Greedy Search Algorithm	102
5.1	Stochastic Local Search	104
5.1.1	The Search Algorithm	104
5.1.2	Measure 1: BLANKET Gain	106
5.1.3	Measure 2: GMPE Gain	108
5.2	Operator-Based Stochastic Greedy Search	109
5.3	Initialization Algorithms	111
5.4	Experiments	112
5.4.1	Application Bayesian Networks: Different Search Operators	114
5.4.2	Application Bayesian Networks: Different Initialization Operators	115
5.5	Conclusion and Future Work	116
Chapter 6	Hard and Easy Bayesian Networks	123
6.1	Methodological Background	124
6.2	Distributions of Bayesian Networks	126
6.2.1	The SAT Model	127
6.2.2	The KD Model	130
6.3	Hard and Easy Networks	131
6.3.1	Threshold Phenomena C/V	131
6.3.2	The Nature of the CPTs	132
6.3.3	Regularity of Graphs	133
6.4	Experiments	134
6.4.1	Ratio Experiments	134
6.4.2	CPT Experiments	136
6.4.3	Regularity Experiments	137
6.5	Conclusion and Future Work	138

Chapter 7	Abstraction for Computing the Most Probable Explanation	143
7.1	Approximation, Bayesian Networks, and Genetic Algorithms	144
7.1.1	Abstraction and Refinement	145
7.1.2	Aggregation and Decomposition	145
7.1.3	Approximation and Genetic Algorithms	146
7.1.4	Noise and Genetic Algorithms	147
7.2	Preliminaries: Abstraction and Refinement Operators	147
7.2.1	Abstraction Hierarchy	147
7.2.2	Abstract State Spaces and Nodes	148
7.2.3	Abstract BNs and Explanations	154
7.3	Abstraction for the MPE Task	155
7.3.1	State Space Size	155
7.3.2	Quality of Abstraction	157
7.3.3	Difficulty of Obtaining Correctness	159
7.4	Abstraction Quality	160
7.4.1	Node Noise Variance	160
7.4.2	Bounding the Node Noise	162
7.4.3	Network Noise	163
7.5	Abstraction Algorithms	164
7.6	Abstraction Experiments	166
7.6.1	Node Experiments	166
7.6.2	MPE Experiments	168
7.7	Conclusion and Future Work	170
Chapter 8	Conclusion and Future Work	171
8.1	Conclusion	171
8.1.1	Genetic Algorithm Niching: Probabilistic Crowding	172
8.1.2	Problem Hardness: Deceptive Bayesian Networks	172
8.1.3	Local Search: Stochastic Greedy Search	173
8.1.4	Problem Hardness: Satisfiability Bayesian Networks	174
8.1.5	Problem Approximation: Abstraction in Bayesian Networks	175
8.2	Future Work	175
8.2.1	Marginalization	176
8.2.2	Hybrid Reasoning and Learning	176
8.2.3	Bayesian Network Approximation	177
Appendix A	Abstraction and Refinement Details	178
A.1	Abstraction and Refinement	178
A.2	Refinement	179
A.2.1	Fixed Children Method	180
A.2.2	Fixed Parents Method	182
A.3	Abstraction (Coarsening)	185
A.4	Proofs	187
Bibliography		189
Vita		201

List of Tables

2.1	Multiplication of belief tables.	36
2.2	Summary of experiments comparing iterative local search (ILS), simulated annealing (SA), and genetic search (GS) for two different BNs, ‘Reduced BN’ and ‘Full BN’. ‘N’, ‘V’, and ‘C’ give the total size of the BN, the number of disease (root) nodes, and the number of feature (leaf) nodes respectively. Note that these numbers are upper bounds since they apply to the BN before pruning, and this was performed before all search algorithms were used. The numbers in the table refer to the percentage of cases in which the best explanation was found.	44
2.3	Summary of experiment on the impact of string or graph representation of individuals in GALGO.	45
2.4	Experimental results for GALGO on BN1, BN2, and BN3. Differences in performance are shown for different scaling conditions: no scaling, uniform scaling, and inverse logarithmic scaling.	46
2.5	Results from Gelsema’s experiments with three BNs BN1, BN2, and BN3.	49
2.6	Root mean squared error (RMSE) for forward simulation versus hybrid of forward simulation and GA as a function of the number of evidence nodes.	52
2.7	Summary of previous research to BN inference using GAs. The ‘Who’ column gives initials for the researchers of the approach as follows. LGS: Lin, Galper, and Shachter. RGK: Rojas-Guzman and Kramer. G: Gelsema. SSW: Santos, Shimony, and Williams. W: Welch.	56
3.1	Two key dimensions of integrated tournament algorithms: nature of the acceptance rule and nature of the current state.	59
3.2	Experimental results for probabilistic crowding on the F1 and F2 functions, aggregated over all generations. The ‘P’ columns show predicted allocations, while the ‘M’ and ‘M+C’ columns show actual allocations for probabilistic crowding with mutation only and with mutation and crossover respectively.	74
3.3	Population size predicted from desired reliability, along with observed reliability.	78
4.1	Conditions for full deception in a bit string of length three.	83
4.2	Non-deceptive onemax fitness function defined over bitstring consisting of three bits.	87
4.3	Conditional probability tables of non-deceptive BN constructed from non-deceptive onemax fitness function.	88
4.4	Conditional probability tables of non-deceptive BN constructed from non-deceptive onemax fitness function with four bits.	88
4.5	Deceptive trap fitness function defined over bit string consisting of three bits.	90
4.6	Conditional probability tables of BN constructed from deceptive trap fitness function, before normalization.	91

4.7	Conditional probability tables of BN constructed from deceptive trap fitness function, after normalization.	91
4.8	Joint probability of deceptive BN.	92
4.9	Conditional probability tables of BN constructed from deceptive trap fitness function with four bits.	92
4.10	Hill fitness function defined over bit string consisting of three bits.	94
4.11	Conditional probability tables of BN constructed from hill fitness function with three bits.	94
4.12	Conditional probability tables of BN constructed from deceptive trap fitness function.	95
4.13	Conditional probability tables of BN constructed from deceptive trap fitness function.	99
5.1	Performance of Hugin on application Bayesian networks. The results in the Total column are for computing an MPE and are in seconds. The Compile and Execute columns give the time taken in each of these two phases; Props gives the number of propagations in the execution phase.	112
5.2	Performance of stochastic greedy search using different search algorithms on application Bayesian networks. T_{max} and F_{max} give information about the time-out limit and the max number of flips respectively. The <i>Mix</i> portfolio is a combination of the BM, GM, BS, and GS search operators and the UN, FS, FDP, and BDP initialization operators. The results in the T columns are for computing an MPE and are in seconds. The results in the S columns present the number of times an MPE was found. Time spent on initialization is included in the time taken for the SGS algorithm.	113
5.3	Performance of stochastic greedy search using different initialization algorithms on application Bayesian networks. The results in the T columns are for computing an MPE and are in seconds. The <i>Mix</i> portfolio is a combination of the BM, GM, BS, and GS search operators and the UN, FS, FDP, and BDP initialization operators. The results in the S columns present the number of times an MPE was found. Time spent on initialization is included in the time taken for the SGS algorithm.	113
6.1	Regular and irregular Bayesian networks classified along the two orthogonal dimensions of children-regularity and parent-regularity. The heading x/y of a cell (such as r/r) means that parent-regularity is x, children-regularity is y.	132
6.2	Speed (in seconds) of Hugin and SGS for MPE computation on Bayesian network translated from CNF formulas. Note that SGS is typically several orders of magnitude faster than Hugin.	135
6.3	Time (seconds) for Hugin to compute the MPE for various parameter settings of the KD model.	136
6.4	The effect of regular and irregular SAT BNs on Hugin computation time (seconds). .	137
6.5	The effect of regular and irregular SAT BNs on SGS/BM/NU computation time (seconds).	138
7.1	To the left, F 's conditional probability table is shown before abstraction of $\Omega_V = a, u, n$ to $\Omega_V = y, n$, to the right it is shown after abstraction.	154
7.2	To the left V 's CPT is shown before abstraction of $\Omega_V = a, u, n$ to $\Omega_V = y, n$, to the right it is shown after abstraction. Notice how these distributions are changed. . .	154
7.3	Abstraction hierarchies A , B , and C with internal and external node variances for different levels.	159

7.4	Results from using different approaches to constructing abstraction hierarchies on different BN nodes. External variance is between the given abstraction level and the ground level.	166
7.5	Results from MPE experiments using the Barley BN using different abstraction algorithms Random, MinDistance, and MinError. Each row in the tables above relates a Refined BN and an Abstracted BN. In the top table, the inference times and explanation probabilities are presented. In the bottom table, we focus on the trade-off between quality of abstraction and speed of inference.	168
A.1	To the left F 's CPT is shown before refinement, to the right it is shown after refinement. Notice how these child distributions are kept the same.	181
A.2	To the left V 's CPT is shown before refinement, to the right it is shown after refinement. Notice how these distributions are changed.	182
A.3	To the left, V 's CPT is shown before refinement, to the right it is shown after refinement.	182
A.4	To the left, F 's CPT is shown before refinement, to the right it is shown after refinement.	183
A.5	To the left, F 's CPT is shown before abstraction to the right it is shown after abstraction.	185
A.6	To the left, V 's CPT is shown before abstraction, to the right it is shown after abstraction.	185

List of Figures

1.1	The ‘family out’ Bayesian networks.	6
1.2	The ‘family out’ causal network.	7
1.3	The three node triplet configurations in a causal graph or Bayesian networks.	8
1.4	Classification and examples of Bayesian networks structure.	12
1.5	The simple genetic algorithm.	14
2.1	Bayesian networks inference by message passing or network propagation (from [Pearl, 1988]).	23
2.2	Example Bayesian networks; the ‘cheating spouse’ example.	28
2.3	Messages and node values in the ‘cheating spouse’ example.	29
2.4	Prior probability distribution in the ‘cheating spouse’ example.	29
2.5	Posterior probability distribution in the ‘cheating spouse’ example.	31
2.6	Propagating the effect of new evidence in a Bayesian networks.	32
2.7	Two examples of Bayesian networks and junction trees representing them. The Bayesian networks is shown to the left, the junction tree to the right.	35
2.8	The six steps involved in creating a junction tree from a Bayesian networks.	36
2.9	Absorption in a junction tree.	38
2.10	Propagation in junction tree of the ‘cheating spouse’ example based on evidence $DA = (1,0)$	39
3.1	Predicted results (solid circles) versus experimental results (open circles) for probabilistic crowding.	70
3.2	Predicted (solid circles) and experimental (open circles) results for niches \mathbb{X}_1 , \mathbb{X}_4 , and \mathbb{X}_8 . Predicted is steady-state expected proportion of population; experimental is measured proportion starting at generation one.	71
3.3	The test functions F1 and F2. F1 is to the left, F2 to the right.	73
3.4	Predicted versus actual allocation of individuals to niches for the test functions F1 and F2. Results for F1 are shown at the top, results for F2 are shown at the bottom.	75
4.1	Graph BN constructed from non-deceptive onemax fitness function.	88
4.2	Fully deceptive function over three bits.	89
4.3	Graph of BN constructed from deceptive trap fitness function with three bits.	93

5.1	Illustrating the criterion of choice concept for stochastic greedy search in the context of choosing between climbing north (N) and east (E). To the left, climbing N, which locally gives the best gain, is globally optimal. To the right, climbing N, which still gives the best gain locally, does not lead to a globally optimal solution. In order to reach the global optimum to the right, at least probabilistically, one can use a stochastic criterion of choice, such that one sometimes chooses E rather than N, even though the latter is locally superior.	103
5.2	The <i>SGS</i> stochastic greedy search algorithm.	118
5.3	The CHOOSE-NODE algorithm, used by <i>SGS</i>	119
5.4	The operator-based <i>SGS</i> stochastic greedy search algorithm.	120
5.5	Effect of using different <i>SGS</i> search operators on application BNs. The MPE computation time (bottom, log scale) and success ratio (top) for Hugin and different variants of <i>SGS</i> is shown. The same portfolio of initialization operators is used for <i>SGS</i> in all cases.	121
5.6	Effect of using different <i>SGS</i> initialization algorithms. The computation time (bottom) and success (top) for Hugin and different variants of <i>SGS</i> is shown. The same portfolio of search algorithms is used for <i>SGS</i> in all cases.	122
6.1	Example of constructing a BN from a CNF formula.	128
6.2	MPE computation time of Hugin and <i>SGS</i> on SAT BNs with $V = 30$ variables and $C = 60$ to $C = 126$ clauses.	140
6.3	Computation time (seconds) as function of C/V ratio in Hugin for KD networks. Note the phase transition around $C/V \approx 1.7$, supporting our claim that other models than the SAT model have phase transitions.	141
6.4	Computation time (seconds) as function of C/V ratio for irregular and regular SAT networks. Results for Hugin are shown at the top, results for <i>SGS</i> to the bottom.	142
7.1	Example of state space abstraction and refinement. Node R_1 contains the original states $r_{1,1}$ to $r_{1,16}$. Node R_2 contains states $r_{2,1}$ to $r_{2,8}$, where any state $r_{2,i}$ is an abstraction of two states $r_{1,2i-1}$ and $r_{1,2i}$ in R_1	150
7.2	Abstraction hierarchy for an agent monitoring node R_i . The abstraction hierarchy rooted at $r_{4,2}$ is similar to the hierarchy rooted at $r_{4,1}$, but has been left out of this figure.	150
7.3	Bayesian network showing abstraction and refinement principles.	151
7.4	Bayesian network with abstraction hierarchy. The Bayesian network, whose node V is abstracted, is shown to the left, the abstraction hierarchy is shown to the right.	152
7.5	Ordering of explanations according to their probability. In the top part of the figure, an example ordering for high quality abstraction hierarchies is shown. In the bottom part, an example ordering for low quality abstraction is shown.	157
7.6	Three possible abstraction hierarchies A , B , and C constructed for a BN node X	160
7.7	The MinDistance abstraction hierarchy construction algorithm.	165

List of Abbreviations

BDP Backward dynamic programming.

BKB Bayesian knowledge bases.

BN Bayesian network.

CNF Conjunctive normal form.

CPT Conditional probability table.

FDP Forward dynamic programming.

GA Genetic algorithm.

GIGA Gene-invariant GA.

GMPE Greedy MPE.

GSAT Greedy SAT.

MPE Most probable explanation.

PCGA Probabilistic crowding GA.

PRSA Parallel recombinative simulated annealing.

SAT Satisfiability.

SGS Stochastic greedy search.

SOF Survival of the fittest.

Chapter 1

Introduction

Complex, uncertain environments are of interest to many areas of research. For that reason, uncertainty reasoning based on numerous approaches, including probability theory, certainty factors, neural networks, and fuzzy logic has been investigated. Although probability theory seems like a natural point of departure for uncertainty reasoning, it was ignored for a long time because one believed probability theory would require too much information and give intractable algorithms.

Recent years have seen solutions to the problems of information requirements and algorithmic intractability. One factor is the technological advances in memory capacity and processor speed of computers. Another factor is the development of Bayesian networks and algorithms that exploit their structure, following Pearl's seminal work [Pearl, 1988] on local propagation in Bayesian networks (BNs). Still, it appears that uncertainty reasoning based on probability theory in general and Bayesian networks in particular has not reached its full potential. For example, exact computation in BNs with just a hundred or a few hundred nodes can be extremely slow. In the present work, we attack the problem of inefficiency of exact inference by focusing on approximations. The approximations are of two types, algorithmic approximations and BN approximations. By algorithmic approximation we mean to compute an answer to a computational problem without guaranteeing that this is an optimal solution to that problem. In other words, we investigate the trade-off between speed and accuracy in a Bayesian network context. Specifically, we are motivated by and investigate the computational problem of efficiently computing the most probable explanation in Bayesian networks. The two main algorithmic approximation techniques we investigate are stochastic, namely genetic algorithms and stochastic local search. Genetic algorithms (GAs) are inspired by nature and are used for adaptation, search, optimization, and learning in complex

and uncertain environments [Goldberg, 1989c]. Stochastic local search is based on the observation that by performing search in the neighborhood of an existing answer, one can iteratively make improvements leading to a good and even optimal answer.

In addition to algorithmic approximations using genetic algorithms and stochastic local search, we consider the problem of approximating a BN using another BN, again in the context of computing a most probable explanation. Once more, there is a trade-off between speed and accuracy: Computation in an approximated BN can be much faster than in the original BN, however at the cost of loss of accuracy.

The rest of this chapter is organized as follows. Section 1.1 describes uncertainty reasoning in AI and in decision theory, emphasizing the history and role of decision and Bayesian networks. Section 1.2 discusses knowledge representation using Bayesian networks, while Section 1.3 gives an overview of inference in Bayesian networks. Section 1.4 presents genetic algorithms. The contributions of this dissertation are discussed in Section 1.5, while Section 1.6 briefly presents the dissertation, chapter by chapter.

1.1 Reasoning and Learning Under Uncertainty

The decision sciences and artificial intelligence (AI) are both concerned with reasoning about action given incomplete information and scarce resources [Horvitz et al., 1988] [Henrion et al., 1991]. The decision sciences consist of Bayesian decision theory, the psychology of judgement, and their application in operations research and decision analysis [Horvitz et al., 1988]. The two sub-areas of the decision sciences that are of greatest interest to us are decision theory and decision analysis. Decision theory is made up of probability theory and utility theory. Decision analysis is concerned with elicitation, representation, reasoning, and search procedures. Since the focus of this dissertation is uncertainty reasoning, we will often not distinguish between decision theory and decision analysis in the following.

This section gives a brief introduction to Bayesian networks; for more detail we refer to introductory books [Pearl, 1988] [Neapolitan, 1990] [Russell and Norvig, 1995] [Jensen, 1996] [Castillo et al., 1997] and articles [Pearl, 1990] [Charniak, 1991] [Basye et al., 1992] [Spiegelhalter et al., 1993].

What does decision theory have to offer AI? Decision-theoretic techniques offer a principled way to reason under uncertainty and incompleteness, take into account complex preferences, and reason about decisions [Horvitz et al., 1988]. Decisions underlie actions that an agent may take, and as AI moves towards complex, real-world problems, uncertainty becomes a prominent feature. Belief and decision networks are at the center of attention in the decision-theoretic approach to AI. There are several reasons for this. First, uncertainty, both pertaining to environment estimation and decision outcome, are core AI concerns. Bayesian networks and decision networks are currently the dominant approach to uncertainty modelling in AI [Pearl, 1988] [Russell and Norvig, 1995], and decision theory, and in particular its subset probability theory, has recently been exploited in several areas of AI. These areas include expert systems, planning, machine learning, control of inference, perception, model construction, temporal reasoning, nonmonotonic reasoning, intelligent tutoring, and natural language processing [Horvitz et al., 1988]. Second, the basis of Bayesian networks and decision networks in probability and decision theory give them a sound scientific foundation as well as an ‘interface’ to other disciplines such as mathematics, statistics, operation research, and economics. Third, Bayesian networks and decision networks have proven useful in applications. Evidence of this is the availability of shells for development of Bayesian networks and decision networks as well as descriptions of applications in the literature.

In the following, we focus on decision-theoretic expert systems, since this is historically the most prominent area [Horvitz et al., 1988]. Decision-theoretic (or normative) expert systems were among the earliest expert systems. Early diagnostic expert systems, developed during the 1960s, were based on probability theory. The following simplifying assumptions were adopted in these early decision-theoretic expert systems. First, that hypotheses (diseases) are mutually exclusive and collectively exhaustive. Second, that any piece of evidence (symptom) is conditionally independent of any other piece of evidence given the hypotheses. Suppose H , E_i and E_j are random variables. Let h be a hypothesis, and e_i and e_j two pieces of evidence. Then the conditional independence assumption amounts to letting $\Pr(E_i = e_i \mid H = h) = \Pr(E_i = e_i \mid H = h, E_j = e_j)$. These two assumptions lead to simpler probability specification and belief propagation. Expert systems based on the above two assumptions performed at the level of experts or better, but their acceptability did not match their performance.

During the 1970s, the rule-based and heuristic paradigms became alternatives to the decision-theoretic paradigm. Two well-known rule-based expert systems that incorporate uncertainty representation and computation are MYCIN [Buchanan and Shortliffe, 1984] and PROSPECTOR [Duda et al., 1976]. The uncertainty calculations in these and similar expert systems were regarded as approximations to the probabilistic ideal. MYCIN and PROSPECTOR are in many ways impressive systems. However, their rule-based approach to reasoning under uncertainty is limited. The areas where rule-based inference prove to be most problematic are in the treatment of prior beliefs and in the assumption of modularity [Horvitz et al., 1988]. Furthermore, there exist empirical studies showing that the uncertainty calculations used in an expert system matter when evidence is weak or conflicting. Thus there are both theoretical and practical reasons to consider alternatives to the rule-based approach to uncertainty.

The limitations of the rule-based approach to expert systems led to a renewed interest in decision theory in the 1980s and 1990s. Research on decision theory for expert systems can be split into knowledge representation, knowledge engineering (or modeling), belief propagation, and explanation [Horvitz et al., 1988]. These areas of research are also of interest to the field of AI as a whole. For purposes of knowledge representation, the notion of a decision basis defines a complete representation of a decision problem. A *decision basis* consists of components representing alternatives, states, preferences, and relationships in a decision situation. Numerous representations of a decision basis have been developed. We will focus on decision networks (or influence diagrams), which is a graphical representation of the decision basis. A *decision network* is an directed acyclic graph (DAG) where the nodes represent properties or quantities of interest. The arcs represent influences between nodes. The nodes in a decision network are of three types: decision nodes, chance nodes, and value nodes. A *Bayesian networks* is a decision network with only chance nodes. The next section describes Bayesian networks knowledge representation, while later sections focus on inference in Bayesian networks.

1.2 Bayesian Networks

BNs (also known as Bayesian, causal, or probabilistic networks) are currently the dominant uncertainty knowledge representation technique in AI [Pearl, 1988] [Neapolitan, 1990] [Russell and

Norvig, 1995] [Jensen, 1996]. Bayesian networks are directed acyclic graphs (DAGs) where nodes represent random variables, and edges represent conditional dependence between random variables. The reason why Bayesian networks are important is their theoretically sound foundation in probability and graph theory, their usefulness in diverse AI contexts, as well as their proven strength in applications.

The notation used in the following is this: Random variables are represented using italic uppercase letters: A, B, C, \dots, X, Y, Z . Values of random variables are italic lowercase letters, for example if A has n values we have values a_1, \dots, a_n , and when A is instantiated to a_i we say $A = a_i$. When the values of a set of random variables are disjoint, we may abbreviate by omitting the random variables. For example, $A = a_i$ would be equivalent to a_i . For a set of (instantiated) random variables we use bold italic uppercase (lowercase) letters. For example, rather than listing the random variables V_1, \dots, V_n , we may say \mathbf{V} . And a set of instantiated random variables $E_1 = e_1, \dots, E_m = e_m$ may be abbreviated as \mathbf{e} (for evidence).

Bayesian networks encode joint probability density functions in a sparse manner. Let V_1, \dots, V_n be random variables. Then the joint probability density function (pdf) is denoted by $\Pr(V_1, \dots, V_n)$, and one can specify the joint pdf as an exponentially sized table—a joint pdf table. For theoretical purposes, working with the joint pdf table is fine, since the conditional probability of a random variable V given observed evidence \mathbf{e} is defined as:

$$\Pr(V \mid \mathbf{e}) = \frac{\Pr(V, \mathbf{e})}{\Pr(\mathbf{e})}$$

if $\Pr(\mathbf{e}) > 0$, else it is undefined. However, as a basis for an inference algorithm the explicit representation and manipulation of a joint pdf table poses problems when there is a large number of random variables. First, the joint pdf table needs exponentially many numbers to be completely specified. Second, even if such a joint was given, it would taken exponential time to manipulate it, for example when marginalizing.

A Bayesian networks addresses these problems by representing a joint pdf intensionally rather than extensionally. If we don't distinguish between random variables and nodes, a Bayesian networks consists of:

- A set of nodes (random variables) and a set of directed edges between the nodes. A node has a finite, mutually exclusive and exhaustive set of states. Real-valued nodes can also be used, and even mixed with discrete-valued nodes [Olesen, 1993], however this dissertation focuses on the discrete case.
- The nodes (vertices) and the directed edges form a directed acyclic graph (DAG).
- For each variable U with parents V_1, \dots, V_n , there is a conditional probability table $\Pr(U | V_1, \dots, V_n) = \Pr(U | \Pi_U)$, where Π is a function giving a node's parents.

An example should make these concepts clear. Consider the following example, originally introduced by Charniak [Charniak, 1991]:

Suppose when I go home at night, I want to know if my family is home before I try the doors. (Perhaps the most convenient door to enter is double locked when nobody is home.) Now, often when my wife leaves the house, she turns on an outdoor light. However, she sometimes turns on this light if she is expecting a guest. Also, we have a dog. When nobody is home, the dog is put in the back yard. The same is true if the dog has bowel troubles. Finally, if the dog is in the back yard, I will probably hear her barking (or what I think is her barking), but sometimes I can be confused by other dogs.

We will refer to this as the ‘family out’ example; it can be formalized as a Bayesian networks as shown in Figure 1.1. In Figure 1.1, the oval nodes represent random variables. Each node is binary. For example, FO has values fo and $\sim fo$. In other words, the family can be out ($FO = fo$, abbreviated fo) or not out ($FO = \sim fo$, abbreviated $\sim fo$). Prior and conditional probabilities are shown in tabular format in the figure. For example, the table at the top left stands for $\Pr(FO)$: $\Pr(fo) = 0.15$ and $\Pr(\sim fo) = 0.85$. Notice that $\Pr(fo) + \Pr(\sim fo) = 1$. In the figure, conditional probabilities are presented similarly in conditional probability tables. For example, $\Pr(do | fo, bp) = 0.99$ and $\Pr(\sim do | fo, bp) = 0.01$ are presented in the left-most row in the $\Pr(DO | FO, BP)$ table.

There are two aspects to any Bayesian networks: a qualitative aspect and a quantitative aspect. The *qualitative aspect* of a Bayesian networks concerns the causal network's (or the DAG's) structure. Figure 1.2 shows the causal network for the ‘family out’ example. In a causal network, the

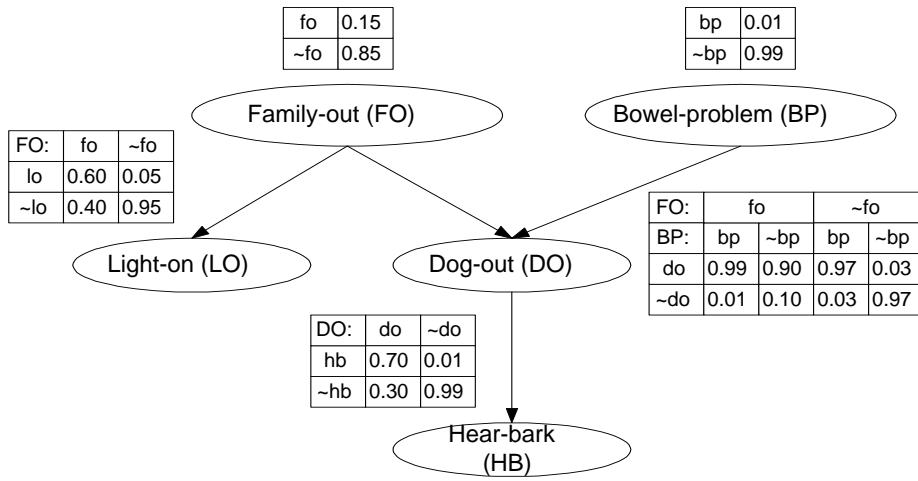


Figure 1.1: The 'family out' Bayesian networks.

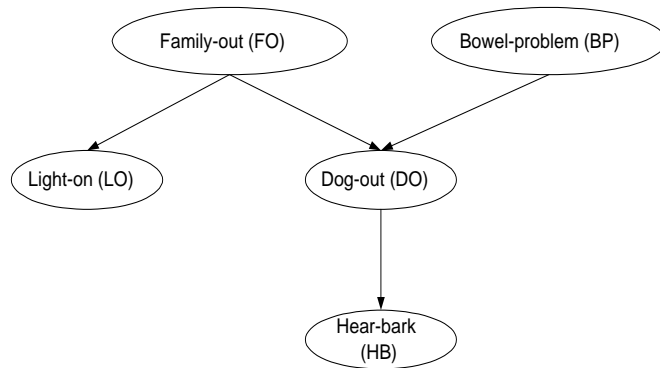


Figure 1.2: The 'family out' causal network.

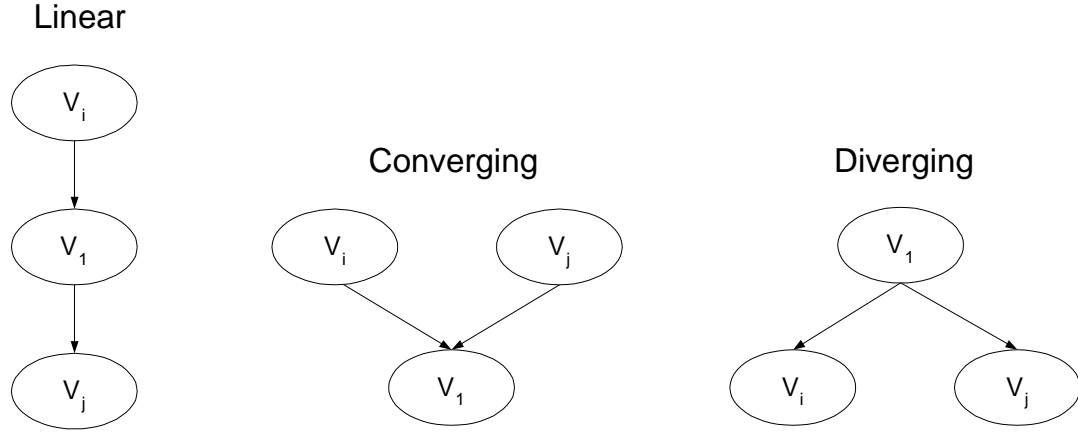


Figure 1.3: The three node triplet configurations in a causal graph or Bayesian networks.

direction of edges connecting nodes is important. We distinguish between three types of connections in causal networks: linear, diverging, and converging. These connection types are illustrated in Figure 1.3. Consider a node V_1 and two of its neighbors V_i and V_j in an arbitrary causal network. The structure of the subgraph induced by V_1, V_i , and V_j is one of:

1. linear: $V_i \rightarrow V_1 \rightarrow V_j$. V_i is V_1 's parent, which again has V_j as child.
2. converging: $V_i \rightarrow V_1 \leftarrow V_j$. V_i and V_j are both among V_1 's parents.
3. diverging: $V_i \leftarrow V_1 \rightarrow V_j$. V_i and V_j are both among V_1 's children.

This terminology can be used about the node V_1 as well as about a node triplet (V_i, V_1, V_j) . For example, in Figure 1.2 the node Dog-out is linear if Family-out, Dog-out, and Hear-bark are considered; it is converging if Family-out, Dog-out, and Bowel-problem are considered. The node triplet Light-on, Family-out, and Dog-out gives an example of the diverging connection type.

The three triplet configurations determine when uncertainty propagation between two nodes takes place along a path in the underlying undirected graph of a causal network. Consider a path P between two nodes U and V in the underlying undirected graph of a causal network. The path P is called *d-separating* if there is an intermediate variable E along the path such that:

- the connection is serial or diverging and the state of E is known, or
- the connection is converging, and neither the state of E nor of any of its descendants is known.

A path that is not d-separating is called a *d-connecting* path.

An important idea regarding uncertainty propagation between two nodes is d-separation. Two variables in a causal network are *d-separated* if all paths between them are d-separating paths. Two variables that are not d-separated are *d-connected*. We will come back to the concepts of d-separated and d-connected after discussing the quantitative aspects of Bayesian networks.

The *quantitative* aspect of a Bayesian networks concerns how concepts from probability theory are added to the causal network, resulting in a Bayesian networks. Since a causal network is a DAG, it's nodes can be sorted topologically into a sequence V_1, \dots, V_n . Corresponding to this sequence, we can use the chain-rule representation of a joint pdf:

$$\Pr(V_1, V_2, \dots, V_n) = \Pr(V_1) \Pr(V_2 | V_1) \cdots \Pr(V_n | V_{n-1}, \dots, V_1).$$

Since each V_i is a node in the causal graph, each factor on the right-hand side specifies the conditional probability for some node V_i in the causal network (we do not distinguish between nodes and random variables). Let Π_V be the set of parents of V . It seems reasonable that

$$\Pr(V_i | V_{i-1}, \dots, V_1) = \Pr(V_i | \Pi_{V_i}),$$

i.e. that V_i is conditionally independent of all other nodes given its parents. Assuming this is the case, it makes sense to specify probabilities in a Bayesian networks by specifying the conditional probability of a node given its parents. More formally, we have the following theorem.

Theorem 1 *Consider a Bayesian networks over the set of nodes V_1, \dots, V_n . Let each node V_i be defined conditionally in terms of its parents Π_{V_i} . Then the joint pdf $\Pr(V_1, \dots, V_n)$ is defined as follows:*

$$\Pr(V_1, \dots, V_n) = \prod_{i=1}^n \Pr(V_i | \Pi_{V_i}). \tag{1.1}$$

Proof sketch: Proof proceeds by induction on the number of nodes in the network. Since the network is a DAG, a node V with no children must exist. The inductive hypothesis applies to

the network with that node removed: The node V is conditionally independent of the rest of the network given the parents. That observation combined with the inductive hypothesis leads to the theorem above.

The notion of conditional probability is used in connection with belief propagation (probability distribution over a node given evidence so far) and belief specification (probability distribution over a node given its parents). Notice that the latter is a special case of the former, and that the former is closely related to d-separation. The significance of d-separation is as follows. If two variables U and V are d-separated, then a change in belief for U has no impact on the belief in V and vice versa. The concept of d-separation can be illustrated using the family out example (adapted from [Charniak, 1991]):

[C]onsider the random variables Family-out and Hear-bark. Are these variables independent? Intuitively not, because if my family leaves home, then the dog is more likely to be out, and thus, I am more likely to hear it. However, what if I happen to know that the dog is definitely in (or out of) the house? Is Hear-bark [conditionally] independent of Family-out then? That is, is $\Pr(hb \mid fo, do) = \Pr(hb \mid do)$? The answer now is yes. After all, my hearing her bark was dependent on her being in or out. Once I know whether she is in or out, then where the family is of no concern.

So, once the status of the node Dog-out is known, it d-separates Hear-bark and Family-out. This is an example of a serial evidence node (Dog-out) d-separating two other nodes (Hear-bark and Family-out), and how this implies conditional independence for the corresponding random variables.

We introduce a special notation for conditional independence. Let U , V , and W be random variables. Assume that U is conditionally independent of V given W , in other words

$$\Pr(U \mid W, V) = \Pr(U \mid W) \tag{1.2}$$

or alternatively

$$\Pr(U, V | W) = \Pr(U | W) \Pr(V | W). \quad (1.3)$$

We will abbreviate this as $I(U, V | W)$. Similarly, conditional dependence will be abbreviated as $D(U, V | W)$. The result that generalizes the example above can now be stated.

Theorem 2 (*Verma and Pearl 1988*). *Consider a Bayesian networks with nodes V , and let X, Y , and Z be subsets of V such that X and Y are d-separated by Z in the Bayesian networks. Then $I(X, Y | Z)$. That is, X and Y are conditionally independent given Z .*

The significance of this theorem is that conditional independence follows from d-separation. This fact is exploited in theorems as well as in algorithms. The algorithmic importance follows from the fact that d-separation can be checked in time linear in the number of edges in a causal network [Geiger et al., 1990].

1.3 Bayesian Network Inference

What can a Bayesian networks be used for? The basic Bayesian networks inference algorithm is to compute the probability density over one variable, say H , given the evidence \mathbf{e} encountered so far, i.e. $\Pr(H | \mathbf{e})$. For the ‘family out’ example, if the light is on and we don’t hear the dog bark, this can be expressed as evidence $\mathbf{e} = \{LO = lo, HB = \sim hb\}$. In this case, the posterior probability of $FO = fo$ is 0.5. In other words: $\Pr(FO = fo | LO = lo, HB = hb) = \Pr(fo | lo, hb) = 0.5$. This is an example of diagnostic inference, since we’re observing symptoms (instantiations of Light-on and Hear-bark) and are interested in a diagnosis (distribution over Family-out). Bayesian networks can also be used for prediction, where the probability densities over symptoms are predicted from assumptions about one or several diagnoses, or for mixed inference, where both symptoms and diagnoses are known and the probability densities over other symptoms, diagnoses or intermediate nodes may be inferred.

More formally, two typical diagnostic (or abductive) inference tasks are belief updating and belief revision [Pearl, 1988]. *Belief updating* computes the posterior belief over a hypothesis node

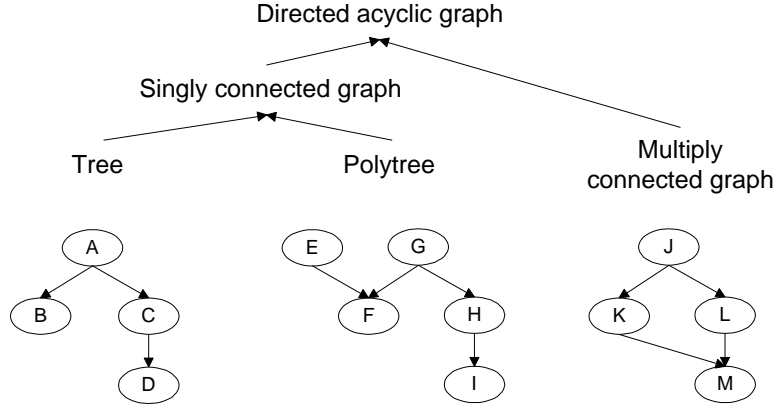


Figure 1.4: Classification and examples of Bayesian networks structure.

H given instantiated evidence nodes E_1, \dots, E_m , more formally $\Pr(H \mid E_1 = e_1, \dots, E_m = e_m)$. Belief updating is also known as probabilistic inference. *Belief revision* computes the posterior belief over a set of hypotheses nodes H_1, \dots, H_k given evidence nodes E_1, \dots, E_m , more formally $\Pr(H_1, \dots, H_k \mid E_1 = e_1, \dots, E_m = e_m)$. Belief revision for the case where all nodes are instantiated is also known as computing an explanation. The task of computing the most probable explanation (MPE), or the k most probable explanations (k MPE) are of particular interest in this dissertation.

A naive algorithm for answering any query concerning a joint pdf is to store a table representing the joint pdf and then marginalize out all the variables not in the query. However, the table is exponentially sized in the number of random variables, and this approach is therefore only feasible for a small number of random variables. Smarter inference algorithms structure the random variables in a Bayesian networks and exploit the Bayesian networks structure.

All Bayesian networks are directed acyclic graphs. Within the class of directed acyclic graphs, we distinguish between singly connected graphs and multiply connected graphs. Singly connected networks are also known as Chow trees [Pearl, 1988]. Within singly connected graphs, we distinguish between trees and polytrees. This Bayesian networks classification, along with example Bayesian networks, is shown in Figure 1.4.

A multiply connected graph is more expressive than a polytree which is more expressive than a tree. Unfortunately, the computational complexity of belief propagation increases with expressiveness. Both trees and polytrees have tree structure in the underlying undirected graph. The

problem with multiply connected graphs is that there are cycles in the underlying undirected graph. Cycles mean that evidence can travel from one node to another through many distinct paths, and this causes problems for belief propagation algorithms. Belief propagation in trees is linear [Pearl, 1988], while belief propagation in multiply connected graphs has been shown to be NP-hard [Cooper, 1990]. So an exponential worst-case time complexity can not be avoided unless $P = NP$.

At the same time, inference in many real-world BNs has proven to be tractable in practice. Algorithms routinely perform inference over multiply connected BNs with up to approximately one thousand nodes. This apparent paradox has led to an intense investigation of BN inference algorithms, which can be classified as exact or approximate. *Exact* algorithms include network propagation [Kim and Pearl, 1983] [Pearl, 1988], conditioning [Pearl, 1988], and clustering [Pearl, 1988] [Lauritzen and Spiegelhalter, 1988] [Jensen et al., 1990b]. *Approximation* algorithms include stochastic simulation [Pearl, 1988], bounded conditioning [Horvitz et al., 1989], and search.

Here is a more detailed description of some inference algorithms:

- Network propagation [Kim and Pearl, 1983] [Pearl, 1988]: Propagates belief by message passing between the nodes in a Bayesian networks, and by performing local computations in the network. The algorithm works for trees and polytrees.
- Cutset conditioning [Pearl, 1988]: Transforms a multiply connected graph into several singly connected graphs, and performs network propagation over each singly connected graph.
- Clustering [Pearl, 1988] [Lauritzen and Spiegelhalter, 1988] [Jensen et al., 1990b]: Transforms a multiply connected graph into an undirected graph that is a tree, and performs belief propagation over the tree. The propagation is similar to that of network propagation.
- Algebraic method [Li and D’Ambrosio, 1994]: Views probabilistic inference as a combinatorial optimization problem, the optimal factoring problem. Probabilistic inference is the problem of finding an optimal factoring given a set of probability distributions. Marginalization is symbolic and query-driven, which is different from the above three approaches where it is numerical and data-driven.

In the following, we present in more detail the network propagation and clustering approaches to

inference in Bayesian networks. The network propagation approach applies to trees and polytrees, while clustering can also be applied to multiply connected graphs.

1.4 Genetic Algorithms

GAs are function optimizers that employ stochastic, population-based search [Holland, 1975] [Goldberg, 1989c]. A population representing candidate solutions is evaluated for fitness using a fitness function. Genetic operators, such as crossover and mutation, then create a new population from the old. The probability of transfer of the genetic material of an individual is a function of its fitness. GAs are attractive because of their robustness, generality, simplicity, and because of their success in applications.

This section presents the simple genetic algorithm, some GA theory, discusses of the central role of building blocks in GAs, and mentions the notions of diversity preservation and approximate fitness evaluation. The reader interested in genetic algorithms is referred to books [Goldberg, 1989c] [Michalewicz, 1992] and articles [Goldberg, 1994] on this topic.

1.4.1 The Simple Genetic Algorithm

A prototypical genetic algorithm, the so-called simple genetic algorithm (SGA) [Goldberg, 1989c], is shown in Figure 1.5. The essence of SGA is really quite simple: The two main data types are individual and population, where a population is an array of individuals. Central in each individual is its chromosomes (or strings). Assuming a binary alphabet $\{0, 1\}$ and a string length $l = 8$, an example chromosome is $C_1 = 10110100$. In the SGA algorithm, the outermost loop is iterated for `maxgen` iterations. For each generation the functions `select`, `crossover`, `mutate`, and `objfunc` are repeatedly invoked. `select(pop)` picks an individual from its input population `pop`. The probability of an individual being picked is proportional to its fitness—this is known as proportional selection. `crossover(chrom1, chrom2, new-chrom1, new-chrom2, Pr(Crossover))` takes chromosomes `chrom1` and `chrom2` as input, creating `new-chrom1` and `new-chrom2` as output by crossing over with probability `Pr(Crossover)`. The SGA uses single-point crossover, where each chromosome is split at just one position. For instance, if $C_1 = 10110100$ is crossed over with $C_2 = 00001111$ at position 4, the chromosomes $C_3 = 00000100$ and $C_4 = 10111111$ result. With probability `Pr(Mutation)`,

Algorithm SGA

```
gen := 1
randomize(oldpop)
repeat
  j := 1
  repeat
    mate1 := select(oldpop)
    mate2 := select(oldpop)
    crossover(oldpop[mate1].chrom, oldpop[mate2].chrom,
              newpop[j].chrom, newpop[j + 1].chrom,
              Pr(Crossover))
    newpop[j].chrom := mutate(newpop[j].chrom, Pr(Mutation))
    newpop[j + 1].chrom := mutate(newpop[j + 1].chrom, Pr(Mutation))
    newpop[j].fitness := objfunc(decode(newpop[j].chrom))
    newpop[j + 1].fitness := objfunc(decode(newpop[j + 1].chrom))
    j := j + 1
  until j > n
  oldpop := newpop
  gen := gen + 1
until gen > maxgen
```

Figure 1.5: The simple genetic algorithm.

`mutate(chrom, Pr(Mutation))` mutates each allele in the chromosome `chrom` and then returns the mutated chromosome. For instance, $C_5 = 00111111$ is computed from C_4 by mutating the first bit. The functions `decode` and `objfunc` make up the interface to the objective function, computing fitness of the two new individuals. The function `decode` maps a genotype—in this case one chromosome—to a phenotype. The eight-bit chromosome may, for example, represent an integer in the range 0–127 or a BN with eight binary nodes. The fitness function `objfunc` takes a value in this phenotypic space and assigns it a fitness value, typically a real number.

Given the setup outlined above, the SGA functions as a black-box function optimizer in the sense that no assumptions are made concerning the objective (or fitness) function. The SGA employs generate and test, a variant of blind search, but is still able to compute a probably approximately correct function optimum in polynomial time, under certain restrictions. Since many computational problems can be cast as optimization problems, the SGA—and more generally GAs—are important algorithms. What makes GAs different from other optimizers is (i) they are population-oriented and (ii) multiple operators (crossover, mutation, selection) are working ‘concurrently’ over the

population.

1.4.2 Genetic Algorithm Theory

A GA explicitly processes strings; implicitly it processes schemata which represent sets of strings.

A schema H is a string over the alphabet $\{0, 1, *\}$. For example, the schema

$$H_1 = 101101 **$$

represents the strings

$$\{10110100, 10110101, 10110110, 10110111\}.$$

A schema's order ω is defined as the number of non- $*$'s in the schema string. For instance, $\omega(H_1) = \omega(101101**) = 6$. A schema's defining length δ is defined as the distance between the non- $*$'s that are farthest apart. H_1 's defining length is $\delta(H_1) = \delta(101101**) = 6 - 1 = 5$.

Let $f(H)$ be the fitness of schema H , \bar{f} the average schema fitness, $m(H, t)$ the number of instances of schema H in the population at time t , and l the length of the chromosome string. The *schema theorem* gives a lower bound on schema processing [Holland, 1975] [Goldberg, 1989c]:

$$m(H, t + 1) \geq m(H, t) \frac{f(H)}{\bar{f}} \left(1 - \Pr(\text{Crossover}) \frac{\delta(H)}{l - 1} - \Pr(\text{Mutation}) \omega(H) \right).$$

Intuitively, the schema theorem says that schemata with above-average fitness, low order, and small defining length grow exponentially. Such highly fit, low-order schemata of small defining length are called *building blocks*. Although the schema theorem is important, one also needs to realize its idealized point of view. Two cases in point are that the values of $f(H)$ and \bar{f} vary over time and are not constant as suggested in the schema theorem, and that it only takes into account schema loss, not schema gain.

1.4.3 Genetic Algorithm Practice

One of the central insights gained over years of GA theory and practice is that building blocks play a key role. The following principles summarize the role of building blocks in GA processing [Goldberg et al., 1992]:

1. Ensure an adequate initial supply of building blocks
2. Ensure that the necessary building blocks are expected to grow
3. Solve problems of bounded building block difficulty
4. Ensure that building block decisions are made well
5. Ensure that building blocks are properly mixed (or exchanged)

All of these points could be expanded upon; however since they are largely intuitively clear, we will only explain point four concerning building block decisions. The problem here is sampling noise. Since each string contains several schemata, there is uncertainty in relating fitness to individual schemata. The solution to the sampling noise problem is population sizing. By making the population large enough, the sampling noise problem is minimized.

So far, the presentation has focused on the SGA. However, the SGA is a representative of a family of algorithms. Other family members are attained by varying the SGA's data structures, selection operator, crossover operator, or mutation operator. Alternative data structures are vectors of real numbers (evolution strategies) and computer programs (genetic programming) [Goldberg, 1989c] [Michalewicz, 1992]. Using alternative data structures can make integration with other algorithms easier, and may therefore be an important part of hybrid GAs. Alternatives to proportionate selection are tournament selection and ranking selection. Selection can also be augmented with elitism, where the k best fit individuals are always retained in the population from one generation to the next. Similarly, there are variants of crossover and mutation.

Two advanced GA issues that are of particular interest to us, are diversity preservation and approximate fitness function evaluation. Both of these aspects are discussed in the next section.

1.5 Contributions of Dissertation

With the notions of Bayesian networks, genetic algorithms, and local search introduced, we present the main contributions of this dissertation.

In each case, we start with presenting the contribution from a Bayesian network point of view. After that, we consider the contribution from a more general point of view, such that it might be appreciated by readers who are primarily interested in, say, genetic algorithms or stochastic local search, rather than Bayesian networks.

1.5.1 Problem Hardness: Deceptive and Satisfiability Bayesian Networks

In order to distinguish the performance of search algorithms, it helps to use hard problems. In this dissertation, we take research on hard problems from other areas and introduce it into the realm of Bayesian networks. In particular, we translate deceptive functions and satisfiability problems into corresponding Bayesian networks, giving deceptive BNs and satisfiability (SAT) BNs respectively. These BNs are part of an experimental paradigm for generating increasingly hard instances. Such hard instances can be used to benchmark new and existing algorithms. For instance, running experiments with the satisfiability Bayesian networks, we have found that existing exact algorithms are extremely inefficient on these hard instances.

1.5.2 Genetic Algorithm Niching: Probabilistic Crowding

Bayesian networks can be highly multi-modal, thus making search algorithms converge to local rather than to global optima. The genetic algorithm technique known as niching addresses the problem of multi-modality. Niching algorithms address multi-modality by maintaining diversity, and in particular they need to maintain useful diversity.

This dissertation introduces a novel niching algorithm, probabilistic crowding. Like its predecessor deterministic crowding, probabilistic crowding is fast, simple, and requires no parameters beyond that of the classical GA. In probabilistic crowding, subpopulations are maintained reliably, and it is possible to analyze and predict how this maintenance takes place. This dissertation also identifies probabilistic crowding as a member of a family of algorithms, which we call integrated tournament algorithms. Integrated tournament algorithms also include deterministic crowding,

restricted tournament selection, parallel recombinative simulated annealing, the Metropolis algorithm, and simulated annealing.

1.5.3 Local Search: Stochastic Greedy Search

We have developed a stochastic local search algorithm augmented with stochastic initialization algorithms. The algorithm, *SGS*, performs several orders of magnitude better than existing exact algorithms on certain hard networks constructed as described above. *SGS* is also comparable in performance to exact algorithms on a benchmark of Bayesian networks developed in applications. We attribute this strength to its dynamic programming and forward simulation based stochastic initialization algorithm.

1.5.4 Problem Approximation: Abstraction in Bayesian Networks

Approximate fitness function evaluation is traditionally concerned with saving computation time; we also use it to decrease the search space size. This part of the dissertation focuses on hierarchical abstraction for computing a most probable explanation in Bayesian networks. The ability to speed up Bayesian network inference using abstraction is an important goal, since speed of computation is a serious problem in large Bayesian networks. We make a connection between abstraction and noise to introduce quality criteria for abstraction. An experimental study is described that applies three abstraction algorithms to two large real-world Bayesian networks in the domains of barley crop yields and electromyography, and quantifies the utility of abstraction.

1.6 Reader's Guide

The rest of this dissertation is organized as follows. Chapter 2 presents previous research on BN inference, with emphasis on inexact methods like genetic algorithms and local search, as well as exact methods related to Hugin. The probabilistic crowding genetic algorithm is presented in Chapter 3. Probabilistic crowding is a niching genetic algorithm based on deterministic crowding. Chapter 4 presents deceptive Bayesian networks. Chapter 5 describes stochastic greedy search, which combines hill-climbing and random steps. Chapter 6 presents experimental results of running stochastic greedy search and Hugin on Bayesian networks constructed from satisfiability instances.

Chapter 7 focuses on model approximation or fitness function approximation. In particular, we consider BN abstraction and refinement, and argue that BN abstraction can be considered as noise. Chapter 8 concludes and outlines directions for future research.

Chapter 2

Bayesian Networks and Inference Algorithms

This chapter presents Bayesian networks as well as on exact and inexact inference algorithms, and also serves to introduce the terminology and notation used elsewhere in this dissertation. Previous research on using genetic algorithms and local search for Bayesian network inference is reviewed. We do not attempt to be complete; the survey is limited to research that covers both Bayesian networks and inference algorithms.

This chapter is organized as follows. Section 2.1 presents definitions. Section 2.2 presents exact inference algorithms, while Section 2.3 discussed inexact algorithms. Section 2.4 summarizes and evaluates the reviewed research, and identifies fruitful research directions.

2.1 Bayesian Network Preliminaries

A Bayesian network (BN) represents a multi-variate probability distribution as a directed acyclic graph (DAG).

Definition 3 *A BN node V is a random variable associated with a state space $\Omega_V = \{v_1, \dots, v_k\}$.*

In the following we will not distinguish between nodes and random variables. For simplicity, but without loss of generality, we often use binary nodes in the following, so a node X has $\Omega_X = \{0, 1\}$.

Definition 4 *A Bayesian network is a tuple $(\mathbf{V}, \mathbf{E}, \mathbf{Pr})$, where (\mathbf{V}, \mathbf{E}) is a directed acyclic graph with nodes $\mathbf{V} = \{V_1, \dots, V_n\}$ and edges $\mathbf{E} = \{V_1, \dots, V_m\}$, and \mathbf{Pr} is a set of conditional probability tables (CPTs). Π_V gives the parents of V , Ψ_V the children of V , and π_V gives an instantiation*

of parents of V . For each node $V \in \mathbf{V}$ there is one CPT, which defines a conditional probability distribution $\Pr(V \mid \Pi_V)$.

Consider a Bayesian network over the set of nodes \mathbf{V} . Then the joint distribution $\Pr(\mathbf{v})$ is:

$$\Pr(\mathbf{v}) = \Pr(v_1, \dots, v_n) = \prod_{i=1}^n \Pr(v_i \mid \pi_{V_i}), \quad (2.1)$$

where $\Pi_{V_i} \subset \{V_{i+1}, \dots, V_n\}$.

Sometime a BN is given *evidence* in terms of setting or clamping some variables to known states. These nodes are called *evidence variables*.

Definition 5 *An explanation \mathbf{x} is a complete assignment $\{X_1 = x_1, \dots, X_n = x_n\}$ that is consistent with evidence. Computing a most probable explanation (MPE) in a BN is finding an explanation such that no other explanation has higher probability.*

It has been shown that exact MPE computation is NP-hard [Shimony, 1994]. Recently, NP-hardness for approximating an MPE to within a constant ratio-bound has also been proven to be NP-hard.

2.2 Exact Inference Methods

One needs to make a distinction between the inference tasks of belief updating (computing marginal distributions) and belief revision (computing an MPE). However, it is often the case that a belief updating algorithm, possibly with small modifications, can be used for belief revision. For example, this is the case for Hugin, which was introduced as a belief updating algorithm [Lauritzen and Spiegelhalter, 1988], but was later extended to encompass belief revision [Dawid, 1992]. A more extreme example of the close relationship between the two algorithmic problems comes from information theory, where a variant of Pearl’s belief propagation algorithm (a belief updating algorithm) is used to compute an MPE [Luby et al., 1998]. This close relationship between belief revision and belief updating needs to be borne in mind when considering algorithms in the following.

2.2.1 Inference by Network Propagation

This section describes belief propagation in Bayesian networks that are singly connected, using the network propagation or the message-passing algorithm [Kim and Pearl, 1983] [Pearl, 1988]. The essence of this algorithm, when applied to trees, is that each node passes a message to its parent and a message to each of its children. In addition, each node has a π -value and a λ -value associated with it, and the posterior probability of a node can be computed from these two values and the messages from neighboring nodes. The algorithms also works for polytrees.

This section gives an introduction to the network propagation approach; it is based on the books of Pearl [Pearl, 1988] and Neapolitan [Neapolitan, 1990].

Preliminaries

The network propagation algorithm is based on message passing. The belief network fragment in Figure 2.1 shows the messages that are passed to and from a node X . Similar messages are passed between all nodes in the Bayesian networks. The scheme is based on the following observation. The probability distribution over X depends on two distinct sets of evidence: Evidence from the subtree rooted at X , and evidence from other parts of the tree. Evidence from other parts of the tree is summarized by U , since U d-separates X from those parts. Similar arguments can be used for other nodes in the tree. Hence, belief updating can proceed by means of local computations (within a node) and message passing (between nodes). In the following, we discuss how the message passing scheme is used for probability propagation in trees.

Messages to a Node

We first consider how the probability distribution over a node X can be computed from incoming messages to X . More formally, let:

- E_X^- be evidence nodes in the tree rooted at X .
- E_X^+ be evidence nodes contained in the rest of the network.

The belief over X , $\text{BEL}(X)$, based on the evidence nodes $E = E_X^- \cup E_X^+$ can be computed using Bayes' rule with E as background evidence:

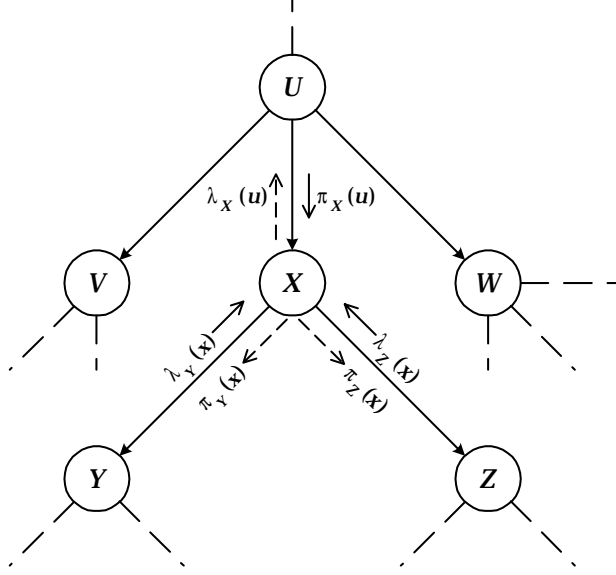


Figure 2.1: Bayesian networks inference by message passing or network propagation (from [Pearl, 1988]).

$$\begin{aligned}
 \text{BEL}(X) &= \Pr(X | E) \\
 &= \Pr(X | E_X^-, E_X^+) \\
 &= \frac{\Pr(E_X^- | X, E_X^+) \Pr(X | E_X^+)}{\Pr(E_X^- | E_X^+)}. \\
 &= \frac{\Pr(E_X^- | X) \Pr(X | E_X^+)}{\Pr(E_X^- | E_X^+)}.
 \end{aligned}$$

But since X is a linear node, we have $I(E_X^-, E_X^+ | X)$ and therefore:

$$\text{BEL}(X) = \frac{\Pr(E_X^- | X) \Pr(X | E_X^+)}{\Pr(E_X^- | E_X^+)}.$$

We now introduce the following definitions:

$$\begin{aligned}
 \lambda(X) &= \Pr(E_X^- | X) \\
 \pi(X) &= \Pr(X | E_X^+).
 \end{aligned}$$

The value $\lambda(X)$ expresses the retrospective or diagnostic support to X , while the value $\pi(X)$ expresses the predictive or causal support to X . When the definitions for $\lambda(X)$ and $\pi(X)$ are introduced into $\text{BEL}(X)$, we get the following equation:

$$\text{BEL}(X) = \alpha \lambda(X) \pi(X), \quad (2.2)$$

where $\alpha = \Pr(E_X^- | E_X^+)^{-1}$ is a normalizing constant.

Using Equation 2.2, $\lambda(X)$ and $\pi(X)$ need to be computed in order to find $\text{BEL}(X)$. Consider $\lambda(X)$ first. The computation of $\lambda(X)$ depends on whether or not X is instantiated or not, so there are two cases.

Case (i): X is not instantiated. Remember that Y and Z are the children of X . Since X is not instantiated, evidence nodes are $E_X^- = E_Y^- \cup E_Z^-$. By d-separation, we have $I(E_X^-, E_X^+ | X)$ and get $\Pr(E_Y^-, E_Z^- | X) = \Pr(E_Y^- | X) \Pr(E_Z^- | X)$ by Equation 1.3. We introduce definitions for $\lambda_Y(X)$ and $\lambda_Z(X)$ as follows:

$$\begin{aligned} \lambda(X) &= \Pr(E_X^- | X) & (2.3) \\ &= \Pr(E_Y^-, E_Z^- | X) \\ &= \Pr(E_Y^- | X) \Pr(E_Z^- | X) \\ &= \lambda_Y(X) \lambda_Z(X). \end{aligned}$$

The λ -message $\lambda_Y(X)$ can be read as ‘message from Y to X ’ (so λ is pronounced ‘message from’). We will cover λ -message computation later, for now just note that $\lambda(X)$ can be computed given the λ -messages.

Case (ii): X is instantiated, so we have $X = x_i$. In this case, a dummy node D is added as a child to X , sending the following messages:

$$\lambda_D(X) = \begin{cases} 1 & \text{if } X = x_i \\ 0 & \text{if } X \neq x_i. \end{cases}$$

In this case we get

$$\lambda(X) = \lambda_D(X)\lambda_Y(X)\lambda_Z(X), \quad (2.4)$$

and the λ -message from D cancels out the effect of the λ -messages from Y and Z . This concludes the presentation of how to compute $\lambda(X)$ in Equation 2.2.

Now, consider the $\pi(X)$ part of Equation 2.2. The value $\pi(X)$ can be computed as follows:

$$\begin{aligned} \pi(X) &= \Pr(X | E_X^+) & (2.5) \\ &= \sum_U \Pr(X | E_X^+, U) \Pr(U | E_X^+) \\ &= \sum_U \Pr(X | U) \Pr(U | E_X^+) \\ &= \Pr(X | U) \bullet \pi_X(U). \end{aligned}$$

Here, $\Pr(X | U)$ is the conditional probability table associated with the edge from U to X ; \bullet denotes dot product. The message $\pi_X(U)$ can be read as ‘message to X from U ’ (so π is pronounced ‘message to’). The definition $\pi_X(U) = \Pr(U | E_X^+)$ is introduced in the last line above.

The above equations give us a way to compute $\text{BEL}(X)$ from the messages to X and the conditional probability table $\Pr(X | U)$. This is summarized in the following equation:

$$\begin{aligned} \text{BEL}(X) &= \alpha\lambda(X)\pi(X) & (2.6) \\ &= \alpha\lambda_Y(X)\lambda_Z(X)\Pr(X | U) \bullet \pi_X(U). \end{aligned}$$

Messages from a Node

We now consider how the node X can compute the messages it needs to send to its neighboring nodes. First, we consider the diagnostic support message $\lambda_X(U) = \Pr(E_X^- | U)$. This is the message from X to its parent node U . There are two cases to consider.

Case (i): X is not instantiated. We condition on X and utilize the fact that $I(E_X^-, U | X)$ as follows:

$$\begin{aligned}
\lambda_X(U) &= \sum_X \Pr(E_X^- | U, X) \Pr(X | U) \\
&= \sum_X \Pr(E_X^- | X) \Pr(X | U) \\
&= \sum_X \lambda(X) \Pr(X | U) \\
&= \Pr(X | U) \bullet \lambda(X).
\end{aligned} \tag{2.7}$$

Here, $\lambda(X)$ is the product of the messages received from the children of X , as discussed earlier. The messages $\lambda_Y(X)$ and $\lambda_Z(X)$, which we assumed were sent to X earlier, can be computed similar to the message $\lambda_X(U)$.

Case (ii): X is instantiated, so we have $X = x_i$. In this case, row i of the conditional probability table $\Pr(X | U)$ makes up $\lambda_X(U)$; we use the following notation:

$$\lambda_X(U) = \Pr(X = x_i | U). \tag{2.8}$$

Second, consider the causal messages (π -messages) that X must send to its children. For example, consider the π -message from X to Y . The causal support to Y from X is determined by E_X^+ and E_Z^- : $\pi_Y(X) = \Pr(X | E_Y^+) = \Pr(X | E_X^+, E_Z^-)$. We use Bayes' rule, treat E_X^+ as background knowledge, and use $I(E_X^+, E_Z^- | X)$ as follows:

$$\begin{aligned}
\pi_Y(X) &= \Pr(X | E_X^+, E_Z^-) \\
&= \frac{\Pr(E_Z^- | X, E_X^+) \Pr(X | E_X^+)}{\Pr(E_Z^- | E_X^+)} \\
&= \frac{\Pr(E_Z^- | X) \Pr(X | E_X^+)}{\Pr(E_Z^- | E_X^+)} \\
&= \alpha \lambda_Z(X) \pi(X).
\end{aligned} \tag{2.9}$$

Here, $\alpha = \Pr(E_Z^- | E_X^+)^{-1}$, $\lambda_Z(X) = \Pr(E_Z^- | X)$, and $\pi(X) = \Pr(X | E_X^+)$. In other words, the equalities above show that the message $\pi_Y(X)$ can be computed by the node X . The message $\pi_X(U)$, which we assumed was sent to X earlier, can be calculated in a way similar to the message

$\pi_Y(X)$.

We can now summarize the role of messages sent to and from a node X as well as the values $\lambda(X)$ and $\pi(X)$ associated with it. A node's λ -value can be calculated from the messages it receives from its children, while the π -value is calculated from the message from its parent. Together, the messages and values are used to update the belief $\text{BEL}(X)$ in the node X , using Equation 2.6. The node X similarly sends messages that allow other nodes to compute their beliefs.

Special Nodes

Some types of nodes need special attention:

- Anticipatory node A : This is an uninstantiated leaf node A . $\text{BEL}(A)$ should be $\pi(A)$, so we let $\lambda(A) = (1, \dots, 1)$.
- Evidence node E : If the i -th value is observed true, let

$$\lambda(E) = (0, \dots, 0, 1, 0, \dots, 0),$$

where there is a one at the i -th position of the vector.

- Dummy node D : This is a node representing virtual or judgemental evidence. Post the message $\lambda_D(X)$, where $\lambda_D(X) = \beta \Pr(O | X)$, where β is a constant and O are the observations made. The constant β just means that $\lambda_D(X)$ need not be a vector that sums to one; it is the relative weight of the vector values that counts.
- Root node R : Set $\pi(R)$ equal to the prior probability of the root variable R .

Example

Belief propagation is best understood by example. The following example is taken from Neapolitan's book [Neapolitan, 1990]; we will refer to it as the 'cheating spouse' example:

A person is suspicious of his or her spouse cheating. Evidence considered by the person is whether or not there are strange phone calls to the spouse, whether the spouse dines with another person, and whether there are reports of the spouse dining with a stranger.

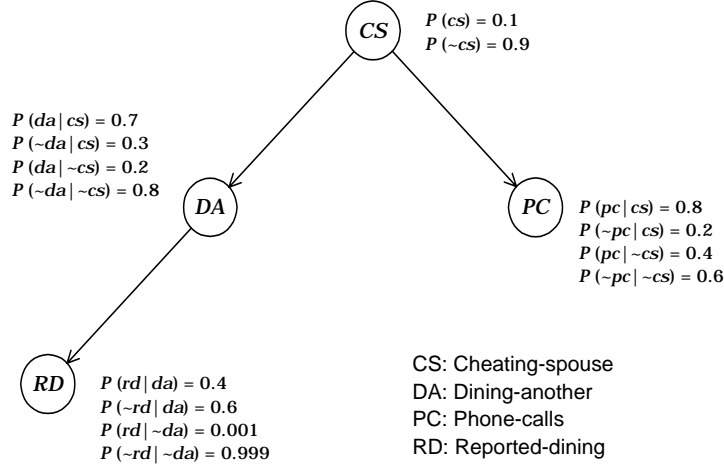


Figure 2.2: Example Bayesian networks; the ‘cheating spouse’ example.

The situation can be formalized as a Bayesian network as shown in Figure 2.2. (This and the following three figures are adapted from [Neapolitan, 1990].) Both the Bayesian network’s structure and conditional probabilities are shown in Figure 2.2. In Figure 2.3, the λ - and π -values and messages for the ‘cheating spouse’ example are shown. The a priori probabilities of each random variable CS , DA , PC , and RD are shown in Figure 2.4. This is the probability distribution over the nodes in the Bayesian network after the prior probabilities over the root node have been propagated, but before any evidence has been entered into the network. We omit the calculations leading to these beliefs.

Now suppose the person receives the information that the spouse dines with another, i.e. $e = \{DA = da\}$. This results in DA ’s λ -value being changed to $(1, 0)$, since DA is an evidence node:

$$\lambda(DA) = (1, 0).$$

We could have added a dummy node D (see Equation 2.4), however the present approach gives the same result.

Now, consider the belief in or posterior probability of the remaining random variables in the network, CS , RD , and PC . That is, we want to compute $\Pr(CS | da)$, $\Pr(RD | da)$, and $\Pr(PC | da)$. The issue is how the messages from DA affect the λ - and π -values of other nodes, directly and

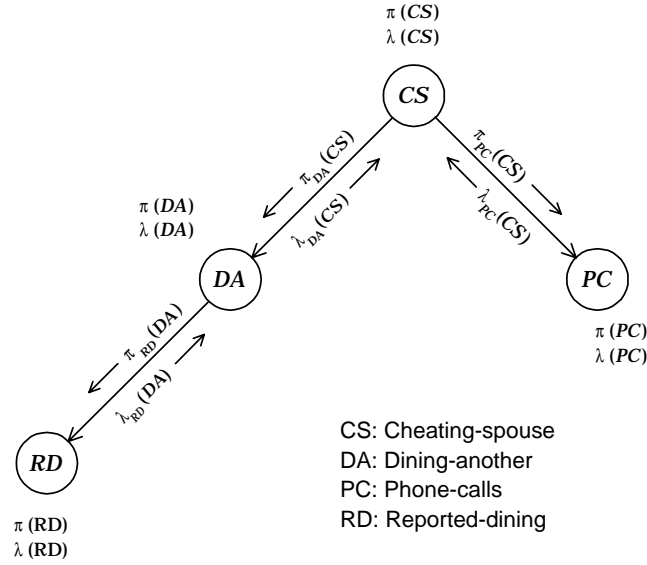


Figure 2.3: Messages and node values in the 'cheating spouse' example.

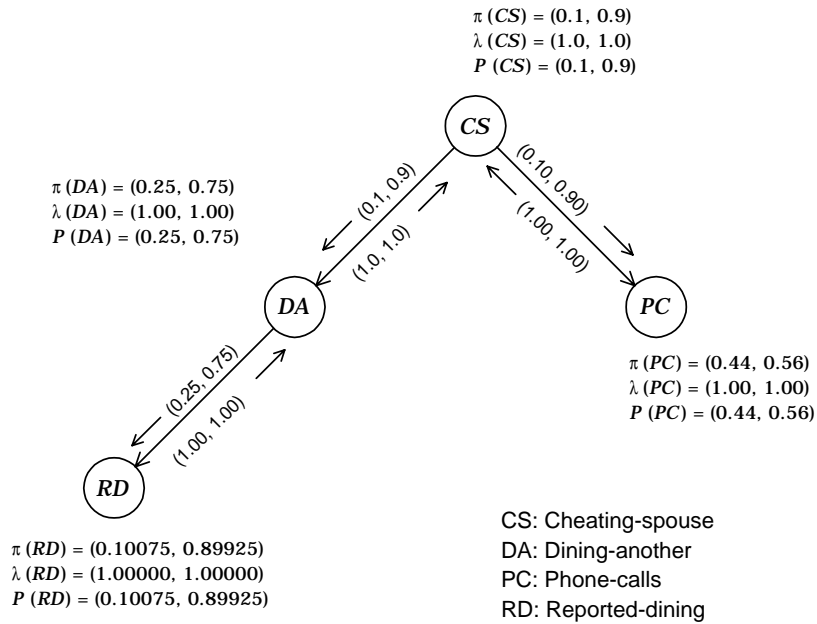


Figure 2.4: Prior probability distribution in the 'cheating spouse' example.

indirectly. We consider one node, namely CS .

Using Equation 2.8, we can compute:

$$\begin{aligned}\lambda_{DA}(CS) &= \Pr(DA = da \mid CS) \\ &= (0.7, 0.2).\end{aligned}$$

The message $\lambda_{PC}(CS) = (1.0, 1.0)$ reflects the fact that there is no evidence in the right branch of the tree, and we use Equation 2.3 to get:

$$\begin{aligned}\lambda(CS) &= \lambda_{DA}(CS)\lambda_{PC}(CS) \\ &= (0.7, 0.2)(1.0, 1.0).\end{aligned}$$

The updated belief over CS can now be computed using Equation 2.6:

$$\begin{aligned}\text{BEL}(CS) &= \alpha\lambda(CS)\pi(CS) \\ &= \alpha(0.7, 0.2)(0.1, 0.9) \\ &= \alpha(0.07, 0.18) \\ &= (0.28, 0.72).\end{aligned}$$

A full overview of values, messages, and posterior probabilities (or beliefs) for the entire network after DA is instantiated is provided in Figure 2.5.

Tree Propagation Algorithm

Consider a node X with m children Y_1, \dots, Y_m and one parent U . The belief distribution of X can be computed given:

- causal support: $\pi_X(U) = \Pr(U \mid E_X^+)$
- diagnostic support from the j -th child: $\lambda_{Y_j}(X) = \Pr(E_{Y_j}^- \mid X)$
- the conditional probability table: $\Pr(X \mid U)$

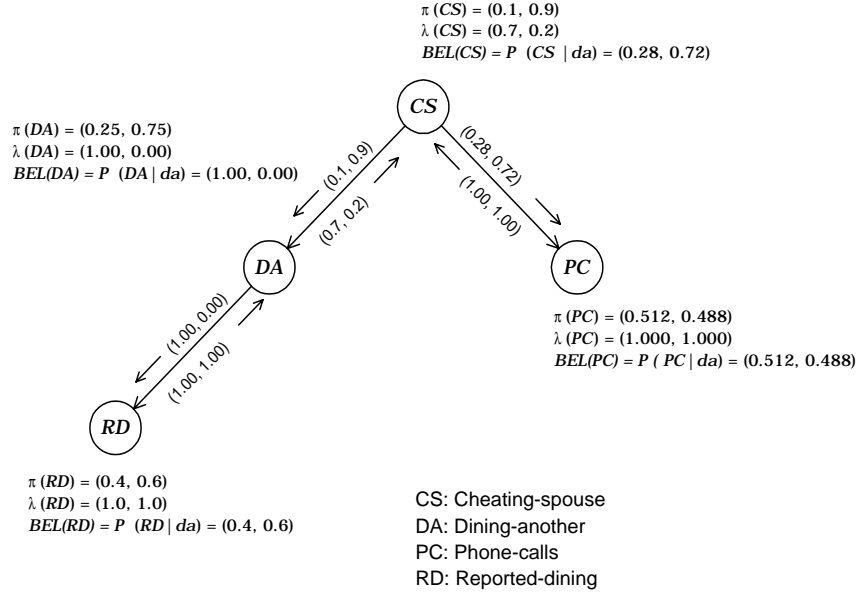


Figure 2.5: Posterior probability distribution in the ‘cheating spouse’ example.

Local belief propagation is achieved using the following three steps:

1. BeliefUpdating. Updated belief is:

$$BEL(X) = \alpha \lambda(X) \pi(X).$$

2. BottomUpPropagation. The message to parent U is:

$$\lambda_X(U) = \sum_X \lambda(X) \Pr(X | U).$$

3. TopDownPropagation. The message to the j -th child Y_j is:

$$\pi_{Y_j}(X) = \alpha \pi(X) \prod_{k \neq j} \lambda_{Y_k}(X).$$

The probabilistic meaning of the nodes is maintained using the above propagation scheme.

We summarize the network propagation approach to belief propagation by means of Figure 2.6. (The figure is adapted from [Pearl, 1988].) The figure shows how the Bayesian network is initially

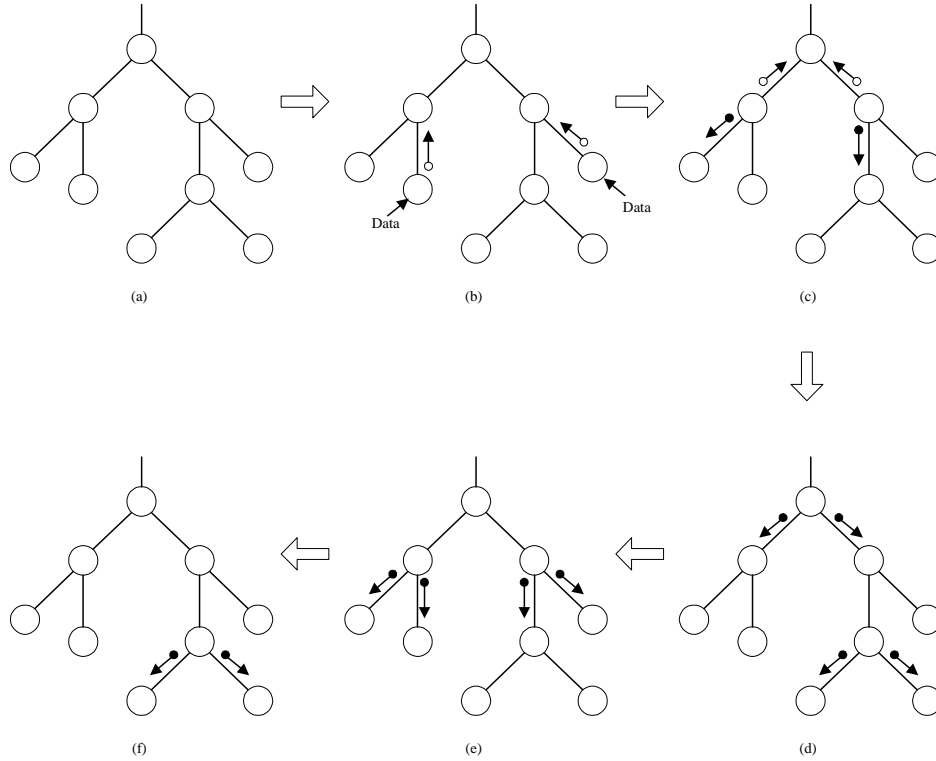


Figure 2.6: Propagating the effect of new evidence in a Bayesian networks.

in equilibrium (a); after which two new pieces of evidence arrives (b). The evidence arrives at the two leaf nodes marked ‘Data’. This leads to messages to parents (white messages) (b); the parents update their parents and other children (black messages) (c). Belief updating continues until all nodes have received the appropriate messages (d)-(f).

The network propagation algorithm has three advantageous features. First, its time complexity is linear in the diameter of the network. Second, the computations required can be performed locally, and they require little local space. Third, the computations are independent of the global control scheme in the sense that different control schemes can lead the network to a stable representation of the joint pdf.

Polytree Propagation Algorithm

The network propagation algorithm can be generalized to polytrees, i.e. singly connected networks where some nodes have more than one parent, but where there is only one path between any

two nodes in the Bayesian network. The propagation scheme is essentially the same as for trees. Consider a node X with m children Y_1, \dots, Y_m and n parents U_1, \dots, U_n . The belief distribution of X can be computed given:

- causal support from the i -th parent: $\pi_X(U_i) = \Pr(U_i | E_{U_i X}^+)$
- diagnostic support from the j -th child: $\lambda_{Y_j}(X) = \Pr(E_{XY_j}^- | X)$
- the conditional probability table: $\Pr(X | U_1, \dots, U_n)$

Local belief propagation is achieved using similar steps as used for trees, i.e. steps similar to BeliefUpdating, BottomUpPropagation, and TopDownPropagation. The propagation formulas are changed to take a node's multiple parents into account.

For example, BeliefUpdating is still based on the formula $\text{BEL}(X) = \alpha \lambda(X) \pi(X)$. Here, $\lambda(X)$ is defined as before, but the definition of $\pi(X)$ is now

$$\pi(X) = \sum_{U_1, \dots, U_n} \Pr(X | U_1, \dots, U_n) \prod_i \pi_X(U_i) \quad (2.10)$$

rather than

$$\pi(X) = \sum_U \Pr(X | U) \pi_X(U) \quad (2.11)$$

as it was for trees.

The advantageous features mentioned for the tree propagation algorithm carry almost entirely over to the polytree propagation algorithm. There is a limitation from the point of view of computational complexity. The summation in Equation 2.10 ranges over all combinations of parent variables. This is exponential, which is problematic if there are more than 4–5 parents. Canonical models of multi-causal interactions, such as the noisy OR, are a way to counteract this problem [Pearl, 1988].

2.2.2 Inference by Clustering

Clustering is one approach to making inference in multiply connected Bayesian networks possible [Pearl, 1988]. The Hugin approach is based on clustering. Hugin propagation operates in two phases: a compilation or clustering phase and a run-time or propagation phase. In the compilation phase, a Bayesian network is transformed into belief universes organized as a tree, a junction tree. A belief universe consists of a set of nodes and a belief table, i.e. a non-normalized joint probability table. In the run-time phase, evidence is propagated between the belief universes in the junction tree.

Hugin propagation was originally developed by Lauritzen and Spiegelhalter [Lauritzen and Spiegelhalter, 1988]; later it was refined by Jensen et al. [Jensen et al., 1990b] [Andersen et al., 1989]. This section is partly based on those sources, but in particular Jensen’s recent book describing the Hugin approach [Jensen, 1996].

Junction tree

A junction tree is a graph where the nodes are cliques and edges have a separator. There are belief tables for both the nodes and the separators: belief tables and separator tables respectively. A junction tree exhibits the *junction tree property*: For any two nodes U and V in the tree, all nodes between them contain $U \cap V$. A junction tree represents a Bayesian network over the variables \mathbf{X} if:

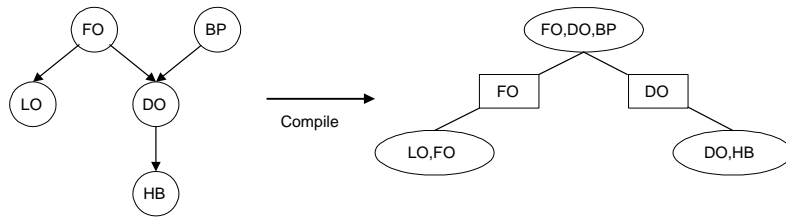
- there is a node containing $A \cup \Pi_A$ for all nodes A in the Bayesian network.
- $\Pr(\mathbf{X})$ is the product of the node belief tables divided by separator belief tables

Examples of Bayesian networks and their corresponding junction trees are shown in Figure 2.7.

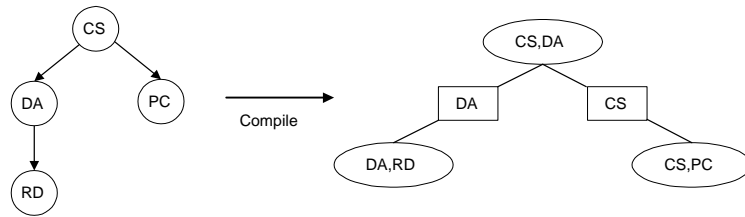
For each cluster C and neighboring separator set S the following holds:

$$\sum_{C \setminus S} B(C) = B(S), \quad (2.12)$$

where B denotes a belief table, a probability table that does not necessarily sum to one. (Belief tables are discussed in detail below.) When Equation 2.12 holds for a neighboring clique and



1. Family out example



2. Cheating spouse example

Figure 2.7: Two examples of Bayesian networks and junction trees representing them. The Bayesian networks is shown to the left, the junction tree to the right.

separator pair we say that they are consistent. When the equation holds for all such pairs we say that the junction tree is locally consistent.

From Bayesian Network to Junction Tree

A junction tree (or a set of belief universes) can be constructed from a Bayesian network in the following way:

- Construct a moral graph: Make an undirected copy of the Bayesian network. Add edges between nodes $A \cup \Pi_A$, where A ranges over all nodes with parents in the Bayesian network.
- Triangulate the moral graph: Add edges to the moral graph such that it becomes triangulated—i.e. no chordless cycle of length greater than three exists.
- Create a junction graph (or system of belief universes): For each clique in the triangulated moral graph, construct a node in the junction graph. Between any two junction tree nodes containing the same nodes in the underlying clique, construct a separator with those nodes. Add edges between the separator and the two nodes.

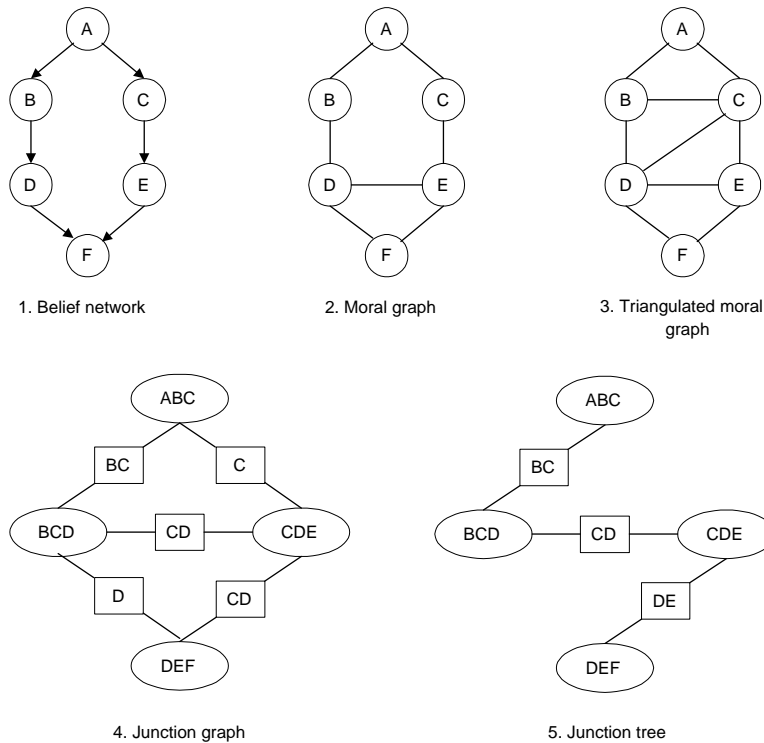


Figure 2.8: The six steps involved in creating a junction tree from a Bayesian networks.

- Create a junction tree: Create a junction tree from the junction graph by deleting edges and separators. The junction tree property must be satisfied by the resulting junction tree.

An example of following the steps above is shown in Figure 2.8.

The most crucial step in the process of creating a junction tree from a Bayesian network is triangulation. This is because triangulation determines the clique sizes, and the size of a belief table is exponential in the size of the clique which it represents. Triangulation is unfortunately known to be NP-hard, but there are heuristic algorithms that perform triangulation quite well [Jensen, 1996] [Huang and Darwiche, 1996].

Preliminaries

The operations of extension, restriction, multiplication, addition, marginalization, and division are defined for belief functions or belief tables [Jensen et al., 1990a]. Here we consider how multiplication, division, and marginalization need to be defined for belief tables. Multiplication is performed

$B(X, Y)$		
	y_1	y_2
x_1	a_{11}	a_{12}
x_2	a_{21}	a_{22}

$B'(X, Y)$		
	y_1	y_2
x_1	b_{11}	b_{12}
x_2	b_{21}	b_{22}

$B''(X, Y)$		
	y_1	y_2
x_1	$a_{11}b_{11}$	$a_{12}b_{12}$
x_2	$a_{21}b_{21}$	$a_{22}b_{22}$

Table 2.1: Multiplication of belief tables.

element-wise, like matrix multiplication with a scalar but unlike multiplication of two matrices. Suppose we have two belief tables $B(X, Y)$ and $B'(X, Y)$. Their product $B''(X, Y) = B(X, Y)B'(X, Y)$ is defined as shown in Table 2.1. An alternative way to write this product is:

$$\begin{aligned}
B''(X, Y) &= B(X, Y)B'(X, Y) \\
&= \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \\
&= \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{pmatrix}.
\end{aligned}$$

We use the convention that the first variable in a belief function, for instance X in $B''(X, Y)$, ranges over rows in the table while the second variable, here Y , ranges over columns.

Belief table multiplication is defined similarly when not all variables are overlapping. This corresponds to multiplication where the dimensions of the tables differ, as in:

$$\begin{aligned}
B''(X, Y) &= B(X, Y)B'(X) \\
&= \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} \\ b_{21} \end{pmatrix} \\
&= \begin{pmatrix} a_{11}b_{11} & a_{12}b_{11} \\ a_{21}b_{21} & a_{22}b_{21} \end{pmatrix}.
\end{aligned}$$

For example, $B''(x_1, y_1) = B(x_1, y_1)B'(x_1) = a_{11}b_{11}$, and $B''(x_1, y_2) = B(x_1, y_2)B'(x_1) = a_{12}b_{11}$.

Division is also performed element-wise, and is defined similar to multiplication. Marginalization is defined analogous to marginalization of probability tables.

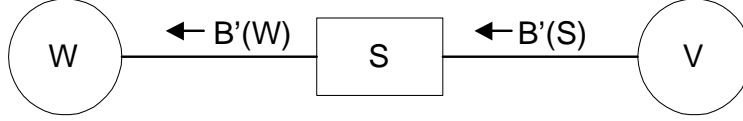


Figure 2.9: Absorption in a junction tree.

Initialization

Once a junction tree is compiled from a Bayesian network, it needs to be initialized. Initialization progresses as follows. First, for each cluster C in the junction tree let $B(C) = 1$. Second, for each variable V_k in the Bayesian network, find a cluster C that contains $V_k \cup \Pi_{V_k}$, and let $B'(C) = B(C) \Pr(V_k \mid \Pi_{V_k})$. After this, the following equation holds:

$$\frac{\prod_{i=1}^n B(C_i)}{\prod_{j=1}^{n-1} B(S_j)} = \frac{\prod_{k=1}^q \Pr(V_k \mid \Pi_{V_k})}{1} = \Pr(\mathbf{U}),$$

where C_i denotes cluster number i , S_j denotes separator number j , q is the number of nodes in the BN, and \mathbf{U} is the set of nodes in the Bayesian network.

Evidence and Absorption

Evidence is treated as follows in the Hugin model. Suppose there is a node X such that $\Pr(X) = (x_1, \dots, x_n)$. Evidence e that X can only be in x_i or x_j gives the following result: $\Pr(X, e) = (0, \dots, 0, x_i, 0, \dots, 0, x_j, 0, \dots, 0)$. This can be regarded as multiplication of the belief table for X with the evidence vector e , where $e = (0, \dots, 0, 1, 0, \dots, 0, 1, 0, \dots, 0)$. In other words, in the evidence vector there is a one where there is evidence, zero elsewhere.

The conditional probability $\Pr(X \mid e)$ can be calculated using the equation $\Pr(X \mid e) = \Pr(X, e) / \Pr(e)$.

Consider the junction tree in Figure 2.9. The nodes W and V are neighbors in the junction tree, S is their separator, and $B(W)$, $B(V)$ and $B(S)$ the respective belief tables. The figure shows how the effect of a change in V 's belief table $B'(V)$ is propagated to W 's $B'(W)$ via S by means of the belief table $B'(S)$. More technically, we say that the node W absorbs from V . Absorption amounts to these actions:

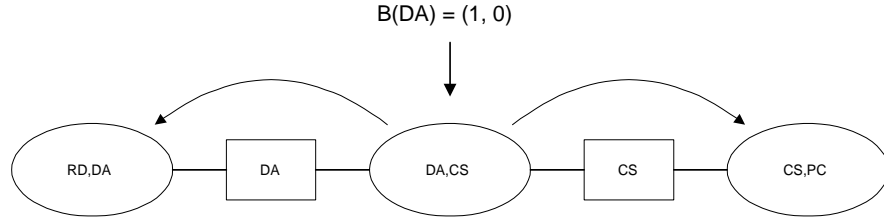


Figure 2.10: Propagation in junction tree of the ‘cheating spouse’ example based on evidence $DA = (1,0)$.

- compute $B'(S) = \sum_{V \setminus S} B(V)$ at V ,
- send $B'(S)$ from V to S , and
- send $B'(W) = B(W) \frac{B'(S)}{B(S)}$ from S to W .

Absorption is the essence of what takes place during belief propagation in the Hugin architecture.

Belief Propagation: Example

In the following we illustrate by example how Hugin propagation works. Consider the cheating spouse example introduced earlier. The result of compiling the cheating spouse Bayesian network into a junction tree is shown in Figure 2.7. Suppose initial propagation has taken place and that the evidence $e = \{DA = da\}$. That is, the evidence vector is $B(DA) = (1,0)$. This gives the following effect:

$$\begin{aligned}
 B'(DA, CS) &= B'(DA, CS)B(DA) \\
 &= \begin{pmatrix} 0.07 & 0.18 \\ 0.03 & 0.72 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\
 &= \begin{pmatrix} 0.07 & 0.18 \\ 0.00 & 0.00 \end{pmatrix}.
 \end{aligned}$$

We marginalize $B'(DA, CS)$ to get $B'(DA)$:

$$B'(DA) = \sum_{CS} B'(DA, CS) = \begin{pmatrix} 0.25 \\ 0.00 \end{pmatrix}.$$

Now, the belief universe $B(RD, DA)$ can be updated:

$$\begin{aligned} B'(RD, DA) &= B(RD, DA) \frac{B'(DA)}{B(DA)} \\ &= \begin{pmatrix} 0.40 & 0.01 \\ 0.60 & 0.99 \end{pmatrix} \frac{\begin{pmatrix} 0.25 & 0.00 \\ 0.25 & 0.75 \end{pmatrix}}{\begin{pmatrix} 0.25 & 0.00 \\ 0.25 & 0.75 \end{pmatrix}} \\ &= \begin{pmatrix} 0.4 & 0.0 \\ 0.6 & 0.0 \end{pmatrix}. \end{aligned}$$

Similar to $B'(DA)$, the belief $B'(CS)$ can be computed by marginalization:

$$B'(CS) = \sum_{DA} B'(DA, CS) = \begin{pmatrix} 0.07 \\ 0.18 \end{pmatrix}.$$

We update the belief universe $B(PC, CS)$ similar to $B(RD, DA)$:

$$\begin{aligned} B'(PC, CS) &= B(PC, CS) \frac{B'(CS)}{B(CS)} \\ &= \begin{pmatrix} 0.08 & 0.36 \\ 0.02 & 0.54 \end{pmatrix} \frac{\begin{pmatrix} 0.07 & 0.18 \\ 0.10 & 0.90 \end{pmatrix}}{\begin{pmatrix} 0.07 & 0.18 \\ 0.10 & 0.90 \end{pmatrix}} \\ &= \begin{pmatrix} 0.056 & 0.072 \\ 0.014 & 0.108 \end{pmatrix}. \end{aligned}$$

The belief table $B'(PC, CS)$ can easily be turned into a probability table $P'(PC, CS)$:

$$\begin{aligned}
P'(PC, CS) &= \frac{1}{0.056 + 0.072 + 0.014 + 0.108} \begin{pmatrix} 0.056 & 0.072 \\ 0.014 & 0.108 \end{pmatrix} \\
&= \begin{pmatrix} 0.224 & 0.288 \\ 0.056 & 0.432 \end{pmatrix}.
\end{aligned}$$

The probability table $P'(PC, CS)$ is marginalized to produce the updated probabilities $P'(CS)$:

$$P'(CS) = \sum_{PC} P'(PC, CS) = (0.28, 0.72).$$

Notice how $P'(CS)$ is the same as the belief $BEL(CS)$ obtained using the network propagation approach in Section 2.2.1.

Belief Propagation: Algorithms

Propagation in the junction tree of belief universes proceeds by the procedures of message passing, DistributeEvidence, CollectEvidence, and Hugin propagation. These are all described below.

Hugin's message passing scheme is essentially that proposed by Kim and Pearl [Kim and Pearl, 1983]: A node U sends a message to a neighbor node when all its other neighbor nodes have sent messages to U . The reader is referred to Section 2.2.1 on network propagation for details on this.

The purpose of Hugin message passing and the other components of the Hugin architecture is summarized in the following two theorems (see [Jensen, 1996, pp. 79–79] for details).

Theorem 6 *Consider a BN representing $\Pr(\mathbf{U})$, and let T be a junction tree corresponding to the BN. After a full round of message passing in T , we have for each node V and separator S that:*

$$B(V) = \sum_{\mathbf{U} \setminus V} \Pr(\mathbf{U}) = \Pr(V) \text{ and } B(S) = \Pr(S).$$

Theorem 7 *Consider a BN representing $\Pr(\mathbf{U})$, and let T be a junction tree corresponding to the BN. Let $e = \{f_1, \dots, f_m\}$ be findings on the variables $\{A_1, \dots, A_m\}$. For each i find a node containing A_i and multiply its table with f_i . Then, after a full round of message passing, we have for each*

node V and separator S that:

$$B(V) = \Pr(V, e) \quad B(S) = \Pr(S, e) \quad \Pr(e) = \sum_V B(V).$$

DistributeEvidence is used for propagating belief *from* a node after it has received evidence. Suppose W has received evidence and has a neighbor V . DistributeEvidence is a recursive procedure that when invoked on V : (i) calibrates V with respect to W , (ii) invokes DistributeEvidence on all neighbors of V except for W . The ‘cheating spouse’ example shown earlier in this section illustrates how DistributeEvidence works. In that example, propagation starts from the node (DA, CS) , and terminates after (RD, DA) and (CS, PC) have been updated.

CollectEvidence is useful for propagating evidence *to* a node after other nodes have received evidence. It works similarly to DistributeEvidence, but with respect to updating a particular node’s posterior probability based on evidence elsewhere in the network.

Hugin propagation is the highest-level algorithm of the Hugin approach. It combines DistributeEvidence and CollectEvidence and works as follows. First, a particular node is picked as the root of the junction tree. Then DistributeEvidence and CollectEvidence are called such that belief is first propagated from the root towards the leafs of the tree, then from the leafs toward the root. This approach is amendable to a recursive implementation. The advantage of Hugin propagation over the message passing scheme is that the junction tree is traversed in a more controlled fashion.

In addition to belief updating, which was described above, it is also possible to do belief revision in the Hugin model. The approach is similar to above, and allows the most probable explanation to be computed [Jensen, 1996, pp. 104–107]. Note that Jensen calls the most probable explanation the configuration of maximal probability.

2.3 Inexact Inference Methods

This section presents and discusses previous research on BN inference using inexact inference methods, including genetic algorithms, stochastic search, and simulated annealing.

2.3.1 Lin et al.

Lin et al. consider abductive diagnosis in medicine, i.e. search for the set of diagnoses (hypotheses) that best explain the evidence [Lin et al., 1990, p. 129]. They see two problems associated with medical abductive diagnosis; uncertainty and search space size. To address these two problems, Lin et al. investigate three randomized search techniques: iterative local search, simulated annealing, and genetic search. Positive results are reported based on an empirical study using the decision-theoretic version of the QMR medical knowledge base, QMR-DT. The algorithms proved to be tractable and converged, in many cases, to the most probable explanation.

Iterative local search (ILS) is a probabilistic hill-climbing variant also known as neighborhood search [Lin et al., 1990, p. 129]. ILS is a combination of next-ascent hill climbing and random-mutation hill climbing as presented by Mitchell [Mitchell, 1996, p. 129]. The limitation of ILS is that it stops on (local) maxima, however since it is iterated it typically finds many maxima, one of which might be global one. Genetic search (GS) is a genetic algorithm derived from Ackley’s iterated GS [Ackley, 1987]. GS is a steady-state GA employing survival of the fittest replacement, where individuals of below-average fitness are replaced at random. Selection is done uniformly at random. Ackley presents both a uniform and an ordered crossover variant of GS. For both variants, one new individual is created.

For their variant of GS, Lin et al. developed a special mutation operator based on a node’s Markov blanket. Given a node H_i , the Markov blanket d-separates the node from the rest of the BN. The Markov blanket is exploited in two ways. First, only the Markov blanket needs to be recalculated after mutation. Second, the Markov blanket is used to determine mutation probability. Assuming, without loss of generality, binary nodes with values 0 and 1, an example mutation probability is given by the Markov-blanket score:

$$\Pr(\text{Mutation of } H_i \text{ from 0 to 1}) = \frac{\Pr(H_i = 1 \mid \mathbf{h}, \mathbf{e})}{\Pr(H_i = 0 \mid \mathbf{h}, \mathbf{e}) + \Pr(H_i = 1 \mid \mathbf{h}, \mathbf{e})}. \quad (2.13)$$

Here, $\Pr(H_i = h_i \mid \mathbf{h}, \mathbf{e})$ denotes the probability that hypothesis H_i has value h_i , other hypotheses \mathbf{h} keep their current values as does the evidence \mathbf{e} . Lin et al. remark that ‘it is difficult to formalize

Experimental BN	N	V	C	ILS	SA	GS
Reduced BN	4070	30	4040	77%	94%	76%
Full BN	4574	534	4040	57%	83%	52%

Table 2.2: Summary of experiments comparing iterative local search (ILS), simulated annealing (SA), and genetic search (GS) for two different BNs, ‘Reduced BN’ and ‘Full BN’. ‘N’, ‘V’, and ‘C’ give the total size of the BN, the number of disease (root) nodes, and the number of feature (leaf) nodes respectively. Note that these numbers are upper bounds since they apply to the BN before pruning, and this was performed before all search algorithms were used. The numbers in the table refer to the percentage of cases in which the best explanation was found.

a theoretical construct for our mutation operator’ [Lin et al., 1990, p. 129], however the intuition is that the probability of mutation is proportional to the context of where mutation takes place. Two different crossover operators were considered in GS, uniform and ordered. They gave approximately the same performance on QMR-DT, which can be explained by the topology of this BN as well as how it was pruned before the experiment.

The QMR-DT BN is a bipartite BN that exploits the causal-independence assumption. One aspect of the QMR-DT is captured in what has been called the independent relevant symptoms model [Castillo et al., 1997]; in addition the noisy-OR assumption is made. Irrelevant nodes were pruned away before GA inference using Shachter’s operations [Shachter, 1988].

Two experiments were performed with QMR-DT: using a reduced BN and using a full BN. Sizes of these networks, before pruning, are given in Table 2.2. Sizes of networks actually used for inference is not given [Lin et al., 1990, p. 129] and is hence unknown. However, it is known that only up to 70 findings is used. For the reduced BN, 30 diseases were selected and an exhaustive search performed to determine the most probable explanation (MPE) in each of 10 different cases used for experimentation. These 10 MPEs made up the gold standard. For the full BN, the gold standard was constructed by taking the best explanation from three sources. First, 500 iterations with ILS; second, 500,000 trials using a variant of the forward sampling algorithm likelihood weighing [Shachter and Peot, 1990] [Fung and Chang, 1990]; and third, the best explanation found by any of the three algorithms in the experimental runs. Experiments with ILS, SA, and GA were performed. The results of the experiments are summarized in Table 2.2. In terms of computational speed, ILS was significantly faster than SA which was significantly faster than GS.

For the reduced BN, ILS is preferred to SA and GS because of its shorter computation time

giving explanations of the same quality. For the full BN, SA is preferred to ILS and GS because of its higher quality explanations. This insight into why SA outperforms ILS for the full BN is noteworthy [Lin et al., 1990]:

We suspect that, for some of the cases, the ILS algorithm has a low probability of ever finding the global maximum. We suspect that there are many local maxima that are close to the global maximum, so that local searches tend to get trapped.

In summary, this research shows that iterative local search, simulated annealing, and genetic search are suitable for computing MPEs; however for difficult cases there is significant room for improvement. Some limitations of the approaches investigated is that the probabilities of the MPEs are not computed, there are no bounds on the probabilities, and the k MPEs are not computed. Also, it is also not clear why genetic search does not perform as well as iterative local search. It might be unfortunate GS parameter settings or that this GA type is not appropriate for BNs. As reflected in the above quote, there is a lack of understanding of what the structure of the search space is, and how this interacts with which search algorithm is best suited to search that space.

2.3.2 Rojas-Guzman and Kramer

Rojas-Guzman and Kramer developed the GALGO system for abductive GA inference in BNs [Rojas-Guzman and Kramer, 1993] [Rojas-Guzman, 1995] [Rojas-Guzman and Kramer, 1996]. The motivation behind GALGO is that approximate inference is attractive for BNs with a high number of variables, a high number of states per variable, and many undirected cycles. Salient features of GALGO are its use of non-binary alphabets, a graph representation, and graph operators. The use of a graph representation instead of a string representation for individuals is based on an informal schema argument: Nodes that are adjacent in a BN might not be adjacent in a string representation of the BN, but can obviously be in a graph. Thus, GA building blocks are less likely to exist in the string representation.

An experiment was performed to investigate the impact of individual representation (string or graph) on GALGO's performance [Rojas-Guzman and Kramer, 1996]. The ALARM BN [Beinlich et al., 1989] was used in the experiment, which is summarized in Table 2.3. Although the time expenditure per generation is smaller for the string representation, the graph representation is

Individual	Runs	Accuracy	Convergence	Run time
String representation	30	83.3%	39.3 generations	62.8 s
Graph representation	35	91.4%	33.0 generations	57.2 s

Table 2.3: Summary of experiment on the impact of string or graph representation of individuals in GALGO.

advantageous both from the point of view of accuracy and run time. However, the difference is not extreme, and there is also the question of how different BNs would behave for different individual representations.

A generation gap model is used, where the worst fit individuals in the population are replaced each generation [Rojas-Guzman and Kramer, 1996, p. 64]. In addition, only a proportion of the population, the breeding population, is allowed to mate. To control the selection pressure of their proportionate selection scheme, three variants of fitness scaling were investigated: no scaling, inverse logarithmic scaling $1/\log^2(x)$, and uniform scaling $1/k$, where k is the size of the breeding population (i.e. the part of the population which is allowed to mate) [Rojas-Guzman and Kramer, 1993]. In later work, a generalization of the inverse logarithmic scaling function was used [Rojas-Guzman and Kramer, 1996]:

$$t(\Pr(\mathbf{x})) = \frac{\varepsilon_2}{(\log(\Pr(\mathbf{x}) + \varepsilon_1))^2 + \varepsilon_1},$$

where ε_1 and ε_2 are constants. Using $\varepsilon_1 = 1 \times 10^{-12}$ and $\varepsilon_2 = 0.01$, $t(\Pr(\mathbf{x}))$ is defined for $\Pr(\mathbf{x}) \in [0, 1]$, with $t(0) \approx 0.00007$ and $t(1) \approx 1$.

GALGO’s crossover operator is cluster-oriented. A focal node F is picked in an individual, and nodes are crossed over within a distance D from F . For instance, if $D = 1$, F and nodes adjacent to F are crossed over. Mutation only mutates non-evidence nodes.

To validate the approach, a set of experiments were performed. The three experimental BNs of greatest interest are BN1, BN2, and BN3. BN1 has 13 nodes and 12 edges, and a search space size of 12,288. BN2 has 20 nodes and 24 edges, and a search space size of 7,962,624. BN3 is a variant of BN2 where the number of edges is reduced to 20. In the experiments, the least fit 20% of the individuals were replaced per generation. The mutation rates were 0.025 for BN1, 0.075 for BN2 and BN3. Termination occurs when there is approximate convergence in the population [Rojas-Guzman

Scaling scheme	Solutions	BN1	BN2	BN3
No scaling	1 in top 1	30%	0%	
	1 in top 10	100%	45%	
	1 in top 50	100%	75%	
Uniform	1 in top 1	95%	30%	
	1 in top 10	100%	60%	
	1 in top 50	100%	100%	
Inverse logarithmic	1 in top 1		20%	8%
	1 in top 10		88%	44%
	1 in top 50		100%	56%

Table 2.4: Experimental results for GALGO on BN1, BN2, and BN3. Differences in performance are shown for different scaling conditions: no scaling, uniform scaling, and inverse logarithmic scaling.

and Kramer, 1996, p. 65].

The results of the experiments are summarized in Table 2.4. The measure of performance in the table is presence of one of the best fit individuals in the population. The gold standard is based on systematic, exhaustive enumeration of the BN in question. In the table, an $X\%$ in the ‘1 in top n ’ row means that in $X\%$ of the runs, an individual among the best n was computed. From the table, we see that uniform scaling outperforms no scaling, the latter probably giving a too strong selection pressure. Inverse logarithmic scaling also counteracts this pressure, and performs as well or better than uniform scaling, except for the ‘1 in top 1’ case for BN2.

Rojas-Guzman and Kramer also experiment with the expansion level of crossover, not finding any significant differences. In addition, an optimal GA parameter set for a particular BN is suggested based on regression analysis. This parameter set is as follows: Population size $n = 150$, $\text{Pr}(\text{Mutation}) = 0.009$, breeding selectivity $B = 0.9$, average lifetime $A = 2$ (i.e. replacement ratio 50%), and crossover expansion level $L = 4$. This parameter set found optimum in 80% of the runs. Rojas-Guzman and Kramer also mention a relationship between, on the one hand, the notions of deception and epistasis in the GA literature and, on the other hand, edges and connectivity in a BN. Zero epistasis occurs in a BN with no edges; maximum epistasis in a maximally connected BN. BNs are typically locally connected, corresponding to gene interactions with a limited number of neighboring genes. This should pave the way for building block processing, however no systematic studies are undertaken to support their intuitions.

In summary, the work by Rojas-Guzman and Kramer makes contributions by representing

individuals as graphs and by controlling selection pressure by using scaling functions. However, many of the limitations identified to apply to Lin et al.’s work still hold. In general, there is a lack of theoretical understanding, for example regarding building block processing in graphs versus in strings. Also, an accuracy of 8%, 44%, and 56% for BN3 for finding an explanation among the top 1, 10, and 50 respectively leaves significant room for improvement. Why isn’t the accuracy higher? It is also a little puzzling why the optimized parameter set outperforms the non-optimized parameter set. For example, the very high replacement ratio runs contrary to many of the other research efforts discussed here.

2.3.3 Gelsema

Gelsema investigated using a GA for abductive reasoning in BNs, and emphasizes the resulting improved efficiency compared to random sampling. The mapping of a BN instantiation into a chromosomal string, the genetic operators, and the use of a BN as an objective function is done similar to the approaches described earlier.

An initial GA population was created as follows. Genes corresponding to evidence nodes were assigned alleles corresponding to the evidence values. Non-evidence root nodes were assigned values with uniform probability, non-evidence non-root nodes were assigned values according to their conditional probability. (The approach is a combination of the forward sampling algorithms uniform sampling and likelihood weighting [Shachter and Peot, 1990] [Fung and Chang, 1990], although Gelsema does not mention this.) The purpose of initializing the population in this way is to bias it towards better regions of the search space.

Steady-state roulette-wheel selection based on rank order is used, where the population is updated after each recombination. Only one of two offspring, presumably the better fit, is allowed into the population, where it replaces the least fit individual. Replacement only takes place if the individual does not exist in the population already. Mutation also operates on this individual, but does not change alleles corresponding to instantiated nodes. The values $\text{Pr}(\text{Crossover}) = 1.0$ and $\text{Pr}(\text{Mutation}) = 0.2$ were chosen ad hoc.

Three binary BNs were used for experimentation, BN1, BN2, and BN3. BN1 consists of 11 nodes and 12 edges. BN2, an extension of BN1, consists of 13 nodes and 16 edges. BN3 is an

Solutions	BN1		BN2		BN3	
	S	E	S	E	S	E
1 in top 5	99%	100%	100%	100%	100%	100%
2 in top 5	94%	96%	99%	99%	100%	100%
3 in top 5	66%	84%	90%	96%	100%	100%
4 in top 5	36%	45%	59%	68%	95%	87%
5 in top 5	9%	4%	28%	18%	69%	12%

Table 2.5: Results from Gelsema’s experiments with three BNs BN1, BN2, and BN3.

extension of BN2 and has 15 nodes and 21 edges. For all three experiments, the population size n was determined by $n = |\Omega_{\text{BN}}|/64$, where $|\Omega_{\text{BN}}|$ is the BN’s state space. The number of generations t_{max} was determined by $t_{\text{max}} = |\Omega_{\text{BN}}|/16$. After each run, the five individuals with highest fitness were recorded, and each experiment consisted of 100 runs.

The experimental results are summarized in Table 2.5. In the table, S stands for solution while E stands for explanation. A solution consists of instantiations to all non-evidence nodes. An explanation, in Gelsema’s terminology, comprises instantiations of all non-evidence disease nodes. It appears that explanations are computed from all solutions in the population by marginalizing out the non-root variables [Gelsema, 1995, p. 871]. For the top three rows, notice that the values are well above 50% for all three BNs, so the results are quite good.

In summary, Gelsema’s results are in some ways much better than those of previous efforts. Partly, this is probably due to his use of quite simple BNs as well as too large sample and population sizes (see below for more on the last point). However, credit should also be given to the approach used. In particular, comparing Table 2.5 with Table 2.4, this steady state selection strategy appears to preserve diversity better, although Gelsema does not discuss this point in the article. Also, the computation of both an explanation (diagnostic hypothesis) and a solution is a contribution; previous efforts have focused on one or the other. Finally, the biased initial population is interesting.

Some of the limitations of this work are as follows. The sizes of the sample space and the population are both unrealistic, being 1/16 and 1/64 respectively. Clearly, for larger BNs one cannot use such large fractions. In general, the use of fairly small BNs of a particular topology makes one ask whether the approach scales well. The comparison to random sampling is not very instructive, and there is very little theory supporting the experiments and the design choices made.

2.3.4 Borghetti et al.

Borghetti et al. use a GA for belief revision in a Bayesian knowledge base (BKB), which is similar to a BN but allows for incompleteness. This work focuses on scoring a GA individual when there is BKB incompleteness and the joint probability is undefined. One could assign such individuals zero fitness, however this does not give the GA any signal, which is problematic in highly incomplete BKBs. For this reason, one wants to score incomplete solutions, but not as high as any complete solution. That is, a lower bound $\Pr(\mathbf{x})$ for the least fit individual \mathbf{x} is sought such that $\Pr(\mathbf{x}) \leq \Pr(\mathbf{y})$ for all individuals \mathbf{y} . Assuming non-zero joint probabilities, an analogue of the following estimate E , cast within the BKB framework, is used for $\Pr(\mathbf{x})$:

$$E = \prod_{i=1}^n \min(\Pr(V_i | \Pi_{V_i})),$$

where $\min(\Pr(V | \Pi_V))$ is the minimal conditional probability in node V 's CPT. Now the fitness of an individual \mathbf{y} is given by $\Pr(\mathbf{y})$ if \mathbf{y} is defined, else it is given by $S(\mathbf{y})$, where S is a scoring function and $S(\mathbf{y}) < E$ for all \mathbf{y} . The scoring function is also formulated such that more incomplete solutions are penalized more than less incomplete solutions.

An experiment where only 4% of the solutions were complete showed that the approach worked. The number of complete solutions found as well as convergence to MPE improved by employing the scoring method.

Since BKBs are beyond the scope of our work, the most interesting aspect of this work is the fact that the scoring function can be regarded as an approach to scaling. An improvement to the method would be to use fitness function $\Pr(\mathbf{x})+E$ if \mathbf{x} is defined, else use $S(\mathbf{x})$. This avoids the assumption of non-zero joint probabilities.

2.3.5 Welch

Welch considers the problem of on-line belief updating in BNs [Welch, 1996]. He concludes that estimates of posterior probabilities based on an archive of trial solutions resulting from using a GA outperforms the traditional Monte Carlo simulation algorithms such as logic sampling [Henrion,

1988], forward simulation [Shachter and Peot, 1990] [Fung and Chang, 1990], and backward simulation [Fung and Favero, 1994] for this class of BNs. In particular, Welch observes experimentally that compared to a GA, the Monte Carlo simulation algorithms have problems with low-probability evidence that is introduced sequentially in on-line environments.

The initial GA population is populated from an archive as well as by performing a Monte Carlo simulation, both forward simulation and backward simulation. A steady-state GA is used, where a new individual replaces the least fit individual in the population [Welch, 1997]. Crossover works as follows. A node X is picked uniformly at random from the BN; a distance d is also selected at random. Let $M(X, d)$ be the Markov neighborhood with distance d and center at X . The offspring C of individuals A and B is formed by taking states within $M(X, d)$ from A and states outside $M(X, d)$ from B . The offspring C is then exposed to the following mutation probability:

$$\begin{aligned} \Pr(\text{Mutate } X \text{ to } X = x_i) &= \Pr(\text{Pick } X \text{ for mutation}) \\ &\quad \times \Pr(\text{Mutate to } X = x_i \mid \text{Pick } X \text{ for mutation}) \end{aligned}$$

where $\Pr(\text{Pick } X \text{ for mutation}) = 0.01$, and $\Pr(\text{Mutate to } X = x_i \mid \text{Pick } X \text{ for mutation})$ is computed as shown in Equation 2.13. Let C' be the offspring after mutation. If C' does not exist in the breeding population and its fitness is higher than the fitness of the least fit individual D , D is replaced with C' . Fitness proportionate selection is used, where two identical parents is allowed. (Note that the probability of having two identical parents is small, since all individuals in the breeding population are different).

For experimentation, the following GA parameters were used: breeding population size 40, crossover probability 0.85, and mutation probability 0.01. The maximum number of generations was varied. For a BN consisting of 32 nodes, the root mean squared error (RMSE) of the estimated belief in each node compared to the true belief was calculated. Here is a description of how computing an estimate using a GA compares to computing an estimate using backward (or evidence based) simulation [Welch, 1996, p. 540]:

A run of 10,000 trials of evidence based simulation required computation time 6 times

Number of evidence nodes	RMSE of forward simulation	RMSE of forward simulation and GA hybrid
0	0.0091	0.0173
1	0.0161	0.0149
2	0.0221	0.0193
3	0.1134	0.0714
4	0.0951	0.0337

Table 2.6: Root mean squared error (RMSE) for forward simulation versus hybrid of forward simulation and GA as a function of the number of evidence nodes.

greater than a combination of 5,000 backward simulations and 2,500 genetic search trials. Even in that case, the archive RMSE was 0.000042, much better than for the lengthy simulation.

To confirm the positive findings for the GA for the 32 node BN, 12 random BNs were generated at random. Each node consisted of 2–4 states and had 0–3 parents, with each state biased towards zero: Each CPT entry has a probability of 0.5 of being zero; if non-zero it is chosen uniformly from the interval $(0, 1]$ under the constraint that entries in a CPT column must add up to one. The average results for these 12 randomly generated BNs are summarized in Table 2.6. The trend here is quite clear: the forward simulation and GA hybrid outperforms pure forward simulation as the number of evidence nodes increases.

The results of Welch are very encouraging from the GA point of view. In particular, Welch shows that a GA is especially useful when evidence has low probability or when sequential updating in a on-line environment is required. The archive and the GA replacement strategy seem to be two important factors. Also, he established interesting performance differences for logic sampling, forward sampling, and backward sampling, although this has not been the emphasis here.

These are some limitations of this study. First, it is not clear to what extent the GA benefits from the initialization of its initial population by forward simulation. Why not just run the GA right away? It might be beneficial to run both forward simulation and the GA, but this is not clear from the paper. Second, there is a lack of theory in the paper. Why does the GA work quite well here, but less so in other previous research? Third, how sensitive is the GA to the skewed distribution used by Welch, in particular as represented in the random BNs?

2.3.6 Miscellaneous

Santos and Shimony present a system with two components: a generator and an evaluator [Santos et al., 1996] [Santos et al., 1997]. The generator, a randomized approximation algorithm, provides approximations quickly. The evaluator, a deterministic approximation algorithm, provides probability bounds. By combining them, Santos and Shimony attempt to get the best of both worlds: The generator searches for highly probable assignments, the evaluator accumulates these assignments to perform belief updating and get bounds. The focus is on high-probability partial assignments, so-called independence-based assignments.

Two different GAs are considered as generators [Santos et al., 1997]. First, a GA with complete assignments is considered. Santos et al. base their GA on the GA of Miller et al. Miller et al. used a GA to the NP-hard problem of multiple fault diagnosis [Miller et al., 1993]. They used a hybrid approach, exploiting local improvement operators, resulting in the GA finding an optimal solution in almost all cases, with good performance. The three most important parts of this research are multiple fault diagnosis, the relative likelihood function, and the genetic algorithm. Multiple fault diagnosis corresponds to a diagnostic BN with multiple root nodes where one needs to consider belief revision and not just belief updating. The relative likelihood function is that of Peng and Reggia’s probabilistic causal model [Peng and Reggia, 1987a] [Peng and Reggia, 1987b], where “relative likelihood gives us the capability to compare diagnoses and select the best one.” The GA is modeled after the simple GA [Goldberg, 1989c], but with the following modifications: two-point crossover, elitism, and a slight initialization bias. Elitism is implemented by keeping the best fit individual if it is better than the worst fit in the previous generation. The initialization bias is due to insertion of an individual with all relevant disorders present. Santos et al. adapt this GA as follows [Santos et al., 1997]: “The mutation rate is relatively high, since we do *not* want to run the genetic algorithm to convergence, but rather to explore the search space. This is also the reason for avoiding the elitist policy.” In addition, the GA of Santos et al. is based on independence-based assignments, where all BN nodes are not necessarily assigned a value. However, within the GA population, all nodes of each individual do have a value.

The second GA that is considered is a messy GA using independence-based assignments within the GA. The use of independence-based assignments within the GA means that the number of

nodes as well as which nodes are assigned differs within the GA population.

Both GAs were used for experimentation. Experiments with BNs containing 2000 nodes and 500 nodes respectively turned out not to be very successful. The two GAs were inferior to cost-sharing heuristic search and to other sampling algorithms.

We now turn to a discussion of this research. The presentation of the GAs used as well as the experiments performed is incomplete, so it is difficult to discuss this research in detail. However, the idea of using of partial assignments, IB assignments, is novel. In many ways, this research is similar to that of Welch, since he also suggest a hybrid approach with a generator (the GA) and an evaluator (the archive). Welch only considers complete assignments, though, and he also does not try to estimate bounds. However, Santos and Shimony do not try to estimate bounds for their GA generator, only for other generators.

Kanazawa et al. focus on the problem of divergence in BN simulation algorithms, in particular as it manifests itself when these algorithms are applied to temporal Bayesian networks [Kanazawa et al., 1995] [Koller, 1996]. The problem is that simulation trials diverge from reality as a process is observed. Three algorithms that attack this problem are introduced. The first two are evidence reversal and survival of the fittest sampling, the third is a combination of these two algorithms. Survival of the fittest sampling (SOF) is closely related to selection in GAs in that highly likely instantiations are propagated to the next time slice in the temporal BN. In particular, the SOF algorithm [Kanazawa et al., 1995]

keeps a fixed number of samples, but generates the sample population for time slice t by a weighted random selection from the samples at time $t - 1$, where the weight is given by the likelihood for the evidence observed at time t . This is closely related to the use of fitness-related propagation in genetic algorithms [...].

After sampling from the approximated belief state of the previous time slice $t - 1$, weighting according to evidence at time t takes place. This produces weighted samples that form an approximated belief state for time slice t , and the process start over again.

All of the three new algorithms (evidence reversal, SOF, and the combined algorithm) outperform likelihood weighing (or forward sampling) [Shachter and Peot, 1990] [Fung and Chang, 1990]

on experimental temporal BNs. Further theoretical and experimental results on SOF are presented by Koller [Koller, 1996].

We now turn to discussing SOF. SOF is in many ways similar to a GA, as noted above. Differences between a GA and SOF are that GAs generally don't have state transition probabilities defined, despite the obvious similarity between a time slice increment in SOF and generation increment in a GA. The essential difference is that a time slice increment is associated with external time, a generation increment with computational time.

Larranaga et al. have focused on structure learning, Bayesian network fusion, and Bayesian network decomposition [Larranaga et al., 1996b] [Larranaga et al., 1996c] [Larranaga et al., 1996a]. Of these issues, only Bayesian network decomposition [Larranaga et al., 1996b] is related to BN inference insofar as it is one step in the process of creating a junction tree from a BN, where the junction tree is used for exact BN inference [Lauritzen and Spiegelhalter, 1988]. This research is of some but limited relevance to the present research since we do not focus on exact Bayesian network inference.

2.4 Summary

In this chapter, we have introduced Bayesian network terminology and reviewed previous research. Clearly, both exact computation and inexact computations have their strong points as well as their limitations. In the following, however, we focus on comparing the different GA techniques, since their performance seems to be the least understood, and in addition they form one of the main algorithmic approaches studied here.

The different approaches taken to BN inference using GAs are summarized in Table 2.7. An entry '?' somewhere means that the information corresponding to the entry is not available or is ambiguous. In the first column, initials for the relevant researchers are presented. These initials should be easy to correlate to the work presented already. Most columns should be easy to understand. Generation gap gives the percentage of the population replaced per generation. The maximum generation gap is $n/n = 1$; minimum generation gap is $1/n$. Replacement information is of the form 'X, Y', meaning that the novel individual(s) as described by X replace(s) the individual(s) as described by Y. For example 'one, worst if not present' means that a newly created

individual is admitted into the population if it is not there from before, replacing the worst fit individual.

An important piece of information that is lacking in the table because it is lacking in most papers is population size. In the table, it is striking how there is a strong tendency towards a steady state GA rather than a generational GA. Another important issue is the amount of hybridization. There are two dimensions of hybridization, the data dimension and the control dimension. The data dimension concerns whether the same data structures are used for the GA and the ‘other’ algorithm. The best example of data hybridization is the research by Rojas-Guzman and Kramer [Rojas-Guzman and Kramer, 1993] [Rojas-Guzman, 1995] [Rojas-Guzman and Kramer, 1996]. The control dimension concerns how control flows between GA and the other algorithm. The other algorithm can be used for pre- or post-processing, or one can have an integrated hybrid. Both Gelsema and Welch use a hybrid approach in that a simulation algorithm initializes the GA’s population [Gelsema, 1995, p. 871] [Welch, 1996, p. 538].

Although the research efforts in the area of using GAs for BN inference have neither been extensive nor well coordinated, important insights have been made. In particular, the following may be concluded regarding the use of GAs for BN inference:

- Does it work? Yes, GAs can perform inference in BNs, as witnessed in particular by the research by Rojas-Guzman, Gelsema and Welch. In particular, the research by Welch is convincing because the GA is compared with and outperforms several well-known Monte Carlo simulation algorithms.
- Why does it work? It seems that the selection and replacement strategies used in the GA is crucial. Most GAs have used steady-state selection rather than generational selection. However, this distinction is in itself not necessarily significant; what is more important is replacement strategy and the like [Syswerda, 1991]. Judging from Table 2.7, it does seem that replacing the worst individual(s) and only letting a new individual into the breeding population if it does not exist there beforehand is a good strategy. We conjecture that this variant of steady-state selection preserves diversity well, while other variants do not. However, at this time this is only a conjecture, it might be that other aspects, such as hybridization, is as or more important.

Who	Generation gap	Selection	Replacement	Scaling
LGS	$1/n$	roulette wheel, uniform	one, below-average	no
RGK	0.2	fitness proportional	two, worst	yes
G	$1/n$	roulette wheel, rank	one, worst if not present	no
SSW	1	?	all	?
W	$1/n$	fitness proportional	one, worst if not present	no

Who	Crossover	Mutation
LGS	one-point/uniform, 1.0	yes, MB score
RGK	cluster, ?	yes, 0.025/0.075
G	two-point, 1.0	yes, 0.2
SSW	two-point, 0.6	yes, > 0.033
W	MB crossover, 0.85	yes, 0.01/MB score

Who	Coding	Inclusion	Initialization	Termination
LGS	string	part of BN	uniform	1000 generations
RGK	graph	full BN	uniform	convergence
G	string	full BN	biased	$ \Omega_{\text{BN}} /16$ samples
SSW	string	part of BN	biased	50 generations
W	string	full BN	biased	convergence

Table 2.7: Summary of previous research to BN inference using GAs. The ‘Who’ column gives initials for the researchers of the approach as follows. LGS: Lin, Galper, and Shachter. RGK: Rojas-Guzman and Kramer. G: Gelsema. SSW: Santos, Shimony, and Williams. W: Welch.

The question of why a GA works for BN inference has received a partial answer. The following are limitations to the research performed so far, limitations that will be addressed in the present research:

- Lack of theory. Previous work has been almost exclusively experimental. There has been some mention of building blocks and epistasis [Rojas-Guzman and Kramer, 1996] and multimodality [Lin et al., 1990, p. 129], however these remarks have been anecdotal and have not been followed up by theoretical studies based on BNs. For example, crucial GA issues such as diversity preservation and population sizing has not been addressed in the GA for BN inference literature. Although a comprehensive GA theory is lacking, there has been progress in the GA community in this area over the recent years, and this progress should be utilized.
- Lack of comparison between different GAs. Typically, a research team has considered one GA with one fixed set of parameters without varying the basic setup much. Again, there are

many things that need to be considered for a GA to work properly, and even though one does not want to consider all possible GA parameter settings it is clear that more empirical studies could be performed.

- Lack of systematic experimentation. It is not clear that one has used BNs that are in some sense hard, either for exact inference or for the stochastic search method or genetic algorithm in question.

In the remainder of this dissertation, we attempt to address some of the limitations of the previous research in this and related areas.

Chapter 3

The Probabilistic Crowding Genetic Algorithm

When searching for probable explanations in a Bayesian network, one might be interested in finding not only a most probable explanation, but all the most probable explanations, or even many highly probable explanations, even though several of them might have lower probabilities than a most probable explanation. Also, Bayesian networks define, in the general case, complex multi-modal functions. These two goals lead us to consider the genetic algorithm technique of niching.

In more general terms than that of computing the most probable explanation, the two main objectives of niching algorithms are (i) to converge to multiple, highly fit, and significantly different solutions, and (ii) to slow down convergence in cases where only one solution is required. Different algorithms have been developed to fulfil these objectives [Goldberg and Richardson, 1987] [Harik, 1995] [Mahfoud, 1995]. One of these algorithms is known as deterministic crowding [Mahfoud, 1995]. Strengths of deterministic crowding are that it is simple, fast, and requires no parameters in addition to those of a classical GA. Deterministic crowding has also been found to work well on test functions as well as in applications. However, deterministic crowding also has some weak points. There is a lack of analysis of convergence; as a result it is not entirely clear what deterministic crowding computes. Considering the internal workings of the algorithm, the main problem appears to be that there is no restorative pressure—species of higher fitness tend to win over species of lower fitness—thus niches may get lost even though they should not be according to their fitness.

This chapter introduces a new niching algorithm, *probabilistic crowding*. As the name suggests, probabilistic crowding is an offspring of deterministic crowding, and as such inherits many of its pleasant characteristics. The main difference is the use of a probabilistic rather than a deterministic

	Population-based	Non-population-based
Probabilistic acceptance	Probabilistic crowding Parallel recombinative simulated annealing	Metropolis algorithm Simulated annealing
Deterministic acceptance	Deterministic crowding Restricted tournament selection	Local search

Table 3.1: Two key dimensions of integrated tournament algorithms: nature of the acceptance rule and nature of the current state.

acceptance function. No longer do stronger individuals always win over weaker individuals, they win proportionally according to their fitness, thus we get a restorative pressure. Using a probabilistic acceptance function is shown to give stable, predictable convergence according to the niching rule, a gold standard for niching algorithms. We show this both analytically and experimentally.

A second purpose of this chapter is to briefly review the family of algorithms to which both deterministic and probabilistic crowding belongs, *integrated tournament algorithms*. Other members of this class are restricted tournament selection [Harik, 1995], elitist recombination [Thierens and Goldberg, 1994], parallel recombinative simulated annealing [Mahfoud and Goldberg, 1995], the Metropolis algorithm [Metropolis et al., 1953], and simulated annealing [Kirkpatrick et al., 1983]. Common to these algorithms is that competition is localized and occurs between what we might call a family of similar individuals. It turns out that slight variations in how the family is formed is crucial to whether one obtains a niching algorithm or not, and more generally this class of algorithms is interesting because it is very efficient and gives a wide spectrum of functionality which can be attained by changing a few parameters.

The rest of this chapter is organized as follows. Section 3.1 presents integrated tournament algorithms. Section 3.2 introduces the probabilistic crowding algorithm. In Section 3.3, we analyze certain variants of probabilistic crowding. Section 3.4 and 3.5 gives empirical evidence that probabilistic crowding works well, while Section 3.6 focuses on population sizing. Section 3.7 concludes and points out directions for future research.

3.1 Integrated Tournament Algorithms

In traditional GAs, mutation and recombination is done first, and then selection (or replacement) is performed second, without regard to similarity between individuals. Many algorithms, such as probabilistic crowding, deterministic crowding, parallel recombinative simulated annealing, restricted tournament selection, the Metropolis algorithm, and simulated annealing work differently, although this distinction has not always been clearly expressed in the literature. What these algorithms, which we shall call integrated tournament algorithms, have in common is that the processes of mutation, recombination, and replacement are all integrated. Intuitively, integrated tournament algorithms can give niching through local tournaments: Similar individuals compete for spots in the population, and fit individuals replace those that are less fit, at least probabilistically. The exact nature of the tournament depends on the algorithm, and is a crucial factor in deciding whether we get a niching algorithm or not. For instance, elitist recombination [Thierens and Goldberg, 1994] is an integrated tournament algorithm, but it is not a niching algorithm.

An early integrated tournament algorithm is the Metropolis algorithm, which originated in physics [Metropolis et al., 1953], and consists of generation and acceptance steps [Neal, 1993]. In the generation step, a new state (or individual) is generated from an existing state; in the acceptance step, the new state is accepted or rejected with some probability. Two common acceptance probability distributions are the Metropolis and the Boltzmann distributions. The Boltzmann distribution is

$$\Pr(E_j) = \frac{\exp(-E_j/T)}{\exp(-E_j/T) + \exp(-E_i/T)}, \quad (3.1)$$

where E_i and E_j are the energies of the old and new states (individuals) respectively.

Simulated annealing is essentially the Metropolis algorithm with temperature added. The temperature controls the probability of accepting a higher-energy (less fit) state (individual). At high temperature, this probability is very high, but it decreases with the temperature. Simulated annealing consists of iterating the Metropolis algorithm at successively lower temperatures, and this way it finds an estimate of the global optimum [Kirkpatrick et al., 1983] [Laarhoven and Aarts,

1987]. Both the Metropolis rule and the Boltzmann rule achieve the Boltzmann distribution

$$\Pr(E_i) = \frac{\exp(-E_i/T)}{\sum_j \exp(-E_j/T)}, \quad (3.2)$$

where $\Pr(E_i)$ is the probability of having a state i with energy E_i at equilibrium, T is temperature. If cooling is slow enough, one is guaranteed to find the optimum.

Within the field of genetic algorithms proper, an early integrated tournament approach is preselection. Cavicchio introduced preselection, in which a child replaces an inferior parent [Goldberg, 1989c]. DeJong turned preselection into crowding [DeJong, 1975]. In crowding, an individual is compared to a randomly drawn subpopulation of c members, and the most similar member among the c is replaced. Good results with $c = 2$ and $c = 3$ were reported by DeJong on multimodal functions.

In order to integrate simulated annealing and genetic algorithms, the notion of Boltzmann tournament selection was introduced [Goldberg, 1990]. Two motivations for Boltzmann tournament selection were asymptotic convergence (as in simulated annealing) and providing a niching mechanism. The Boltzmann (or logistic) acceptance rule, shown in Equation 3.1, was used. Boltzmann tournament selection was the basis for parallel recombinative simulated annealing (PRSA) [Mahfoud and Goldberg, 1995]. PRSA also used Boltzmann acceptance, and introduced the following two rules for handling children and parents: (i) In double acceptance and rejection, both parents compete against both children. (ii) In single acceptance and rejection, each parent competes against a pre-determined child in two distinct competitions. Like simulated annealing, PRSA uses a cooling schedule. Both mutation and crossover are used, to guarantee convergence to the Boltzmann distribution at equilibrium. Three different variants of PRSA were tested empirically with good results, two of these have proofs of global convergence. Deterministic crowding [Mahfoud, 1995] is similar to PRSA. Differences are that deterministic crowding matches up parents and children by minimizing some distance measure, and it uses the deterministic acceptance rule of always picking the best fit individual in each parent and child pair.

Another integrated tournament algorithm is the gene-invariant GA (GIGA). In GIGA, children replace the parents [Culberson, 1992]. Parents are selected, a family constructed, children selected, and parents replaced. Family construction amounts to creating a set of pairs of children, and from

this set one pair is picked according to some criterion, such as highest average fitness or highest maximal fitness. The genetic invariance principle is that the distribution over any one position on the gene does not change over time. GIGA with no mutation obeys the genetic invariance principle, so the genetic material of the initial population is retained. In addition to selection pressure provided by selection of better child pairs in a family, there is selection pressure due to sorting of the population combined with selection of adjacent individuals.

Restricted tournament selection is another integrated tournament algorithm [Harik, 1995]. The approach is a modification of standard tournament selection, based on local competition. Two individuals \mathbf{x} and \mathbf{y} are picked, and crossover and mutation is performed in the usual way, creating new individuals \mathbf{x}' and \mathbf{y}' . Then w individuals are randomly chosen for \mathbf{x}' , and among these the closest one, \mathbf{x}'' , competes with \mathbf{x}' for a spot in the new population. A similar procedure is applied to \mathbf{y}' . The parameter w is called the window size. The window size is set to be a multiple of s , the number of peaks to be found: $w = c \times s$, where c is a constant. Restricted tournament selection illustrates that integrated tournament algorithms only need to have their operations conceptually integrated; the key point is that individuals compete locally (with similar individuals) for a spot in the population.

In summary, important dimensions of integrated tournament algorithms are the form of the acceptance rule, whether the algorithm is population-based, whether temperature is used, which operators are used, and whether the algorithm gives niching or not. Table 3.1 shows two of the key dimensions of integrated tournament algorithms, and how different algorithms are classified along these two dimensions. The importance of the distinction between probabilistic and deterministic acceptance is as follows. It seems easier to maintain a diverse population with probabilistic acceptance, and this is the goal of niching algorithms. Processes similar to probabilistic acceptance occur elsewhere in nature, for instance in chemical reactions and in statistical mechanics.

Concerning operators, one important distinction is whether similar individuals are brought together to compete implicitly or explicitly. The *implicit* approach, of which PRSA, deterministic crowding, probabilistic crowding are examples, integrate the operations of variation and selection. The *explicit* approach, examples of which are crowding and restricted tournament selection, search for similar individuals in the population. So in addition to variation and selection, there is a search

step. Note that explicit versus implicit is a matter of degree, since even deterministic crowding with crossover searches for a given child for the closest among the parents. Whether the integrated tournament algorithm gives niching or not depends on the nature of the family competition. If the family competition is based on similarity, such that two or more similar individuals compete for a place in the population, the result is niching, else no niching is obtained. For example, deterministic crowding, restricted tournament selection, and probabilistic crowding are niching algorithms, while elitist recombination and GIGA are not.

The probabilistic crowding algorithm and its steady state distribution is what we turn to in the next sections.

3.2 Probabilistic Crowding Algorithm

Probabilistic crowding is based on the deterministic crowding algorithm [Mahfoud, 1995]; the two main differences being (i) the probabilistic acceptance rule and (ii) the fact that we have a variant without crossover at all.

Let \mathbf{x} and \mathbf{y} be two similar individuals that have been picked to compete to replace one of these two individuals in the next generation. Similarity comes about implicitly, when mutation only is employed, or explicitly, by using a distance measure in connection with crossover or explicit search for family members. In probabilistic crowding, \mathbf{x} and \mathbf{y} compete in a probabilistic tournament. The probability of \mathbf{x} winning is given by:

$$p_{\mathbf{x}} = p(\mathbf{x}) = \frac{f(\mathbf{x})}{f(\mathbf{x}) + f(\mathbf{y})}, \quad (3.3)$$

where f is the fitness (or objective) function. Notice that probabilistic crowding is primarily a distance-based niching algorithm, since competition occurs within families, between similar individuals. Here, the family consists of two members \mathbf{x} and \mathbf{y} , but this can easily be generalized to larger families.

Three variants of the probabilistic crowding algorithm have been investigated: Variant M (with mutation only), Variant M&C (with mutation and crossover), and Variant C (with crossover only). These variants differ in the way in which one attains \mathbf{x} and \mathbf{y} ; and also when crossover is used a

measure of distance is needed.

One of the most important questions to ask about an integrated tournament algorithm is what the characteristics of its steady-state (equilibrium) distribution are. In particular, we are interested in this for niches. A niche is a set of fitness function values that have the same local optimum under some local search algorithm; see [Mahfoud, 1995] for details. The notation $\mathbf{x} \in \mathbb{X}$ will be used to indicate that individual \mathbf{x} is a member of niche \mathbb{X} . Let q be the number of niches, and \mathbb{X}_i the i -th niche. The *niching rule*

$$N_i = \frac{f_i}{\sum_{j=1}^q f_j} \quad (3.4)$$

gives allocation of N_i individuals to \mathbb{X}_i . Here, f_j is a measure of fitness in niche \mathbb{X}_j , for example fitness of best fit, average fitness, or fitness sum. The niching rule, which can be derived from the sharing rule [Goldberg and Richardson, 1987], is considered a gold standard for niching algorithms. In the following we will see how probabilistic crowding gives this rule as a special case.

3.3 Analysis of Probabilistic Crowding

We analyze probabilistic crowding, first the special case with two niches, second the more general case with several niches. Two kinds of analyses are provided: at steady state and of the form of convergence of the population. We assume some variation operator, which typically would be mutation or crossover. In the analysis we consider one representative per niche; for example if the niche is \mathbb{X} , the representative is \mathbf{x} . We perform a deterministic analysis, thus focusing on the mean in the stochastic processing of a GA.

3.3.1 Two Niches, Same Jump Probabilities

Suppose we have a variation operator that results in two types of jumps; short jumps and long jumps. When an individual is treated with a short jump it stays within its niche, when it is treated with a long jump it jumps to some other niche. The probabilities are p_s and p_l respectively, and $p_s + p_l = 1$. That is, we either jump short or long.

Consider mother \mathbf{m} (individual before variation operator was applied) and daughter \mathbf{d} (individ-

ual after variation operator was applied). Suppose we have niches \mathbb{X} and \mathbb{Y} , and think about how \mathbb{X} can gain individuals from one generation to the next. (i) The first possibility is $\mathbf{m} \in \mathbb{X}$. The first case is that the daughter \mathbf{d} stays in the mother \mathbf{m} 's niche if a short jump is made; in this case it doesn't matter whether \mathbf{m} or \mathbf{d} win since both are in the same niche. The second case is that the daughter jumps and loses. (The case where the daughter jumps and wins is a loss for \mathbb{X} , and is not participating in the difference equation below.) (ii) The second possibility is that $\mathbf{m} \in \mathbb{Y}$. Now, gain for niche \mathbb{X} happens when the daughter jumps to \mathbb{X} and wins.

The above argument can be formalized in a difference equation. Let the proportion of individuals in niche \mathbb{Z} at generation t be $Z(t)$. By assumption we have two niches, \mathbb{X} and \mathbb{Y} , and the proportions of interest at time t are denoted $X(t)$ and $Y(t)$ respectively. Note that $X(t) + Y(t) = 1$ for any t . This gives rise to the following difference equation

$$\begin{aligned} X(t+1) &= p_s X(t) + p_l p_{\mathbf{x}} X(t) + p_l p_{\mathbf{x}} Y(t) \\ &= X(t) - p_l X(t) + p_l p_{\mathbf{x}}. \end{aligned} \tag{3.5}$$

We will solve this equation in two ways—considering the steady state and getting a closed form formula. At steady state we have $X(t+1) = X(t) = X_{ss}$, substituting this into Equation 3.5 leads to

$$\begin{aligned} X_{ss} &= X_{ss} - p_l X_{ss} + p_l p_{\mathbf{x}} \\ &= p_{\mathbf{x}} \\ &= \frac{f(\mathbf{x})}{f(\mathbf{x}) + f(\mathbf{y})}, \end{aligned} \tag{3.6}$$

where $\mathbf{x} \in \mathbb{X}$, $\mathbf{y} \in \mathbb{Y}$. In words, we get the niching rule of Equation 3.4 at steady state, as one would hope for.

Now we turn to obtaining a closed form formula. Considering two niche proportions $X(t)$ and $Y(t)$, we have

$$\begin{aligned} X(t+1) &= p_s X(t) + p_l p_{\mathbf{x}} X(t) + p_l p_{\mathbf{x}} Y(t) \\ Y(t+1) &= p_s Y(t) + p_l p_{\mathbf{y}} Y(t) + p_l p_{\mathbf{y}} X(t) \end{aligned} \tag{3.7}$$

The solution to the above system of difference equations can be written as:

$$\begin{aligned} X(t) &= p_{\mathbf{x}} + p_s^t X(0) - p_s^t p_{\mathbf{x}} X(0) - p_s^t p_{\mathbf{x}} Y(0) \\ Y(t) &= p_{\mathbf{y}} - p_s^t X(0) + p_s^t p_{\mathbf{x}} X(0) + p_s^t p_{\mathbf{x}} Y(0), \end{aligned} \quad (3.8)$$

and we see how as t tends to infinity we get the niching rule, expressed as $p_{\mathbf{x}}$ and $p_{\mathbf{y}}$, for both niches.

For illustration, suppose $X(0) = Y(0) = \frac{1}{2}$ in the initial population. This gives solutions:

$$\begin{aligned} Y(t) &= p_{\mathbf{y}} + \left(\frac{1}{2} - p_{\mathbf{y}}\right) p_s^t \\ X(t) &= p_{\mathbf{x}} + \left(\frac{1}{2} - p_{\mathbf{x}}\right) p_s^t, \end{aligned} \quad (3.9)$$

and again we see how we get the desired result as t tends to infinity. Also note that a smaller p_s gives faster convergence to the niching rule proportion. In summary, we see that initialization does not affect the fact that the niching rule is achieved in the limit.

3.3.2 Two Niches, Different Jump Probabilities

Here we relax the assumption of equal jump probabilities for the two niches. Rather than jump probabilities p_s and p_l , we have jump probabilities p_{ij} for jumping from niche \mathbb{X}_i to niche \mathbb{X}_j , where $i, j \in \{0, 1\}$. We also use the notation $X_i(t)$ for the proportion of individuals in niche \mathbb{X}_i at time t . The facts $p_{11} + p_{12} = 1$ and $p_{21} + p_{22} = 1$ are used below, too.

We obtain an expression for $X_1(t+1)$ using reasoning similar to that used for Equation 3.5, for $X_1(t+1)$ we get:

$$\begin{aligned} X_1(t+1) &= p_{11}X_1(t) + p_{12}p_{\mathbf{x}}X_1(t) + p_{21}p_{\mathbf{x}}X_2(t) \\ &= p_{11}X_1(t) + (1 - p_{11})p_{\mathbf{x}}X_1(t) + p_{21}p_{\mathbf{x}}(1 - X_1(t)) \\ &= p_{11}X_1(t) + p_{\mathbf{x}}X_1(t) - p_{11}p_{\mathbf{x}}X_1(t) - p_{21}p_{\mathbf{x}}X_1(t) + p_{21}p_{\mathbf{x}}. \end{aligned} \quad (3.10)$$

At steady state we have $X_1(t+1) = X_1(t) = X_1$, leading to

$$X_1 = p_{11}X_1 + p_{\mathbf{x}}X_1 - p_{11}p_{\mathbf{x}}X_1 - p_{21}p_{\mathbf{x}}X_1 + p_{21}p_{\mathbf{x}}$$

which after some manipulation simplifies to the following allocation ratio for niche \mathbb{X}_1

$$X_1 = \frac{p_{\mathbf{x}_1}}{p_{\mathbf{x}_1} + \frac{p_{12}}{p_{21}}p_{\mathbf{x}_2}} = \frac{p_{\mathbf{x}_1}}{p_{\mathbf{x}_1} + \rho_{12}p_{\mathbf{x}_2}}. \quad (3.11)$$

Here, $\rho_{12} = \frac{p_{12}}{p_{21}}$ is denoted the transmission ratio from \mathbb{X}_1 to \mathbb{X}_2 . In general, we have that ρ_{ij} is the transmission ratio from niche \mathbb{X}_i to \mathbb{X}_j . Clearly, ρ_{12} is large if the in-flow into \mathbb{X}_2 is large relative to the out-flow from \mathbb{X}_2 . We may compare Equation 3.11 to the niching rule proportion N_1 according to Equation 3.4:

$$N_1 = \frac{p_{\mathbf{x}_1}}{p_{\mathbf{x}_1} + p_{\mathbf{x}_2}}. \quad (3.12)$$

Clearly, using $\rho_{12} = 1$ in Equation 3.11 gives the same allocation as Equation 3.12. $\rho_{12} > 1$ means that niche \mathbb{X}_2 will have a larger subpopulation at equilibrium than under perfect niching, giving \mathbb{X}_1 a smaller subpopulation, while $\rho_{12} < 1$ means that \mathbb{X}_2 's subpopulation at equilibrium will be smaller than under perfect niching, giving \mathbb{X}_1 a larger subpopulation.

The size of a niche as well as the operators used will have an impact on p_{12} and p_{21} . Disregarding the effect of operators, a large \mathbb{X}_2 niche will have $\rho_{12} > 1$ and therefore give a smaller subpopulation for \mathbb{X}_1 . Likewise, a small \mathbb{X}_2 niche will have $\rho_{12} < 1$ and thus a larger subpopulation for \mathbb{X}_1 .

Along similar lines, the ratio for niche \mathbb{X}_2 turns out to be

$$X_2 = \frac{p_{\mathbf{x}_2}}{\frac{p_{21}}{p_{12}}p_{\mathbf{x}_1} + p_{\mathbf{x}_2}} = \frac{p_{\mathbf{x}_2}}{\rho_{21}p_{\mathbf{x}_1} + p_{\mathbf{x}_2}},$$

with $\rho_{21} = \frac{p_{21}}{p_{12}}$.

Note that values for ρ_{12} and ρ_{21} , or more generally ρ_{ij} for the transmission ratio from niche i to j , will be unknown. So one can not use known values for ρ_{12} and ρ_{21} in the equations above, for instance. However, it is possible to estimate transmission ratios using sampling or one may use worst-case values.

Finally, note that the same result can be established by solving these two simultaneous difference equations:

$$\begin{aligned} X_1(t+1) &= p_{11}X_1(t) + p_{12}p_{\mathbf{x}}X_1(t) + p_{21}p_{\mathbf{x}}X_2(t) \\ X_2(t+1) &= p_{22}X_2(t) + p_{21}p_{\mathbf{Y}}X_2(t) + p_{12}p_{\mathbf{Y}}X_1(t), \end{aligned}$$

which yields fairly complex solutions which can be solved by eliminating all terms with generation t in the exponent. These solutions can then be simplified, giving exactly the same result as above.

3.3.3 Multiple Niches, Different Jump Probabilities

We generalize from two to q niches. Let the probability of transfer from the i -th to j -th niche under the variation operator be p_{ij} , where $\sum_{j=1}^q p_{ij} = 1$. The probability of the individual $\mathbf{x}_i \in \mathbb{X}_i$ winning over the individual $\mathbf{x}_j \in \mathbb{X}_j$ is

$$p_{ij}^* = \frac{f(\mathbf{x}_i)}{f(\mathbf{x}_i) + f(\mathbf{x}_j)}.$$

We can now set up the following system of i difference equations:

$$X_i(t+1) = \sum_j p_{ij}p_{ij}^* X_i(t) + \sum_j p_{ji}p_{ij}^* X_j(t)$$

Unfortunately, these equations are hard to solve. But by introducing the assumption of local balance (known as detailed balance in physics [Laarhoven and Aarts, 1987]), progress can be made. The condition is [Neal, 1993, p. 37]

$$X_i p_{ij} p_{ji}^* = X_j p_{ji} p_{ij}^*. \tag{3.13}$$

The local balance assumption is that individuals (or states) are in balance: The probability of an individual \mathbf{x}_i being transformed into another individual \mathbf{x}_j is the same as the probability of the second individual \mathbf{x}_j being transformed into the first individual \mathbf{x}_i . We can assume this is for a niche rather than for an individual, similar to what we did above, thus giving Equation 3.13. On

the left-hand side of Equation 3.13 we have the probability of escaping niche \mathbb{X}_i , on the right-hand side of Equation 3.13 we have the probability of escaping niche \mathbb{X}_j . Simple rearrangement gives

$$X_i = \frac{p_{ji}p_{ij}^*}{p_{ij}p_{ji}^*} X_j = \rho_{ji} \frac{f(\mathbf{x}_i)}{f(\mathbf{x}_j)} X_j, \quad (3.14)$$

where \mathbf{x}_i and \mathbf{x}_j are representatives for niches \mathbb{X}_i and \mathbb{X}_j respectively. Consider the k -th niche, and express all other niches, using Equation 3.14, in terms of this niche. In particular, express an arbitrary X_i using a particular X_k :

$$X_i = \rho_{ki} \frac{f(\mathbf{x}_i)}{f(\mathbf{x}_k)} X_k.$$

Now, we introduce the fact that $X_1 + \dots + X_q = 1$, where q is the number of niches:

$$\rho_{k1} \frac{f(\mathbf{x}_1)}{f(\mathbf{x}_k)} X_k + \rho_{k2} \frac{f(\mathbf{x}_2)}{f(\mathbf{x}_k)} X_k + \dots + X_k + \dots + \rho_{kq} \frac{f(\mathbf{x}_q)}{f(\mathbf{x}_k)} X_k = 1.$$

Solving for X_k gives

$$X_k = \frac{1}{\rho_{k1} \frac{f(\mathbf{x}_1)}{f(\mathbf{x}_k)} + \rho_{k2} \frac{f(\mathbf{x}_2)}{f(\mathbf{x}_k)} + \dots + 1 + \dots + \rho_{kq} \frac{f(\mathbf{x}_q)}{f(\mathbf{x}_k)}},$$

and we may use the fact that

$$\frac{f(\mathbf{x}_k)}{f(\mathbf{x}_k)} \rho_{kk} = 1.$$

The above two equations combined gives

$$X_k = \frac{f(\mathbf{x}_k)}{\sum_{i=1}^q \rho_{ki} f(\mathbf{x}_i)}, \quad (3.15)$$

where we define $\rho_{kk} = 1$. Notice how the transmission ratio ρ_{ki} from \mathbb{X}_k to \mathbb{X}_i generalizes the transmission ratio ρ_{12} from Equation 3.11. Equation 3.15 is the most general theoretical result on probabilistic crowding presented in this chapter; it generalizes the niching rule of Equation 3.4. The niching rule applies to sharing with roulette-wheel selection [Mahfoud, 1995], and much of that

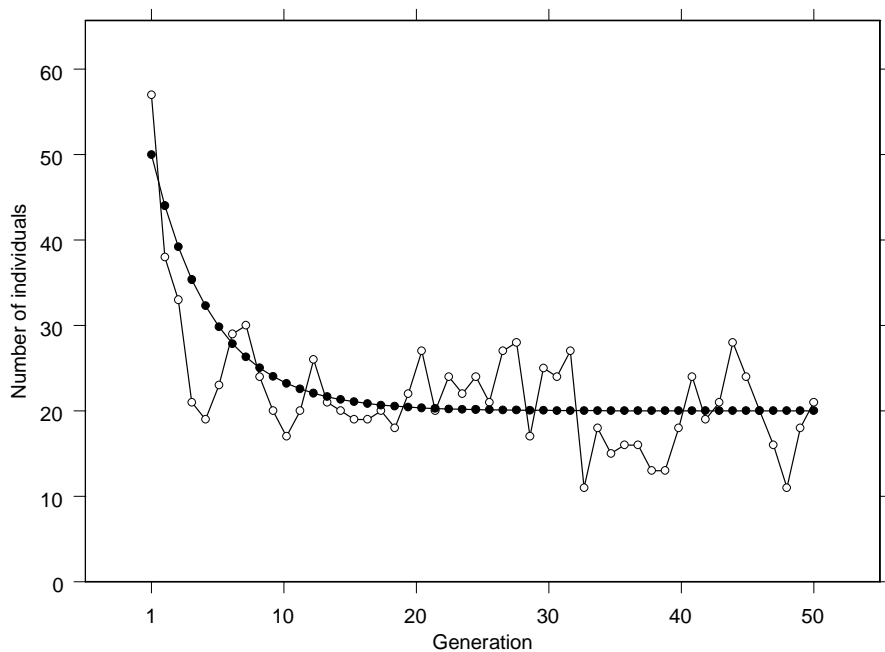


Figure 3.1: Predicted results (solid circles) versus experimental results (open circles) for probabilistic crowding.

approach to analyzing niching GAs can now be carried over to probabilistic crowding.

In order to validate the theoretical developments so far, we perform experiments in the next two sections.

3.4 Experiments Using Idealized Operators

The purposes of these experiments are: (i) Check that the deterministic difference equation analysis models the stochastic situation well; (ii) Check that the approach of picking a candidate from each niche is reasonable in the analysis. In order to achieve these goals, we use quite large population sizes in the following.

The experiments in this first section are done using a fitness function with only q discrete niches each of size one, mutation probability p_l idealized as uniform change to one of the other niches and the probabilistic crowding acceptance rule to choose the winner.

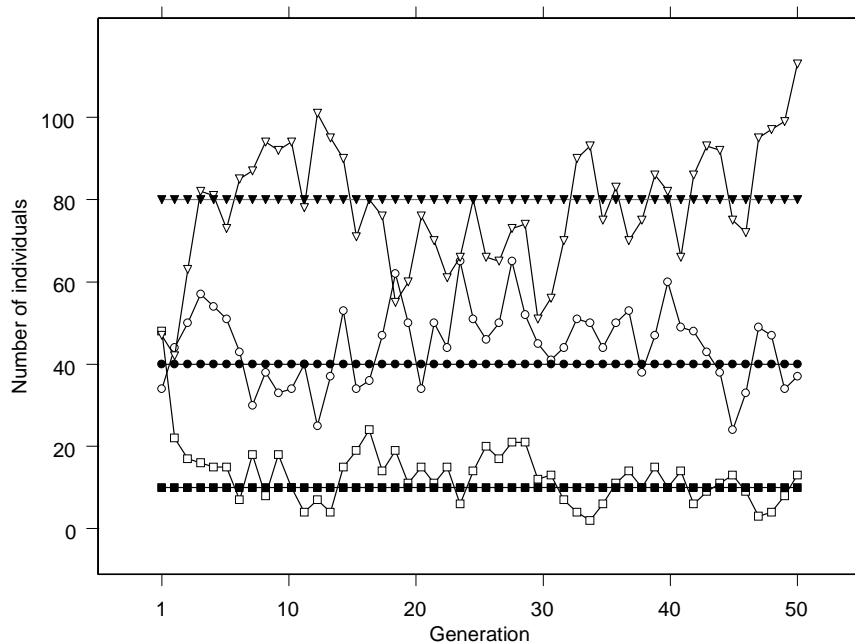


Figure 3.2: Predicted (solid circles) and experimental (open circles) results for niches \mathbb{X}_1 , \mathbb{X}_4 , and \mathbb{X}_8 . Predicted is steady-state expected proportion of population; experimental is measured proportion starting at generation one.

3.4.1 Two Niches, Same Jump Probabilities

We use Equation 3.9, here with $f(\mathbf{x}) = 1, f(\mathbf{y}) = 4$. This gives

$$X(t) = \frac{1}{5} + \left(\frac{1}{2} - \frac{1}{5}\right) p_s^t = \frac{1}{5} + \frac{3}{10} p_s^t$$

and

$$Y(t) = \frac{4}{5} + \left(\frac{1}{2} - \frac{4}{5}\right) p_s^t = \frac{4}{5} - \frac{3}{10} p_s^t.$$

We let $q = 2, p_s = 0.8$, use population size 100, and let the GA run for 50 generations. A plot of experimental versus predicted results for niche \mathbb{X} is provided in Figure 3.1. A ‘mutation’ probability $p_l = 1 - p_s = 0.2$ might seem high, but recall that this operation gives jumps between niches, and is not the usual bit-wise mutation. In the figure, we notice that the experimental results follow the prediction very well. There is some noise, but this is as expected, since a probabilistic acceptance

rule is used.

3.4.2 Multiple Niches, Same Jump Probabilities

Consider the function $f(x) = x$, where x is an integer between 1 and 8 inclusive, so $q = 8$. Here we can use Equation 3.15 with $\rho_{ji} = 1$, giving

$$X_i = \frac{f(x_i)}{\sum_{i=1}^q f(x_i)}, \quad (3.16)$$

with for example $X_1 = 1/36$, $X_4 = 4/36$, and $X_8 = 8/36$.

A plot of experimental versus predicted results for $p_s = 0.8$ is provided in Figure 3.2. A population size of $n = 360$ is used, and the GA is run for 50 generations. With the proportions just mentioned for X_1 , X_4 , and X_8 , we get predicted population sizes $nX_1 = 10$, $nX_4 = 40$, and $nX_8 = 80$. Again, we notice that the empirical results follow the predicted results very well, although there is a certain level of noise also in this case, as expected.

An analysis of the amount of noise can be performed as follows. Consider the GA's operation as n Bernoulli trials where the probability of picking from the i -th niche is given by Equation 3.16, so $p_i = X_i$. This gives a binomial distribution, where $\mu_i = np_i$, $\sigma^2 = np_i(1 - p_i)$. For niche \mathbb{X}_1 we get for example

$$\sigma^2 = 360 \frac{1}{36} \frac{35}{36},$$

which gives $\sigma \approx 3.1$, and similarly for \mathbb{X}_4 and \mathbb{X}_8 we get $\sigma \approx 6.0$ and $\sigma \approx 7.9$ respectively. The fact that the noise increases with the fitness of a niche, as seen in Figure 3.2, is therefore not surprising.

3.5 Experiments Using Traditional Operators

In this section, we introduce traditional mutation and crossover operators into the probabilistic crowding experiments. A population size of $n = 200$, and 100 generations is used. Two probabilistic crowding variants are investigated, variant M and variant M&C. For the variant M, mutation probability 0.1 is used; for the variant M&C, crossover probability 0.6 and mutation probability

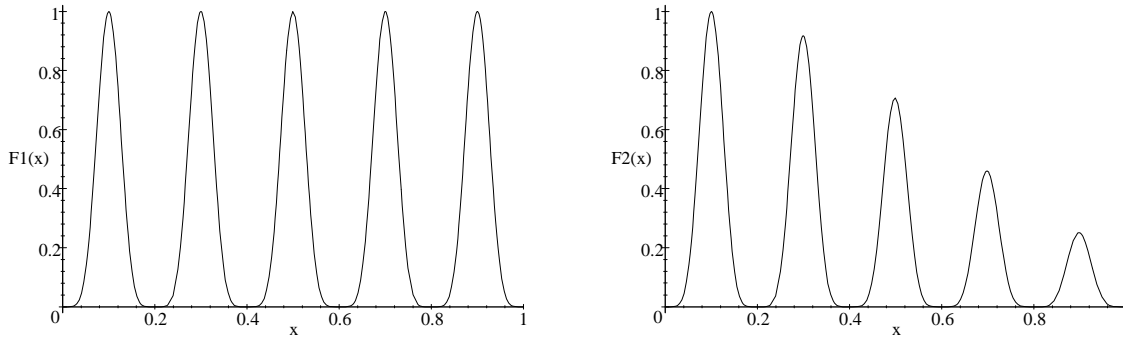


Figure 3.3: The test functions F1 and F2. F1 is to the left, F2 to the right.

0.3 is used.

Table 3.2 shows results from the experiments. Results are presented for the F1 and F2 test functions [Goldberg and Richardson, 1987], where

$$\begin{aligned}
 F1(x) &= \sin^6(5\pi x) \\
 F2(x) &= e^{-2(\ln 2)\left(\frac{x-0.1}{0.8}\right)^2} \sin^6(5\pi x).
 \end{aligned}$$

These two functions are plotted in Figure 3.3.

The question is whether one should focus on niche allocation, peak allocation, or a combination of both. The last alternative is chosen here, since the strength of probabilistic crowding is how individuals are evenly allocated. According to this, the functions have been split up into 25 equally-sized intervals on the X -axis. These intervals are presented as rows in Table 3.2, and as bars in Figure 3.4. The columns present results for test functions F1 and F2. For each test function, predicted allocation and experimental allocation is shown. Predicted allocation is computed by discretization of the test function in question, then integrating over an interval $[a, b]$, and finally normalizing by dividing by the integral of the function over $[0, 1]$. Experimental allocation is merely the allocation of individuals to the interval $[a, b]$ divided by the total number of individuals allocated to the interval $[0, 1]$.

The main result in Table 3.2 is that for both F1 and F2, the probabilistic crowding variants M and M+C give allocation of trials close to that predicted. This confirms that Equation 3.15 (and its special cases as presented) can be applied also when classical GA operators are used, at least

Interval of function domain	F1			F2		
	P	M	M+C	P	M	M+C
0.00 – 0.04	0.00083	0.0027	0.0019	0.0012	0.0016	0.0019
0.04 – 0.08	0.041	0.027	0.032	0.061	0.062	0.078
0.08 – 0.12	0.12	0.14	0.11	0.17	0.26	0.34
0.12 – 0.16	0.041	0.021	0.025	0.061	0.055	0.047
0.16 – 0.20	0.00083	0.0014	0.0014	0.0012	0.0017	0.0022
0.20 – 0.24	0.00083	0.0011	0.0026	0.0012	0.0020	0.0023
0.24 – 0.28	0.041	0.021	0.024	0.058	0.043	0.049
0.28 – 0.32	0.12	0.16	0.17	0.16	0.23	0.19
0.32 – 0.36	0.041	0.025	0.029	0.055	0.046	0.042
0.36 – 0.40	0.00083	0.0019	0.0024	0.0011	0.0012	0.0019
0.40 – 0.44	0.00083	0.0016	0.0018	0.00097	0.0012	0.0018
0.44 – 0.48	0.041	0.024	0.029	0.046	0.032	0.030
0.48 – 0.52	0.12	0.13	0.10	0.12	0.11	0.088
0.52 – 0.56	0.041	0.029	0.029	0.041	0.026	0.028
0.56 – 0.60	0.00083	0.0018	0.0017	0.00078	0.0019	0.0026
0.60 – 0.64	0.00083	0.0015	0.0024	0.00067	0.0016	0.0015
0.64 – 0.68	0.041	0.024	0.025	0.031	0.018	0.015
0.68 – 0.72	0.12	0.18	0.16	0.080	0.043	0.033
0.72 – 0.76	0.041	0.024	0.027	0.026	0.018	0.015
0.76 – 0.80	0.00083	0.0020	0.0032	0.00048	0.0010	0.00055
0.80 – 0.84	0.00083	0.0014	0.0030	0.00039	0.0011	0.0011
0.84 – 0.88	0.041	0.026	0.028	0.017	0.012	0.011
0.88 – 0.92	0.12	0.14	0.15	0.044	0.024	0.0096
0.92 – 0.96	0.041	0.027	0.032	0.014	0.0086	0.0066
0.96 – 1.00	0.00083	0.0015	0.0024	0.00025	0.00055	0.0014

Table 3.2: Experimental results for probabilistic crowding on the F1 and F2 functions, aggregated over all generations. The ‘P’ columns show predicted allocations, while the ‘M’ and ‘M+C’ columns show actual allocations for probabilistic crowding with mutation only and with mutation and crossover respectively.

for a certain class of fitness functions. There are two qualifications here. First, note that there is a small ‘smoothing’ effect for the intervals of lowest fitness, for example the interval 0.00 – 0.04. This is mostly due to allocations in early generations, when individuals are distributed more uniformly, according to the uniform random initialization. Second, there is a slightly higher allocation than predicted to intervals of high fitness, such as the intervals 0.08 – 0.12 and 0.28 – 0.32. It is currently not clear what the reason is for this slightly higher allocation.

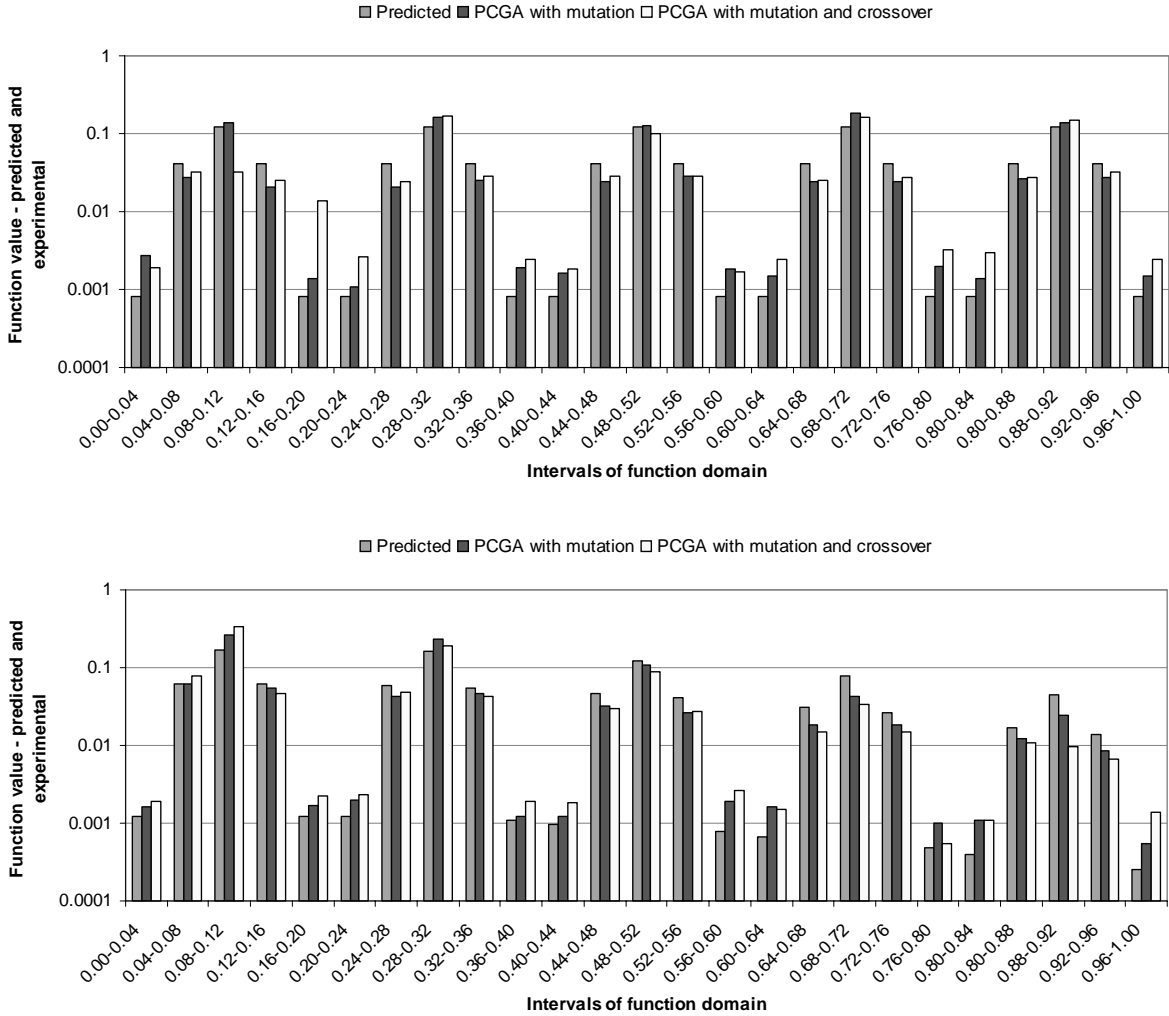


Figure 3.4: Predicted versus actual allocation of individuals to niches for the test functions F1 and F2. Results for F1 are shown at the top, results for F2 are shown at the bottom.

3.6 Population Sizing

To derive population sizing equation results for probabilistic crowding, we are going to exploit the fact that the approach gives an equilibrium. Then, at equilibrium, we require that with high probability, we shall find an individual of the desired fitness in the population. Typically, the desired fitness is the fitness of an individual of high fitness, such as the best fit or more generally the k -th best fit. The desired fitness might be known in advance, or it might be estimated. Underestimating the desired fitness will give more conservative population sizing results.

In the following, we first introduce theoretical results, drawing on Mahfoud's research [Mah-

foud, 1995]. Second, we give examples of how these results can be used and provide experimental verification.

3.6.1 Population Sizing Theory

Consider an individual \mathbf{x}_i and its fitness $f(\mathbf{x}_i) = f_i$. The probability of having the \mathbf{x}_i individual at equilibrium is given by the equilibrium probability $\Pr(f_i)$. The probability of not having \mathbf{x}_i , or \mathbf{x}_i not occurring in the population, is therefore, for one individual

$$1 - \Pr(f_i).$$

When we consider a population of n individuals, the probability of not having \mathbf{x}_i is just the joint probability

$$(1 - \Pr(f_i))^n.$$

Now, consider \mathbf{x}_i as a niche representative, and suppose we want to identify c niches. The probability of one or more niches not occurring, X , is bounded as [Mahfoud, 1995]

$$X \leq \sum_{i=1}^c (1 - \Pr(f_i))^n,$$

and we can set

$$X \approx \sum_{i=1}^c (1 - \Pr(f_i))^n$$

as argued elsewhere [Mahfoud, 1995, p. 163].

The probability of having an individual of fitness f_i or better among n individuals is therefore

$$1 - X,$$

and in order to have an individual \mathbf{x}_i with fitness f_i or better, we require that this probability is

bounded from below

$$1 - X \geq \gamma.$$

From this, a population sizing equation for identifying c niches can be derived [Mahfoud, 1995, p. 175]

$$n = \left\lceil \frac{c}{r} \left[-\ln \left(\frac{1 - \gamma^{\frac{1}{g}}}{c} \right) \right] \right\rceil. \quad (3.17)$$

Here $r = f_{\min}/f_{\max}$ is the ratio between minimal and maximal of fitness optima in the c niches, g is the number of niches, and γ is the probability of maintaining all c niches.

Two slight complications related to the derivation above are acknowledged. First, the fitness ratio r might not be known. However, in many cases it should be easy to give estimates of r if not of f_{\min} and f_{\max} . Second, there is the question of niche loss. Equation 3.17 is based on considering selection alone. Here, the two only possible outcomes are that an existing niche is maintained, or an existing niche is lost. In order for a success to occur, it is required that the c niches are maintained for g generations, so not a single niche can be lost during these g generations. This is reflected in Equation 3.17 as follows. When the number of generations g increases, the expression $\gamma^{\frac{1}{g}}$ will get closer to one, and the required population size n will increase as a result. This reflects the fact that with selection only operating, niches can only be lost.

One could argue that the focus on loss only is too conservative, since niches may be lost, but they may also be gained or generated, due to the operation of crossover, mutation, or other generative operators. And when inspecting the population at the last generation, say, one is interested in whether a representative for a niche is there or not, and not whether it had been lost previously. It seems interesting to investigate both of these two approaches, in both cases using Equation 3.17. Using g with the actual number of generations run can be used to give a conservative population sizing estimate, while setting $g = 1$ gives a less conservative population sizing estimate. Both of these approaches are investigated in the following.

	Less conservative (set $g = 1$)		Conservative (set $g = 50$)	
Desired γ	Population size	Observed γ_o	Population size	Observed γ_o
0.80	11	0.74	27	1.0
0.95	17	0.93	32	1.0

Table 3.3: Population size predicted from desired reliability, along with observed reliability.

3.6.2 Population Sizing Experiments

As an example, consider the fitness function used in Section 3.4.2. Suppose we want to reliably maintain the three best fit niches \mathbb{X}_6 , \mathbb{X}_7 , and \mathbb{X}_8 . For population sizing purposes we need to consider all of these niches.

We use Equation 3.17 for population sizing as follows: Given known parameters c , r , γ , and g , we compute the population size n . Two different approaches are used to setting g . We either set $g = 1$ or $g = 50$, since 50 generations are used in the experiments here. The first setting $g = 1$ corresponds to essentially ignoring the effect of niche loss over generations. The second setting $g = 50$ is more conservative, and takes niche loss into account as discussed above. So for the conservative population size we get

$$n = \left\lceil \frac{3}{\frac{6}{8}} \left[-\ln \left(\frac{1 - 0.8^{\frac{1}{50}}}{3} \right) \right] \right\rceil = 27,$$

while the less conservative population size is

$$n = \left\lceil \frac{3}{\frac{6}{8}} \left[-\ln \left(\frac{1 - 0.8}{3} \right) \right] \right\rceil = 11.$$

Observed γ_o is computed as follows. For desired γ , r_e experimental runs were performed. Runs in which the top c niches did not have a representative were counted, resulting in a value for failure runs r_f . These are runs where, at the last generation, at least one of the \mathbb{X}_6 , \mathbb{X}_7 , or \mathbb{X}_7 niches does not have a representative. Finally, the observed γ_o was computed as $\gamma_o = (r_e - r_f)/r_e$. In the experiments reported here, $r_e = 100$ was used.

In Table 3.3, the results from using the population sizing equation (3.17) are summarized. We see that the less conservative population sizing approach is in closer correspondence to the empirical data than the more conservative population sizing approach. This shows that the assumption of

niche loss, at least for this particular test function, might be overly conservative.

3.7 Conclusion and Future Work

A novel niching algorithm, probabilistic crowding, has been introduced. Probabilistic crowding has been shown theoretically as well as empirically to give reliable niching according to a novel, general niching rule, a rule which generalizes the niching rule known from previous research. The two core ideas in probabilistic crowding are (i) to hold tournaments between similar individuals, and (ii) to let tournaments be probabilistic. These two principles leads to a niching algorithm which is simple, predictable, and fast. Probabilistic crowding also does not have any additional parameters compared to classical GAs, which is often an advantage. The algorithm belongs to a family of algorithms, integrated tournament algorithms, which are also defined and presented in this chapter.

Future work includes the following. First, experiments on harder fitness functions, such as complex Bayesian networks, would be interesting. Second, the relationship to Metropolis and simulated annealing could be investigated further, in particular the use of temperature. A third issue is population sizing. Fourth, the approach of holding a probabilistic tournament could be extended to include the distance function and possibly also the mating step.

Chapter 4

Deceptive Bayesian Networks

One way to systematically vary the hardness of BNs is to define a mapping from the deceptive problems used in the GA literature to BNs, and then vary the hardness of the deceptive problems, giving BNs with varying hardness as well.

The notion of deception was introduced in order to systematically investigate the conditions under which GA schema processing may lead a GA away from fitness optima [Goldberg, 1987]. Functions of unitation were later discussed in connection with deceptive functions. A function of unitation (or bit-counting function) is defined over a bit string. It depends on the number of ones in the bit string, and not their positions. There are several advantages to such artificial functions, notably their ease of specification and analysis [Goldberg, 1992]. Unfortunately, they are not used in applications. In contrast, Bayesian networks (BNs) have proven to be an important knowledge representation formalism, for instance for probabilistic reasoning in expert systems [Pearl, 1988] [Neapolitan, 1990]. However, it is not clear a priori whether BNs can be challenging problems for GAs or not, and in particular whether they can be deceptive. And if BNs can be deceptive, is there a way to characterize the conditions under which they are deceptive? The advantage of translating into BNs would be that BNs appear to be more expressive, so one can create BNs that are deceptive or ‘close’ to being deceptive, but which are not possible to express as functions as unitation.

The current research is motivated by previous and ongoing research on using GAs for BN inference [Rojas-Guzman and Kramer, 1993] [Rojas-Guzman and Kramer, 1996] [Welch, 1996] [Mengshoel, 1997] [Mengshoel and Wilkins, 1998b] [Mengshoel and Wilkins, 1998a]. We believe that this line of research can benefit from an increased focus on the relationship between a BN and the joint probability distribution defined by the BN, which is the GA’s fitness function. Here we

focus on characteristics of a Bayesian network, given a certain joint probability distribution, which is derived from a function of unitation. There has been some research within the BN community on typical properties of the joint distribution defined by a BN [Druzdzel, 1994], but little research on varying the hardness of constructed networks, which is what we do here.

This chapter introduces an approach to mapping functions of unitation into Bayesian networks. In addition, instances from three classes of functions of unitation are mapped to Bayesian networks: onemax functions, trap functions, and hill functions. Onemax functions of unitation are non-deceptive, and the constructed onemax BNs are easy as well. Trap functions of unitation, on the other hand, are deceptive. The trap function instances mapped into highly connected, hard BNs. Hill functions of unitation are similar to trap functions of unitation, but they are not deceptive. The instances considered were still mapped to highly connected BNs. These results show that BNs can be deceptive, and suggest that deceptive functions of unitation are indeed hard, not only for GAs, and that other functions of unitation can be hard as well, although less so than the deceptive ones. In addition, this research paves the way for controlled and systematic experimentation, since experimental BNs of varying difficulty can be constructed using the mapping. An additional advantage of constructing a BN from a function of unitation is that the latter is a parametric function while the former is non-parametric.

The rest of this chapter is structured as follows. Section 4.1 presents schemata, deception, and functions of unitation. Section 4.2 describes how to construct BNs from functions of unitation, and the following sections show how BNs can be constructed from different classes of functions of unitation. Section 4.3 constructs BNs from onemax functions of unitation, which are non-deceptive functions. Section 4.4 considers trap functions of unitation, which are deceptive functions. Section 4.5 discusses hill-shaped functions of unitation. Section 4.6 parametrizes the BNs, while Section 4.7 concludes and presents future work. Other approaches to constructing hard BNs are discussed in Chapter 6.

4.1 Schemata, Deception, and Functions of Unitation

The notion of GA deception is based on the assumption that schema processing plays a central role in GAs. In the following we introduce some definitions that are helpful in discussing static schema

processing.

Definition 8 Let $p : \{0, 1\}^n \rightarrow R$ be a fitness function, and let $B = b_1 \dots b_n$ and $S = s_1 \dots s_n$ with $b_i \in \{0, 1\}$ and $s_i \in \{0, 1, *\}$. Now define $inst(S) = inst(s_1 \dots s_n)$ to refer to an instantiation of the schema S . An instantiation B of S ($B = inst(S)$) is such that if $s_i = *$, then $b_i = 0$ or $b_i = 1$. If $s_i \neq *$, then $b_i = s_i$. Now the set of instantiations of S , $Inst(S)$, can be defined as follows:

$$Inst(S) = \{B \mid B = inst(S)\}.$$

Given the definition of an instantiation of a string, the notion of schema fitness can be introduced.

Definition 9 (Schema fitness) The schema fitness $q : \{0, 1, *\}^n \rightarrow R$ of schema $S = \{0, 1, *\}^n$ is defined as follows:

$$q(S) = \sum_{B \in Inst(S)} p(B).$$

When there is no ambiguity, we write $p(S)$ instead of $q(S)$.

As an example of the above definition,

$$q(*1*) = p(010) + p(110) + p(011) + p(111).$$

One aspect of GAs that we will focus on is their optimizing capability. For simplicity we assume in the following that there is one optimal string—this can easily be generalized.

Definition 10 Let f be a fitness function, and let $X_{OPT} = \arg \max_{X \in \{0, 1\}^n} f(X)$ be an optimal string. Then any schema S such that $inst(S) = X_{OPT}$ is called an optimal schema.

Schema processing operates on schema partitions [Grefenstette, 1993] [Mitchell, 1996]. In the 3-bit case, the schema partitions for $(d**)$ are $(0**)$ and $(1**)$. Competition between bit strings takes place within a schema partition. It is a common assumption within the GA community that GAs start by processing low-order schema partitions, and then gradually shift towards processing high-order schema-partitions.

Schema partition	Conditions for full deception
$p(d **)$	$p(0 **) > p(1 **)$
$p(*d*)$	$p(*0*) > p(*1*)$
$p(**d)$	$p(**0) > p(**1)$
$p(dd*)$	$p(00*) > p(01*), p(00*) > p(10*), p(00*) > p(11*)$
$p(d*d)$	$p(0*0) > p(0*1), p(0*0) > p(1*0), p(0*0) > p(1*1)$
$p(*dd)$	$p(*00) > p(*01), p(*00) > p(*10), p(*00) > p(*11)$

Table 4.1: Conditions for full deception in a bit string of length three.

Definition 11 *A schema partition is deceptive if there exists a schema with higher fitness than the optimal schema in the schema partition. A fitness function is deceptive if it contains a deceptive schema partition. A fitness function is fully deceptive if all schema partitions (except for the maximally specified one) are deceptive. A fitness function is non-deceptive if no schema partition is deceptive.*

As an example, consider a fully deceptive 3-bit string, and assume without loss of generality that 111 is the global maximum, 000 the deceptive maximum. This is formalized in the optimality condition

$$p(111) > p(000) \tag{4.1}$$

and the deceptive conditions shown in Table 4.1. Note that in this case, the deceptive schemata are all instances of the same string, namely the deceptive maximum. According to Definition 11, this need not be the case.

There has been extensive research on deception within the GA community. The connection between GAs and Walsh functions has been presented by Goldberg [Goldberg, 1989b] [Goldberg, 1989a]. He emphasizes the utility of using Walsh functions and Walsh polynomials for analyzing GAs [Goldberg, 1989b]. These methods are extended to analyze deception and disruption to schema processing by genetic operators. In addition, the first fully deceptive function—of bit length three—is presented [Goldberg, 1989a].

The notion of unitation has been used repeatedly for the purpose of analysis of hard and easy fitness functions.

Definition 12 Let B be a bit string of length n . The unitation $u(B)$ of B is a function defined as

$$u(B) = u(b_1 \dots b_n) = b_1 + \dots + b_n = \sum_{i=1}^n b_i.$$

A function of unitation is defined using $u(B)$.

In other words, unitation means the number of ones in a bit string, so $u(100) = 1$ and $u(111) = 3$.

Research on deception has also been critiqued. For example, Bäck says that ‘it may well be the case that [deceptive functions] are the problems for which Genetic Algorithms *must* fail according to the construction of the problem and the provable limits of Genetic Algorithms’ [Bäck, 1996, p. 128]. The results in this chapter, in particular Section 4.4, do indeed suggest that deceptive functions are inherently difficult, not only for GAs but for other algorithms as well. It should, however, be emphasized that we only give examples and not formal proofs of this.

4.2 Constructing Bayesian Networks

Two factors of a BN make it harder: A highly connected topology and highly skewed probability distributions. Concerning the topology, polynomial time algorithms exist for BNs that are trees (also known as singly connected networks) [Pearl, 1988]. Concerning skewness, the situation is not as clear. However, it is known that skewed distributions can make stochastic simulation converge much more slowly, in particular when low-probability states are instantiated.

Both functions of unitation and BNs can be used as GA fitness functions; the advantage of functions of unitation compared to BNs is that there are fewer parameters to manipulate. In particular, the number of conditional probabilities that needs to be supplied for a node in a BN grows exponentially with the number of parents that the node has. In contrast, the number of parameters in a function of unitation grows at most linearly with the number of bits.

This is how to systematically create a Bayesian network from a fitness function:

1. Use the fitness function to create a joint probability distribution. For example, if we want a fully deceptive BN consisting of k nodes, a fully deceptive trap function (see Section 4.4) consisting of k bits is first constructed. This deceptive trap function is then used to construct a deceptive joint probability distribution. This is a table consisting of 2^k rows, one row per explanation in the

Bayesian network.

2. Use the joint probability distribution to construct a Bayesian network that represents it, but in a more compact manner. Possibly, some approximation needs to be taken here, but the essential characteristics (such as deception) need to be maintained. Assuming binary nodes, this creates a network with k nodes (and some number of edges) from the 2^k table.

To prepare for a more formal description of the above, the following notation is introduced:

- $cc(i)$: cell count for cell i . Cell count is the number of samples that exist for an explanation. For example, with four samples for the explanation $X_1 = 0, X_2 = 1, X_3 = 1$, the cell count is $cc(011) = 4$. Often, cells are indexed by a decimal number rather than a binary number, giving for example, $cc(3) = 4$.
- f : fitness function (in the following a function of unitation)
- $\text{Bin}(i)$: function that gives the integer i in binary
- k : number of bits in fitness function
- $\text{Round}(x)$: rounds the number x to the nearest integer
- c : constant; in the following $c = 3$ is used
- $\text{Build}()$: constructs a pattern \mathbf{P} , further described below
- $\text{Direct}()$: constructs edges from the pattern \mathbf{P} , further described below
- $\text{Estimate}()$: estimates conditional probabilities, further described below

The Construct algorithm for constructing a k -node BN from a k -bit fitness function, such as a k -bit trap function, is as follows:

1. Let there be k nodes $\{X_1, X_2, \dots, X_k\}$. Let each node have two states $\{0, 1\}$, i.e. $|\Omega_{X_i}| = 2$ for all i .
2. For $i = 0$ to $2^k - 1$: $cc(i) \leftarrow \text{Round}(\exp(f(\text{Bin}(i)) + c))$
3. $\mathbf{P} \leftarrow \text{Build}((X_1, X_2, \dots, X_k), (|\Omega_{X_1}|, |\Omega_{X_2}|, \dots, |\Omega_{X_k}|), cc)$

4. $\mathbf{E} \leftarrow \text{Direct}(\mathbf{P})$

5. $\mathbf{Pr} \leftarrow \text{Estimate}((X_1, X_2, \dots, X_k), (|\Omega_{X_1}|, |\Omega_{X_2}|, \dots, |\Omega_{X_k}|), cc, \mathbf{E})$

The above algorithm constructs a DAG (\mathbf{V}, \mathbf{E}) and conditional probability tables \mathbf{Pr} which together make up a Bayesian network. Examples of how Construct works are given in sections 4.3, 4.4, and 4.5.

The Construct algorithm is based on the observation that a BN can be *induced* from the joint distribution table; this is learning with a complete data set. Approaches for machine learning of BNs can therefore be utilized, here TETRAD [Scheines et al., 1994] is used. The algorithms Build and Estimate both correspond to TETRAD procedures (see [Scheines et al., 1994]), while Direct is a quite simple algorithm. Given nodes (X_1, X_2, \dots, X_k) along with their cardinalities $(|\Omega_{X_1}|, |\Omega_{X_2}|, \dots, |\Omega_{X_k}|)$ and a cell count cc , Build constructs a set of patterns \mathbf{P} which expresses conditional independence and dependence relations between the random variables. Direct takes the patterns \mathbf{P} and creates a set of edges \mathbf{E} . Estimate takes the same parameters as Build and the edges \mathbf{E} , and constructs conditional probability tables \mathbf{Pr} . Build and Estimate were used with their default parameter settings when the results in the following three sections were produced.

Why is the exponent \exp taken above? This is because BNs are set up to work by multiplication of probabilities, and this is ‘counteracted’ by taking the exponent here. More formally, an exponential function of unitation can be written as follows, where the function of unitation $u(b_1 \dots b_n) = b_1 + \dots + b_n$ is assumed:

$$\begin{aligned} e^{u(B)+c} &= e^{u(b_1 \dots b_n)+c} \\ &= e^{b_1 + \dots + b_n + c} \\ &= e^{\sum_{i=1}^n b_i + c} \\ &= e^c \prod_{i=1}^n e^{b_i}. \end{aligned}$$

Note that the last line above is of the same form as Equation 1.1. This justifies the use of $e^{u(B)+c}$ in the Construct algorithm above.

i	$B = \text{Bin}(i)$	$u(B)$	$f(B)$	$f(B) + c$	$cc(i)$
0	000	0	0	3	20
1	001	1	1	4	55
2	010	1	1	4	55
3	011	2	2	5	148
4	100	1	1	4	55
5	101	2	2	5	148
6	110	2	2	5	148
7	111	3	3	6	403

Table 4.2: Non-deceptive onemax fitness function defined over bitstring consisting of three bits.

4.3 Onemax Functions

Onemax functions are generalizations of the unitation $u(B)$ of a bit string B . Ackley introduced the onemax function

$$f(B) = 10 \times u(B),$$

i.e. just a constant multiple of the function of unitation [Ackley, 1987]. More generally, a onemax function of unitation is

$$f(B) = d \times u(B), \tag{4.2}$$

where d is a constant. By letting $d = 1$ in Equation 4.2, we get the onemax function $f(B) = u(B)$, which is used in the following.

Table 4.2 shows bit strings and corresponding cell counts for this onemax function. The conditional probability tables created by the Construct algorithm based on the input in Table 4.2 is shown in Table 4.3, the graph is shown in Figure 4.1. It is easy to check that the BN constructed from the onemax function is non-deceptive, like the onemax function itself. Using a four bit onemax function in a similar way as above, the BN whose conditional probabilities are shown in Table 4.4 is attained.

As one might have expected, the BNs constructed from the two onemax functions are quite ‘easy’. There are no interactions between nodes in the BNs, and the conditional probability tables are the same within any one BN node

Pr(X_1)	
0	0.269
1	0.731

Pr(X_2)	
0	0.269
1	0.731

Pr(X_3)	
0	0.269
1	0.731

Table 4.3: Conditional probability tables of non-deceptive BN constructed from non-deceptive onemax fitness function.

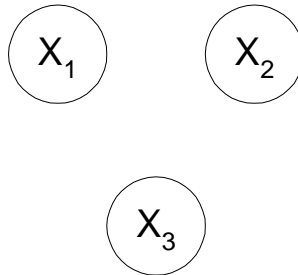


Figure 4.1: Graph BN constructed from non-deceptive onemax fitness function.

Pr(X_1)	
0	0.269
1	0.731

Pr(X_2)	
0	0.269
1	0.731

Pr(X_3)	
0	0.269
1	0.731

Pr(X_4)	
0	0.269
1	0.731

Table 4.4: Conditional probability tables of non-deceptive BN constructed from non-deceptive onemax fitness function with four bits.

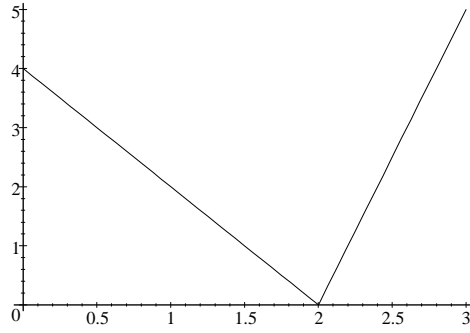


Figure 4.2: Fully deceptive function over three bits..

4.4 Trap Functions

This section focuses on deception in BNs. Deceptive BNs are constructed from trap functions, which are deceptive functions of unitation. The advantage of trap functions is that they have few parameters but can still be made deceptive.

Ackley used a trap function for empirical investigations [Ackley, 1987], and this function has later been generalized [Deb and Goldberg, 1993]. The trap function definition of Deb and Goldberg is the following:

$$f(B) = \begin{cases} \frac{a}{z}(z - u(B)) & \text{if } u(B) \leq z \\ \frac{b}{k-z}(u(B) - z) & \text{otherwise} \end{cases}, \quad (4.3)$$

where a is the local (deceptive) optimum, b is the global optimum, and z is the slope-change location. The point of a trap function is that there are two optima, a and b , and by varying the parameters, one can make it more or less difficult to find the global optimum b as opposed to the local optimum a . It has been shown that when the following condition holds, a trap function is fully deceptive [Deb and Goldberg, 1993, p. 99]:

$$\frac{a}{b} = r \geq \frac{2 - 1/(k - z)}{2 - 1/z}, \quad (4.4)$$

The smallest bit string that can be fully deceptive consists of three bits [Goldberg, 1987]. In

i	$B = \text{Bin}(i)$	$u(B)$	$f(B)$	$f(B) + c$	$cc(i)$
0	000	0	4	7	1097
1	001	1	2	5	148
2	010	1	2	5	148
3	011	2	0	3	20
4	100	1	2	5	148
5	101	2	0	3	20
6	110	2	0	3	20
7	111	3	5	8	2981

Table 4.5: Deceptive trap fitness function defined over bit string consisting of three bits.

the following, we construct a 3-node BN that is fully deceptive. The construction is based on the trap function in Equation 4.3. Let $k = 3$, $z = 2$, $a = 4$, and $b = 5$ in that equation, so we get the function

$$f(B) = \begin{cases} 2(2 - u(B)) & \text{if } u(B) \leq 2 \\ 5(u(B) - 2) & \text{if } u(B) > 2 \end{cases}$$

which can also be written

$$f(X) = \begin{cases} 2(2 - X) & \text{if } X \leq 2 \\ 5(X - 2) & \text{if } X > 2 \end{cases}$$

and is shown in Figure 4.2. We note that the condition for full deception of Equation 4.4 is met:

$$\frac{4}{5} = r \geq \frac{2 - 1/(3 - 2)}{2 - 1/2} = \frac{2}{3}.$$

Now the following trap function values can be calculated: $f(000) = 4$, $f(001) = 2$, $f(011) = 0$, and $f(111) = 5$. This gives bit string fitness as shown in Table 4.5. Schemata fitness values can be computed from Table 4.5, and it can easily be checked that this fitness function is indeed fully deceptive.

The BN whose graph is shown in Figure 4.3 and whose conditional probability tables are shown in Table 4.6 is constructed from the fitness function f shown in Table 4.5. In order to make this BN deceptive, the $\text{Pr}(Y_1)$ table is replaced with a new table $\text{Pr}(Y'_1)$ as follows. Since 111 is the

Pr(Y ₁)	
0	0.3084
1	0.6916

Pr(Y ₂ Y ₁)		
Y ₁ :	0	1
0	0.8811	0.0530
1	0.1189	0.9470

Pr(Y ₃ Y ₁ , Y ₂)				
Y ₁ :	0		1	
Y ₂ :	0	1	0	1
0	0.8811	0.8809	0.8809	0.0067
1	0.1189	0.1190	0.1190	0.9933

Table 4.6: Conditional probability tables of BN constructed from deceptive trap fitness function, before normalization.

Pr(Y ₁)	
0	0.54
1	0.46

Pr(Y ₂ Y ₁)		
Y ₁ :	0	1
0	0.8811	0.0530
1	0.1189	0.9470

Pr(Y ₃ Y ₁ , Y ₂)				
Y ₁ :	0		1	
Y ₂ :	0	1	0	1
0	0.8811	0.8809	0.8809	0.0067
1	0.1189	0.1190	0.1190	0.9933

Table 4.7: Conditional probability tables of BN constructed from deceptive trap fitness function, after normalization.

global optimum and 000 the deceptive optimum, the condition

$$\frac{\Pr(111)}{\Pr(000)} > 1$$

needs to be maintained. After some calculation we get $\Pr(Y_1 = 0) < 0.5479$. It turns out that for example $\Pr(Y'_1 = 0) = 0.54$ and $\Pr(Y'_1 = 1) = 0.46$ gives a fully deceptive BN when the two other tables in Table 4.6 are kept the same. Table 4.7 gives the CPTs after this normalization has been done. Table 4.8 gives the joint distribution defined by this BN.

What happens if we consider a four-bit trap function? Let $k = 4$, $z = 3$, $a = 4$, and $b = 5$ in

Y_1	Y_2	Y_3	$\Pr(Y_1, Y_2, Y_3)$
0	0	0	$0.54 \times 0.8811 \times 0.8811 = 0.4192$
0	0	1	$0.54 \times 0.8811 \times 0.1189 = 0.0566$
0	1	0	$0.54 \times 0.1189 \times 0.8809 = 0.0566$
0	1	1	$0.54 \times 0.1189 \times 0.1190 = 0.0076$
1	0	0	$0.46 \times 0.0530 \times 0.8809 = 0.0215$
1	0	1	$0.46 \times 0.0530 \times 0.1190 = 0.0029$
1	1	0	$0.46 \times 0.9470 \times 0.0067 = 0.0029$
1	1	1	$0.46 \times 0.9470 \times 0.9933 = 0.4327$

Table 4.8: Joint probability of deceptive BN.

$\Pr(Y_1)$		$\Pr(Y_2 Y_1)$			$\Pr(Y_3 Y_1, Y_2)$				
0	0.3834	$Y_1:$	0	1	$Y_1:$	0		1	
1	0.6166	0	0.7916	0.1296	$Y_2:$	0	1	0	1
		1	0.2084	0.8704	0	0.7915	0.7917	0.7917	0.0310
					1	0.2084	0.2082	0.2082	0.9690

$\Pr(Y_4 Y_1, Y_2, Y_3)$								
$Y_1:$	0				1			
$Y_2:$	0		1		0		1	
$Y_3:$	0	1	0	1	0	1	0	1
0	0.7915	0.7918	0.7918	0.7917	0.7918	0.7917	0.7917	0.0067
1	0.2085	0.2082	0.2082	0.2083	0.2082	0.2083	0.2083	0.9933

Table 4.9: Conditional probability tables of BN constructed from deceptive trap fitness function with four bits.

Equation 4.3, giving the function

$$f(B) = \begin{cases} \frac{4}{3}(3 - u(B)) & \text{if } u(B) \leq 3 \\ 5(u(B) - 3) & \text{if } u(B) > 3 \end{cases}.$$

The result of running Construct, as shown in Table 4.9, is quite similar to that for the three-bit trap function.

As one might have expected, the BNs constructed from the two trap functions turn out to be ‘harder’ than the BNs constructed from the two onemax functions. In particular, there are two things to notice. First, the graph in Figure 4.3 is fully connected, indicating interactions between all random variables. Second, the conditional probability tables in Table 4.6 and in Table 4.9 contain more varied and extreme entries than those for the onemax functions. In particular, $\Pr(Y_2 | Y_1 = 1)$

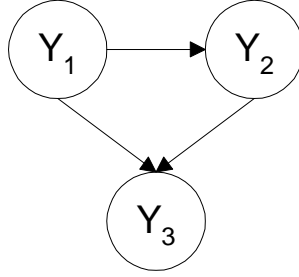


Figure 4.3: Graph of BN constructed from deceptive trap fitness function with three bits.

and $\Pr(Y_3 \mid Y_1 = 1, Y_2 = 1)$ are such extreme entries.

4.5 Hill Functions

Intuitively, onemax functions are ‘easy’ while trap functions are ‘hard’, and we have seen how the corresponding constructed BNs reflect this. But could it be the case that not only trap functions, but also other functions of unimodality, result in highly connected BNs? In order to investigate this question, we constructed a hill function by subtracting a trap function from its maximal value:

$$g(B) = b - f(B),$$

where f is a trap function, b is the trap function’s maximal value. For instance, for the 3-bit trap function presented in Section 4.4 we get

$$g(B) = \begin{cases} 1 + \frac{4}{3}u(B) & \text{if } u(B) \leq 3 \\ 5(4 - u(B)) & \text{if } u(B) > 3 \end{cases}.$$

Fitness values and cell counts for the 3-bit case are shown in Table 4.10. Somewhat surprisingly, the same graph was induced as for the corresponding trap function, shown in Figure 4.3. However, the conditional probability tables, shown in Table 4.11, are different. The most prominent differences are the following. First, $\Pr(Z_1)$ is slightly more skewed than $\Pr(Y_1)$. $\Pr(Z_2 \mid Z_1 = 1)$ is much less skewed than $\Pr(Y_2 \mid Y_1 = 1)$. $\Pr(Z_3 \mid Z_1 = 1, Z_2 = 1)$ is very similar to $\Pr(Y_3 \mid Y_1 = 1, Y_2 = 1)$, and the other conditional probability distributions are also quite similar to each other.

i	$B = \text{Bin}(i)$	$u(B)$	$f(B)$	$f(B) + c$	$cc(i)$
0	000	0	1	4	55
1	001	1	3	6	403
2	010	1	3	6	403
3	011	2	5	8	2981
4	100	1	3	6	403
5	101	2	5	8	2981
6	110	2	5	8	2981
7	111	3	0	3	20

Table 4.10: Hill fitness function defined over bit string consisting of three bits.

$\Pr(Z_1)$		$\Pr(Z_2 Z_1)$		
0	0.3757	$Y_1:$	0	1
1	0.6243	0	0.1192	0.5300
		1	0.8808	0.4700

$\Pr(Z_3 Z_1, Z_2)$				
$Z_1:$	0		1	
$Z_2:$	0	1	0	1
0	0.1201	0.1191	0.1191	0.9933
1	0.8799	0.8809	0.8809	0.0067

Table 4.11: Conditional probability tables of BN constructed from hill fitness function with three bits.

Pr(Y_1)	
0	q_1
1	$1 - q_1$

Pr(Y_2 Y_1)		
Y_1 :	0	1
0	q	q_2
1	$1 - q$	$1 - q_2$

Pr(Y_3 Y_1, Y_2)				
Y_1 :	0		1	
Y_2 :	0	1	0	1
0	q	q	q	q_3
1	$1 - q$	$1 - q$	$1 - q$	$1 - q_3$

Pr(Y_4 Y_1, Y_2, Y_3)								
Y_1 :	0				1			
Y_2 :	0		1		0		1	
Y_3 :	0	1	0	1	0	1	0	1
0	q	q	q	q	q	q	q	q_4
1	$1 - q$	$1 - q$	$1 - q$	$1 - q$	$1 - q$	$1 - q$	$1 - q$	$1 - q_4$

Table 4.12: Conditional probability tables of BN constructed from deceptive trap fitness function.

Similar to the 4-bit trap function, a 4-bit hill function was also investigated. The resulting conditional probability tables are omitted due to space restrictions. These tables show the same trend as the 3-bit tables, both by themselves and compared to the 4-bit trap BN. First, $\Pr(Z_1)$ is somewhat more skewed than $\Pr(Y_1)$. $\Pr(Z_2 | Z_1 = 1)$ is much less skewed than $\Pr(Y_2 | Y_1 = 1)$, and $\Pr(Z_3 | Z_1 = 1, Z_2 = 1)$ is much less skewed than $\Pr(Y_3 | Y_1 = 1, Y_2 = 1)$. Other conditional probability distributions are quite similar to each other.

4.6 General BNs from Deceptive Trap Functions

In this section, we investigate trap functions translated into BNs, in particular BNs with 4 nodes. However, the method should, with some extensions, be scalable to BNs of arbitrary size. In particular, we consider how to construct BNs based on trap functions without using TETRAD as was done earlier in this chapter.

4.6.1 4-node Trap BNs

The CPTs of the BN we focus on is presented in Table 4.12. Note that the conditional probabilities q_i are parameters; in an instantiated BN these would have values $q_i \in [0, 1]$ and obey the constraint that each column should sum to one. In the following, a shorthand notation is used, where $\Pr(Y_1 = y_1, Y_2 = y_2, Y_3 = y_3, Y_4 = y_4) = q_{y_1 y_2 y_3 y_4}$. Given the BN in Figure 4.12, the joint probabilities can be computed, resulting in joint distribution table with 2^4 entries. If we only consider the

unitation, we want several of the joint probabilities to be equal or close to equal, and therefore we pick representatives:

$$\begin{aligned}
 q_{0000} &= q_1 q q q \\
 q_{1000} &= (1 - q_1) q_2 q q \\
 q_{1100} &= (1 - q_1)(1 - q_2) q_3 q \\
 q_{1110} &= (1 - q_1)(1 - q_2)(1 - q_3) q_4 \\
 q_{1111} &= (1 - q_1)(1 - q_2)(1 - q_3)(1 - q_4)
 \end{aligned}$$

Here, q_{0000} , q_{1000} , q_{1100} , q_{1110} , and q_{1111} are given, and are used to compute values for the right-hand side.

There is a clear pattern here, and we might formulate the following algorithm: Consider the indices $y_1 \dots y_4$, and suppose we are currently at index y_i . As long as we have seen only ones so far, i.e. $y_j = 1$ for all $j \leq i$, we multiply by $(1 - q_i)$ for the i th index. For the first zero seen, or $y_j = 0$ and $y_k = 1$ for all $k < j$, we multiply by q_i . After the first zero, we multiply by q for a zero, $(1 - q)$ for a one.

All that remains now is to know the q_i 's and q . Note that by construction, $q_{0000}, q_{1000}, \dots, q_{1110}$ and q_{1111} are known, and so the q_i 's as well as q may be computed from them.

To get some more intuition for the structure of Table 4.12, we can compare it to the onemax function. Most columns in Table 4.12 are similar to the onemax function: If we pick a 0, we get q ; if we pick a 1 we get $(1 - q)$. The exceptions are for Y_1 and if all previous bits (or nodes) have been ones. As an example of the latter case, the global optimum may be attained, and it is attained with the probability $q_{1111} = (1 - q_1)(1 - q_2)(1 - q_3)(1 - q_4)$.

4.6.2 Solving the Equations for 4-node Trap BNs

We have five equations and five unknowns, so we can solve for the unknowns. Starting with the first equation, we can solve for q_1 , assuming that q is known:

$$q_1 = \frac{q_{0000}}{q^3} \tag{4.5}$$

Then we solve for q_2 and substitute for q_1 in the second equation:

$$q_2 = \frac{q_{1000}}{(1 - q_1)q^2} = \frac{q_{1000}}{\left(1 - \frac{q_{00000}}{q^3}\right)q^2} \quad (4.6)$$

Similarly, we have:

$$q_3 = \frac{q_{1100}}{(1 - q_1)(1 - q_2)q} = \frac{q_{1100}}{\left(1 - \frac{q_{00000}}{q^3}\right) \left(1 - \frac{q_{1000}}{\left(1 - \frac{q_{00000}}{q^3}\right)q^2}\right) q}, \quad (4.7)$$

$$\begin{aligned} q_4 &= \frac{q_{1110}}{(1 - q_1)(1 - q_2)(1 - q_3)} \\ &= \frac{q_{1110}}{\left(1 - \frac{q_{00000}}{q^3}\right) \left(1 - \frac{q_{1000}}{\left(1 - \frac{q_{00000}}{q^3}\right)q^2}\right) \left(1 - \frac{q_{1100}}{\left(1 - \frac{q_{00000}}{q^3}\right) \left(1 - \frac{q_{1000}}{\left(1 - \frac{q_{00000}}{q^3}\right)q^2}\right) q}\right)} \end{aligned} \quad (4.8)$$

and

$$\begin{aligned} q_{1111} &= (1 - q_1)(1 - q_2)(1 - q_3)(1 - q_4) \\ &= \left(1 - \frac{q_{00000}}{q^3}\right) \left(1 - \frac{q_{1000}}{\left(1 - \frac{q_{00000}}{q^3}\right)q^2}\right) \left(1 - \frac{q_{1100}}{\left(1 - \frac{q_{00000}}{q^3}\right) \left(1 - \frac{q_{1000}}{\left(1 - \frac{q_{00000}}{q^3}\right)q^2}\right) q}\right) \\ &\quad \times \left(1 - \frac{q_{1110}}{\left(1 - \frac{q_{00000}}{q^3}\right) \left(1 - \frac{q_{1000}}{\left(1 - \frac{q_{00000}}{q^3}\right)q^2}\right) \left(1 - \frac{q_{1100}}{\left(1 - \frac{q_{00000}}{q^3}\right) \left(1 - \frac{q_{1000}}{\left(1 - \frac{q_{00000}}{q^3}\right)q^2}\right) q}\right)}\right). \end{aligned} \quad (4.9)$$

By simplifying the last equation, we get:

$$q_{1111} = \frac{q^3 - q_{00000} - q_{1000}q - q_{1100}q^2 - q_{1110}q^3}{q^3}, \quad (4.10)$$

where we can solve for q . Once q is known, q_1, \dots, q_4 can easily be computed. This gives us CPT values to fill into Table 4.12.

4.6.3 Numerical Solutions

So far, we have not used numerical quantities. Now we turn to computing values for q_{0000} , q_{1000} , q_{1100} , q_{1110} , and q_{1111} . We consider a four-bit trap function with $k = 4$, $z = 3$, $a = 4$, and $b = 5$, giving the function

$$f(U) = \begin{cases} \frac{4}{3}(3 - U) & \text{if } U \leq 3 \\ 5(U - 3) & \text{if } U > 3 \end{cases},$$

where U is the unitation of a bit string. It is easy to check that these parameters ($a = 4, b = 5, z = 3,$ and $k = 4$) satisfy the deceptive condition:

$$\frac{a}{b} = \frac{4}{5} \geq \frac{2 - 1/(k - z)}{2 - 1/z} = \frac{3}{5}.$$

This gives $f(0) = 4, f(1) = \frac{8}{3}, f(2) = \frac{4}{3}, f(3) = 0, f(4) = 5$, and the total sum of fitness is:

$$\sum_{i=0}^4 \binom{4}{i} f(i) = \binom{4}{0} 4 + \binom{4}{1} \frac{8}{3} + \binom{4}{2} \frac{4}{3} + \binom{4}{3} 0 + \binom{4}{4} 5 = \frac{83}{3}$$

The fitness values we get are therefore $q_{0000} = \frac{12}{83}, q_{1000} = \frac{8}{83}, q_{1100} = \frac{4}{83}, q_{1110} = 0$, and $q_{1111} = \frac{15}{83}$. By substituting these values into Equation 4.10, we get an equation in q only:

$$68q^3 - 12 - 8q - 4q^2 = 0.$$

Here, the solution is $q = 0.653$. (Note that the numerical values from now on are approximate.) We substitute into Equation 4.5, Equation 4.6, Equation 4.7, Equation 4.8, and Equation 4.9

Pr(Y ₁)	
0	0.519
1	0.481

Pr(Y ₂ Y ₁)		
Y ₁ :	0	1
0	0.653	0.470
1	0.347	0.530

Pr(Y ₃ Y ₁ , Y ₂)				
Y ₁ :	0		1	
Y ₂ :	0	1	0	1
0	0.653	0.653	0.653	0.290
1	0.347	0.347	0.347	0.710

Pr(Y ₄ Y ₁ , Y ₂ , Y ₃)								
Y ₁ :	0				1			
Y ₂ :	0		1		0		1	
Y ₃ :	0	1	0	1	0	1	0	1
0	0.653	0.653	0.653	0.653	0.653	0.653	0.653	0
1	0.347	0.347	0.347	0.347	0.347	0.347	0.347	1

Table 4.13: Conditional probability tables of BN constructed from deceptive trap fitness function.

respectively, and get:

$$\begin{aligned}
q_1 &= \frac{q_{0000}}{q^3} = 0.519 \\
q_2 &= \frac{q_{1000}}{(1-q_1)q^2} = \frac{q_{1000}}{(1-\frac{q_{0000}}{q^3})q^2} = 0.470 \\
q_3 &= \frac{q_{1100}}{(1-q_1)(1-q_2)q} = \frac{q_{1100}}{\left(1-\frac{q_{0000}}{q^3}\right)\left(1-\frac{q_{1000}}{(1-\frac{q_{0000}}{q^3})q^2}\right)q} = 0.290 \\
q_4 &= \frac{q_{1110}}{(1-q_1)(1-q_2)(1-q_3)} \\
&= \frac{q_{1110}}{\left(1-\frac{q_{0000}}{q^3}\right)\left(1-\frac{q_{1000}}{(1-\frac{q_{0000}}{q^3})q^2}\right)\left(1-\frac{q_{1100}}{\left(1-\frac{q_{0000}}{q^3}\right)\left(1-\frac{q_{1000}}{(1-\frac{q_{0000}}{q^3})q^2}\right)q}\right)} \\
&= 0
\end{aligned}$$

The BN CPTs in Table 4.13 are just the above numerical values substituted into Table 4.12. It should be noted that our construction does not guarantee that the resulting BN is fully deceptive. In fact, we are not even guaranteed equal probabilities for strings (or explanations) of the same unitation. As an example, for two of the bitstrings of unitation two we have $q_{0011} = q_1q(1-q)(1-q) = 0.0408$ and $q_{1001} = (1-q_1)q_2q(1-q) = 0.0512$, while the trap function's value is $4/83 = 0.0482$. However, if one wants to guarantee full deception, one can consider the maximum, minimum, and certain other probabilities of explanations computed from the BN, and make sure that these obey certain sufficient conditions for deceptive functions [Deb and Goldberg, 1994].

4.7 Conclusion and Future Work

Characterizing classes of fitness functions for which evolutionary computation is appropriate is one of the most important current research directions. This chapter focuses on two types of fitness functions: functions of unitation and Bayesian networks, and in particular this chapter introduces an approach to mapping functions of unitation into Bayesian networks. In addition, instances from three classes of functions of unitation have been mapped to Bayesian networks: onemax functions, trap functions, and hill functions. Onemax functions of unitation are non-deceptive, and the constructed onemax BNs are non-deceptive and easy as well. Trap functions of unitation, on the other hand, are deceptive. The trap function instances mapped into highly connected, hard BNs, which can be made deceptive. Hill functions of unitation are similar to trap functions of unitation, but they are not deceptive under the current definition. The instances considered were still mapped to highly connected BNs. However, overall the skewness in the conditional distributions was smaller for the hill BN than for the trap BNs.

Both the trap and the hill functions give highly connected Bayesian networks, suggesting that they are in a sense difficult. In retrospect, this appears reasonable since functions of unitation are *global*, meaning that the entire string is taken into account for fitness computation. However, when the skewness of the conditional distributions is also taken into account, the hill BNs are generally less skewed than the trap BNs, supporting the intuition that hill BNs are ‘easier’ than trap BNs.

While the construction underlying the onemax, trap, and hill functions was based on enumeration, we have also investigated 4-node parametrized BNs, basing them on deceptive trap functions. This is an approach which promises to scale better than the previous one.

Future work in three areas stands out: systematic experimentation, other measures of function difficulty, and other aspects of the relation between functions of unitation and BNs. Regarding systematic experimentation, the approach described here can be used to create BNs for systematic experimentation, along the following lines: Suppose we have a deceptive BN consisting of k nodes Y_1, \dots, Y_k . Now we can construct a BN consisting of n nodes by making c copies of the k nodes as

follows:

$$\Pr(X_1, \dots, X_n) = \prod_{i=1}^c \Pr(Y_{1_i}, \dots, X_{n_i}).$$

There are several parameters to vary when constructing these networks, even when they are constructed from functions of unitation: Size of deception, size of deceptive network, number of deceptive sub-networks, and signal difference between deceptive and non-deceptive optima.

Regarding other aspects related to functions of unitation and BNs, additional research is needed to exactly formalize the conditions under which BNs are deceptive or non-deceptive, similar to that done for functions of unitation [Deb and Goldberg, 1993]. Second, one could investigate empirically the degree to which there is deception in some ‘natural’ BNs, i.e. BNs that have been developed for application purposes. Third, certain extensions may be made to the construction in Section 4.6: avoid zero in tables such as Table 4.13 above; generalize argument above, so that it applies to BNs of any size, not just the 4-node one; investigate dropping edges in the deceptive BN model, since fully connected BNs are realistic only for BNs of small size (and in fact, the edge to node ratio in application BNs is surprisingly low).

Chapter 5

The Stochastic Greedy Search Algorithm

In this chapter we present a stochastic local search algorithm, stochastic greedy search (*SGS*), which also has a stochastic initialization component. The core of *SGS* is a generalization of the GSAT [Selman et al., 1992, Selman et al., 1994] algorithm to the probabilistic case. We introduce and experiment with two variations of the gain (or change in utility) function, the core of the algorithm. One of the gain functions, GMPE gain, is a generalization of GSAT, while the other gain function, Blanket gain, is a probabilistic version of the GSAT gain.

Our research on stochastic local search has an algorithmic component, *SGS*, and a methodological component. The algorithmic component is the main topic of this chapter, while the methodological aspects are in focus in Chapter 6. However, since the justification for *SGS* is to be effective on certain BNs where exact inference is hard, we briefly preview our approach to systematically study the scalability of algorithms for MPE computation. First, given that randomly generated networks give, in general, easy inference tasks and are therefore not useful (and may even be misleading) for scalability studies, we develop a way to generate increasingly harder BNs in a systematic way. We show how to generate BNs for which computing the MPE implies identifying a satisfying assignment of a corresponding CNF formula. We then build on work that has been done in the deterministic case [Mitchell et al., 1992], and generate hard instances of these problems. Extensive experiments have been done with one of the best probabilistic inference packages available, the Hugin system, which is based on clustering and propagation in join trees [Lauritzen and Spiegelhalter, 1988, Dawid, 1992] — see also Section 2.2.2 for a presentation of clustering. These experiments show that the clustering approach has significant problems computing the MPE of increasingly hard instances,

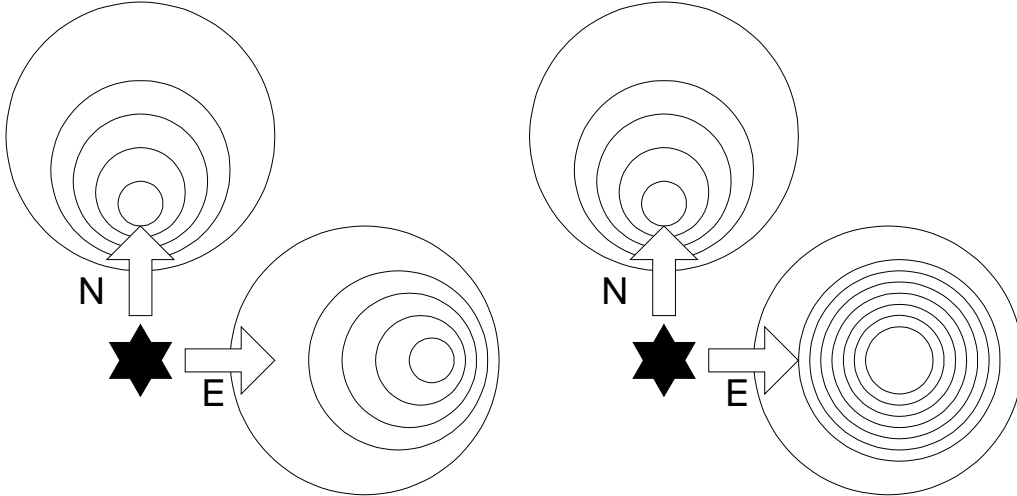


Figure 5.1: Illustrating the criterion of choice concept for stochastic greedy search in the context of choosing between climbing north (N) and east (E). To the left, climbing N, which locally gives the best gain, is globally optimal. To the right, climbing N, which still gives the best gain locally, does not lead to a globally optimal solution. In order to reach the global optimum to the right, at least probabilistically, one can use a stochastic criterion of choice, such that one sometimes chooses E rather than N, even though the latter is locally superior.

until it breaks down completely and requires an unreasonable amount of time. For certain BNs, *SGS* performs several orders of magnitude better than the Hugin system.

In this chapter, we experiment with our stochastic local search algorithm on application BNs. We find that our algorithm is quite effective on these networks, and performs comparably to Hugin. Here, one significant component in the success of the algorithm is the stochastic initialization module, in particular the dynamic programming and forward simulation algorithms. Essentially, this initialization module suggests a good starting point to the stochastic local search component of *SGS*. The dynamic programming algorithms compute a complete solution of a (randomly) simplified network, for which the inference algorithm is tractable. Two other significant reasons for *SGS*'s success are the ability to exploit different search and initialization algorithms or operators, and also the GMPE measure of gain turns out to be powerful.

The rest of this chapter is organized as follows. Section 5.1 describes our top level stochastic local search algorithm. Section 5.2 describes operator-based *SGS*, and in Section 5.3 we present initialization algorithms. In Section 5.4 we present the experimental design and results. Section 5.5 concludes and outlines future work. Chapter 6 is closely related to this chapter, as it focuses

on systematically generating BNs and using them for *SGS* and Hugin experimentation.

5.1 Stochastic Local Search

This section presents the stochastic local search algorithm, *SGS*. We first present the top-level algorithm, then we present the two measures of gain, Blanket gain and GMPE gain.

5.1.1 The Search Algorithm

Stochastic greedy search (*SGS*) is a local search algorithm augmented with stochastic initialization and noise steps. *SGS* is based on previous research on stochastic local search for computing satisfying assignments in propositional formulas [Selman et al., 1992, Selman et al., 1994]. There is also a similarity to the stochastic simulation algorithm [Pearl, 1988] which is primarily intended for belief updating but can also be used for belief revision. *SGS* is also related to ILS (see Chapter 2 and [Lin et al., 1990, p. 129]) and SLS [Kask and Dechter, 1999]. However, ILS does not incorporate the noise and initialization capabilities of *SGS*. SLS, which was developed independently and concurrently with *SGS*, shares several characteristics, such as noise — or stochastic — steps and initialization. However, several of the details are different in *SGS* and in SLS, including the initialization algorithms, the measures of gain, as well as the fact that *SGS* has an operator-based variant which we describe in Section 5.2.

SGS, which is presented in Figure 5.2, starts from an explanation \mathbf{x} , and performs local changes to \mathbf{x} in order to improve the probability $\Pr(\mathbf{x})$. Different *measures of gain* M , closely related to but not necessarily directly based on $\Pr(\mathbf{x})$, can be used to guide the local changes, and we present BLANKET and GMPE below. In addition to a measure of gain, *SGS* encompasses the notion of *criterion of choice* C . This criterion determines how the algorithm decides, based on the gain, which variable to flip next. We have investigated two types of criteria of choice, Max and Stochastic. The first, Max, chooses greedily and yields classical steepest-ascent hill-climbing. The Stochastic criterion yields probabilistic hill-climbing, by probabilistically deciding which state to flip, typically based on the state’s relative utility according to the chosen measure of gain M . The Stochastic criterion is related to the stochastic simulation algorithm as well as to the noise strategies used in stochastic local search [Selman et al., 1994]. When we want to make the gain

M and choice C explicit, we denote the algorithm SGS/MC , with $M \in \{B, G\}$ and $C \in \{M, S\}$, where B stands for Blanket, G for GMPE, M for Max, and S for Stochastic. Note that M and C can also be very simple, or essentially by-passed, such as in the case where uniform noise (NU) is applied. For uniform noise, a non-evidence node is first picked at random, and then a new state of that node is picked at random.

Intuitions behind the concepts of gain and criterion of choice are illustrated in Figure 5.1. Suppose our current state is where one of the stars is, and we need to decide between north (N) and east (E) as our next step. A measure of gain defines how we measure progress for each of our candidate next steps. A criterion of choice determines how we choose between the different candidate steps, based on the candidate gains, using a Max criterion or a Stochastic criterion. In both situations in Figure 5.1, a Max criterion would always choose to step north, while a Stochastic criterion would step north only most of the time. Note that in the situation to the right, climbing E happens to be the better choice, since this hill is higher. Clearly, in the situation to the left, always stepping north is very good, while in the situation to the right it is very bad. The Stochastic criterion avoids always making such very good or very bad choices by climbing according to a probability distribution.

The SGS input parameters MTRIES (maximum number of tries) and MFLIPS (maximum number of flips) control, respectively, how many restarts are performed and how many local flips are done before a restart. P-NOISE controls the noise level. Applying noise amounts to taking a random step with probability P-NOISE, and taking a greedy step using the chosen measure of progress with probability $1 - \text{P-NOISE}$. The input parameter GAIN controls which measure of gain and which measure of utility is used. The ℓ parameter controls the termination of SGS , and we terminate when an explanation \mathbf{b} is found such that the utility of \mathbf{b} is ℓ (e.g., with a probabilistic progress measure we have $\Pr(\mathbf{b}) = \ell$). Other termination criteria can easily be incorporated into SGS . In particular, we have implemented the option of terminating after executing SGS for a certain time period, even though this is not shown in Figure 5.2.

The SGS algorithm returns two values. The first value is a Boolean value signifying whether search was successful or not; the second value \mathbf{b} is the explanation with the highest score. The input value $\ell = 0$ means that no noise should be applied; in this case SGS is successful when

reaching a local optimum. With $\ell > 0$, *SGS* is successful if ℓ is attained.

SGS uses *ComputeGain*, see Figure 5.3, to evaluate candidate tuples, i.e. to compute the gain of nodes and states. *ComputeGain* iterates over all nodes and over all states of each node X_i , excluding the node's current state x_i , and computes which nodes are candidates to have their state changed, along with their gain.

The *ChooseState* function applies the criterion of choice and picks which state is flipped. Then, in $\text{Utility}(\mathbf{x}, \text{GAIN})$ the utility measure related to GAIN is applied. If GAIN is BLANKET, the well-known probability product utility of Equation 2.1 is used:

$$\text{Utility}(\mathbf{x}, \text{BLANKET}) = \Pr(\mathbf{x}).$$

If GAIN is GMPE, then the following probability sum utility is used:

$$\text{Utility}(\mathbf{x}, \text{GMPE}) = \sum_{i=1}^n \Pr(x_i \mid \pi_{X_i}).$$

Using one of these two utility measures we keep the better explanation of \mathbf{x} and \mathbf{b} , and iterate unless the limit ℓ has been reached.

Note that *ComputeGain* implements both the BLANKET and the GMPE gains; which one is used depends on the value of the input parameter GAIN. Within the inner-most loop, *ComputeGain* contains implementations of the BLANKET and GMPE measures, which are presented below. Within that loop there are also calls to *NodeChildrenProbability*, which computes the utility of node X_i 's state in explanation \mathbf{x} , using the utility gain specified by GAIN. In *ComputeGain*, we have for simplicity assumed that all non-evidence nodes are iterated over. The generalization to a subset of all nodes is easy.

5.1.2 Measure 1: BLANKET Gain

To simplify exposition, we assume $|\Omega_X| = 2$ in this and the following subsection. The intuition behind BLANKET is that the state of a node X should be flipped from x to \bar{x} if this leads to a higher probability in the explanation which contains the instantiation x for node X . More formally,

we compute the BLANKET gain $\Delta_B(x, \bar{x})$, where $C_i \in \Psi_X$:

$$\Delta_B(x, \bar{x}) = \frac{\Pr(\bar{x} \mid \pi_X) \times \prod_{C_i} \Pr(c_i \mid \bar{\pi}_{C_i})}{\Pr(x \mid \pi_X) \times \prod_{C_i} \Pr(c \mid \pi_{C_i})}. \quad (5.1)$$

The BLANKET gain is obtained as follows. Let \mathbf{x} be the explanation before flipping, \mathbf{x}' the explanation after flipping some number of nodes. A reasonable measure of the goodness of those flips is

$$\Delta_M(\mathbf{x}, \bar{\mathbf{x}}) = \frac{\Pr(\mathbf{x}')}{\Pr(\mathbf{x})}, \quad (5.2)$$

since the probability of the flipped explanation $\Pr(\mathbf{x}')$ can be computed by taking the probability of the old explanation $\Pr(\mathbf{x})$ and multiplying it with the gain of Equation 5.2:

$$\Pr(\mathbf{x}') = \Delta_M(\mathbf{x}, \bar{\mathbf{x}}) \Pr(\mathbf{x}).$$

By applying Equation 2.1, we can write (5.2) as

$$\Delta_M(\mathbf{x}, \mathbf{x}') = \frac{\prod_{i=1}^n \Pr(x'_i \mid \pi'_{X_i})}{\prod_{i=1}^n \Pr(x_i \mid \pi_{X_i})},$$

where $x'_i = x_i$ if node X_i was not flipped, $x'_i = \bar{x}_i$ if node X_i was flipped, and parent instantiations π'_{X_i} and π_{X_i} are changed to reflect the flipped nodes.

If only one node X is flipped, we can exploit the locality of the BN to restrict the amount of computation needed. In particular, we only need to consider the Markov blanket of X , that is the CPT in X itself as well as CPTs in X 's children $C \in \Psi_X$:

$$\begin{aligned} \Delta_M(\mathbf{x}, \bar{\mathbf{x}}) &= \frac{\prod_{i=1}^n \Pr(x'_i \mid \pi'_{X_i})}{\prod_{i=1}^n \Pr(x_i \mid \pi_{X_i})} \\ &= \frac{\Pr(\bar{x} \mid \pi_X) \times \prod_{C_i} \Pr(c_i \mid \bar{\pi}_{C_i}) \times \prod_{Y_i} \Pr(y_i \mid \pi_{Y_i})}{\Pr(x \mid \pi_X) \times \prod_{C_i} \Pr(c_i \mid \pi_{C_i}) \times \prod_{Y_i} \Pr(y_i \mid \pi_{Y_i})} \\ &= \frac{\Pr(\bar{x} \mid \pi_X) \times \prod_{C_i} \Pr(c_i \mid \bar{\pi}_{C_i})}{\Pr(x \mid \pi_X) \times \prod_{C_i} \Pr(c_i \mid \pi_{C_i})}, \end{aligned}$$

where \mathbf{Y} are the BN nodes other than X and Ψ_X , $\mathbf{Y} = \mathbf{V} - X \cup \Psi_X$, and $Y_i \in \mathbf{Y}$. This is exactly the BLANKET gain $\Delta_B(x, \bar{x})$ (5.1).

The implementation of the BLANKET measure in *SGS* is shown in Figure 5.3. The assignment to Old is the denominator in BLANKET, while the assignment to New is BLANKET’s numerator. BLANKET is not defined if the numerator is zero. This leads us to test for zero in ComputeGain, and take the following action: If Old (numerator in Equation 5.1) and New (denominator in Equation 5.1) are both zero, we define BLANKET gain to be one. If Old is zero but New is non-zero, we greedily pick the new state s , which is \bar{x} in the binary case, and return. So in this case, we are more greedy than usual, since the first state in the first node where a flip leads from a zero to a non-zero probability is picked as a candidate for flipping. It is not hard to think about BN topologies where such an approach is too greedy, and using the GMPE gain would be better.

5.1.3 Measure 2: GMPE Gain

We may take the logarithm of the BLANKET gain in (5.1), under the assumption that all CPT entries involved are non-zero

$$\begin{aligned} \ln(\Delta_B(x, \bar{x})) &= \ln\left(\frac{\Pr(\bar{x} \mid \pi_X) \times \prod_{C_i} \Pr(c_i \mid \bar{\pi}_{C_i})}{\Pr(x \mid \pi_X) \times \prod_{C_i} \Pr(c \mid \pi_{C_i})}\right) \\ &= \ln(\Pr(\bar{x} \mid \pi_X)) + \sum_{C_i} \ln(\Pr(c_i \mid \bar{\pi}_{C_i})) - \left[\ln(\Pr(x \mid \pi_X)) + \sum_{C_i} \ln(\Pr(c \mid \pi_{C_i})) \right]. \end{aligned}$$

We now drop the assumption that all CPT entries involved are non-zero, since, after all, zero entries often occur in BNs. We also drop the logarithms, giving the GMPE gain

$$\Delta_G(x, \bar{x}) = \left[\Pr(\bar{x} \mid \pi_X) + \sum_{C_i} \Pr(c \mid \bar{\pi}_{C_i}) \right] - \left[\Pr(x \mid \pi_X) + \sum_{C_i} \Pr(c \mid \pi_{C_i}) \right], \quad (5.3)$$

where $C_i \in \Psi_X$. The main advantage of this new measure is that it is well-defined and gives a measure of gain even when zero entries are involved. And as we will see in experiments, this is often very important. Also, we can handle the zero case without complicating the algorithm as was needed for BLANKET—see Figure 5.3. One can regard the GMPE gain as a smoothed variant of the Blanket gain, giving an approximation of the Blanket gain.

Note that the GMPE gain is not correct, that is a positive GMPE gain does not imply an increase in an explanation’s probability. However, GMPE gain is a strict generalization of the GSAT gain [Selman et al., 1992], and as such it remains correct on essentially deterministic BNs, such as those described in Section 6.2.1. The GSAT measure is constructed for satisfiability formulated as an optimization problem, the MAXSAT problem. MAXSAT is an optimization problem where one optimizes the number of satisfied clauses; clearly when optimum is reached in MAXSAT one has also solved the underlying SAT problem.

The reasoning behind the GSAT gain and the GMPE gain shown in Equation 5.3 is similar. Using the GSAT gain, we flip the variable which gives the highest increase in satisfied clauses. Using the GMPE gain, we flip the BN node which gives the highest increase in probabilities when added (and not multiplied as usual). So the GMPE gain is closely related to the probability sum utility measure.

5.2 Operator-Based Stochastic Greedy Search

The *SGS* algorithm we have considered so far has been generalized, giving what we denote an operator-based variant of *SGS*. The operator-based *SGS* retains the structure of standard *SGS*, with two nested loops for tries and flips, and one or several initializations at the start of each try. What is new in operator-based *SGS* is that when an initialization or a search step is attempted, the algorithm chooses one among a portfolio of operators. The choice is done following a round-robin schedule or according to a probability distribution defined by associating a probability with each operator. We have the following classes of operators:

- Initialization operators: Uniform initialization (UN), forward simulation (FS), backward dynamic programming (BDP), and forward dynamic programming (FDP). These operators are presented in Section 5.3.
- Search operators:
 - Greedy operators: GMPE gain with max criterion (GM), Blanket gain with max criterion (BM). These operators are essentially the same as the corresponding greedy search algorithms presented in Section 5.3.

- Stochastic (noise) operators: Uniform noise (NU), Blanket gain with stochastic criterion (BS), GMPE gain with stochastic criterion (GS), stochastic simulation (SS).

The operator-based variant of *SGS*, which we call *OSGS*, is presented in Figure 5.4. *OSGS* also uses a portfolio of search operators `SearchPortfolio` and a portfolio of initialization operators `InitializationPortfolio`. In *OSGS*, we also make a stronger distinction between utility and gain than in standard *SGS*. In particular, each operator has an associated gain, which is unrelated to how overall utility is computed. So, for example, all operators can use GMPE gain, while the overall utility of an explanation can be computed using its probability product utility or the probability sum utility.

The structure of *OSGS* is exactly like that of *SGS*: There is an outer loop for tries, and an inner loop for flips. The three subroutines `Instantiate`, `Search` and `CheckLimit` do much of the work in *OSGS*, employ the Blanket and GMPE measures of gain, and operate as follows. `Instantiate(\mathbf{x} , b , i , UTILITY)` instantiates the explanation \mathbf{x} , using those among the initialization operators UN, FS, BDP, and FDP that are part of the `InitializationPortfolio`. If b is set to **true**, initialization is performed i times, and the best of these explanations, according to the utility measure, is used as basis for search. If b is set to **false**, initialization is performed once. `Search(\mathbf{x})` performs a search step on the explanation \mathbf{x} , using the measure of gain associated with the chosen search operator. The search operator used is chosen from among the stochastic and the greedy operators in the `SearchPortfolio`. `CheckLimit(p , ℓ)` returns **true** if $\ell > 0$ and $p + \varepsilon \ell \geq \ell$, else **false** is returned. Here, ε is a small positive constant, typically we use $\varepsilon = 10^{-3}$ to avoid erroneous non-termination due to numerical rounding errors.

Note how *OSGS* is a generalization of standard *SGS*; we can regard standard *SGS*'s noise and hill-climbing steps as two operators chosen from a `SearchPortfolio` of size two, where one search operator is a greedy operator, the other is a stochastic operator. And we can regard standard *SGS*'s initialization as an initialization operator chosen from an `InitializationPortfolio` of size one.

The advantage of operator-based *SGS* compared to standard *SGS* is generality and flexibility, which leads, as we will see in Section 5.4, to better performance. In addition, the operators make *OSGS* very similar to a GA, which makes their integration easier as a next step. And the standard *SGS* can be simulated using the operator-based *SGS*, so the functionality of standard *SGS* is not

lost. The main disadvantage with operator-based *SGS* is that it is a little slower, due to the overhead associated with the operators. However, we have found it to be only approximately 20% slower, and its benefits outweighs this cost. In the following, we often do not distinguish between these two variants of *SGS*.

5.3 Initialization Algorithms

Stochastic local search can be improved by starting from a good initial explanation. We have investigated the following initialization algorithms for stochastic generation of initial explanations: uniform initialization, forward simulation, forward dynamic programming, and backward dynamic programming. These stochastic initialization algorithms, who all have a complexity of $O(n)$, are presented in the following.

Uniform initialization (UN) assigns initial states independently and uniformly at random. The algorithm goes through all nodes, and if a non-evidence node X is encountered, each state has a probability of $1/|\Omega_X|$ of being chosen. Forward simulation (FS) is a well-known BN simulation algorithm [Henrion, 1988]: Root nodes R are initialized independently at random according to their prior distribution $\Pr(R)$, while non-root nodes are initialized according to their conditional distributions $\Pr(V \mid \Pi_V)$, and only after their parent nodes Π_V have been initialized. Evidence nodes are of course not changed.

The dynamic programming (DP) initialization algorithms are based on the Viterbi algorithm [Rabiner, 1989]. The Viterbi algorithm, after our generalization of it to BNs, correctly computes an MPE in BNs which are trees. So on BNs which are ‘close’ to being trees we also expect these algorithms to work well. These algorithms split a BN into trees, initialize each tree independently, and then combine the explanations in each tree to give an explanation for the entire BN.

Backward dynamic programming (BDP) is inspired by the backward Viterbi algorithm. First, BDP divides the BN into trees, starting from the root nodes. This is done using stochastic depth-first search, such that different trees are typically constructed each time. Second, BDP sets up the DP arrays, starting from the leaf nodes. The DP arrays are one numerical array and one state array per BN node. Third, BDP constructs an explanation by forward propagation over the DP arrays.

BN parameters				Hugin			
Network	N	E	E/N	Compile	Execute	Total	Props
Barley	48	84	1.75	10.48	35	45.84	7
Link	724	1125	1.55	39	2218	2257	157
Munin1	189	282	1.49	147.9	131.2	279.1	1
Pigs	441	592	1.34	1.281	28.02	29.3	111
Pir3	143	168	1.17	0.047	0.015	0.062	3
Water	32	66	2.06	2.469	2.078	4.547	1

Table 5.1: Performance of Hugin on application Bayesian networks. The results in the Total column are for computing an MPE and are in seconds. The Compile and Execute columns give the time taken in each of these two phases; Props gives the number of propagations in the execution phase.

Forward dynamic programming (FDP) is inspired by the forward Viterbi algorithm. FDP works similar to BDP, but starts stochastic search from the leaf nodes, constructs DP arrays from the root nodes, and performs backward propagation from the leaf nodes.

We introduce the following terminology for BNs with tree topology. A forward tree is a BN tree where FDP computes an MPE. A backward tree is a BN tree where BDP computes an MPE.

We often use the same notation as for search operators to indicate which initialization operators are part of the InitializationPortfolio. For example, *SGS/FDP/BDP* means that *SGS* with FDP and BDP in the InitializationPortfolio has been used.

5.4 Experiments

There are several ways in which one can investigate the *SGS* algorithm empirically. In the following, the main focus is on application instances. Section 5.4.1 covers the effect of using different search operators, while Section 5.4.2 covers the effect of applying different initialization operators.

Since *SGS* is incomplete, one question is when it should conclude that it has found an MPE. This is controlled by the values of the *SGS* and *OSGS* parameters MTRIES, MFLIPS, T_{\max} , and ℓ . GSAT literature says that ‘[a]s a rough guideline, setting MFLIPS equal to a few times the number of variables is sufficient’ [Selman et al., 1992]. For WSAT, a variant of GSAT with more directed noise steps [Selman et al., 1994], typically very few tries, with corresponding low MTRIES values, are used [Selman et al., 1994]. Another parameter that is important is noise level P-NOISE, and one also needs to set this. In the following, we have used existing results as guidelines as well as

Bayesian Network	Settings		<i>SGS/Mix</i>		<i>SGS/BM</i>		<i>SGS/BS</i>		<i>SGS/GM</i>		<i>SGS/GS</i>	
	T_{\max}	F_{\max}	T	S	T	S	T	S	T	S	T	S
Barley	581	96	265.4	9	581.1	0	581	0	209	9	448.6	3
Link	2257	1448	72.49	10	63.8	10	2257	0	86.69	10	143.7	10
Munin1	2688	378	11.85	10	15.45	10	2032	5	8.052	10	17.84	10
Pigs	493	882	114.3	10	493	0	493	0	147.7	10	419.9	2
Pir3	62	286	1.799	10	1.938	10	3.091	10	1.056	10	2.494	10
Water	606	64	0.572	10	2.066	10	2.391	10	1.158	10	1.00	10

Table 5.2: Performance of stochastic greedy search using different search algorithms on application Bayesian networks. T_{\max} and F_{\max} give information about the time-out limit and the max number of flips respectively. The *Mix* portfolio is a combination of the BM, GM, BS, and GS search operators and the UN, FS, FDP, and BDP initialization operators. The results in the T columns are for computing an MPE and are in seconds. The results in the S columns present the number of times an MPE was found. Time spent on initialization is included in the time taken for the SGS algorithm.

Bayesian Network	Settings		<i>SGS/Mix</i>		<i>SGS/UN</i>		<i>SGS/FS</i>		<i>SGS/FDP</i>		<i>SGS/BDP</i>	
	T_{\max}	F_{\max}	T	S	T	S	T	S	T	S	T	S
Barley	581	96	265.4	9	581	0	581	0	217.9	10	581	0
Link	2257	1448	72.49	10	2257	0	2.439	10	2257	0	2257	0
Munin1	2688	378	11.85	10	2688	0	0.4406	10	456.7	10	2688	0
Pigs	493	882	114.3	10	493	0	493	0	66.25	10	493	0
Pir3	62	286	1.799	10	62.02	0	62.02	0	61.92	1	0.011	10
Water	606	64	0.572	10	0.420	10	0.1187	10	2.10	10	13.22	10

Table 5.3: Performance of stochastic greedy search using different initialization algorithms on application Bayesian networks. The results in the T columns are for computing an MPE and are in seconds. The *Mix* portfolio is a combination of the BM, GM, BS, and GS search operators and the UN, FS, FDP, and BDP initialization operators. The results in the S columns present the number of times an MPE was found. Time spent on initialization is included in the time taken for the SGS algorithm.

experimented until a good parameter setting was found.

For comparison to *SGS*, we use the state-of-the-art algorithm Hugin [Lauritzen and Spiegelhalter, 1988] [Dawid, 1992]. Hugin uses two phases to compute an MPE, a compilation phase and an execution or propagation phase. The execution phase may consist of several propagations. In the following, the total time of the compilation phase and the execution phase for Hugin to compute an MPE is used when comparing to *SGS*.

For the experiments discussed here, as well as those reported in Chapter 6, a Dell 410 using 1GB of RAM and up to 9GB of swap space has been used. Note that this setup favors Hugin, which is RAM-intensive. Stochastic local search, on the other hand, uses very little RAM.

5.4.1 Application Bayesian Networks: Different Search Operators

For the application BNs, we first ran Hugin. The resulting MPE probability was then used as a utility limit ℓ in *SGS*, and in addition a time limit T_{\max} was used. The time-out T_{\max} input to *SGS* depended on the time used by Hugin to compute the MPE, denote this time T_H . Given T_H , T_{\max} was set as follows: For $0 < T_H \leq 0.1$: $T_{\max} = 1000T_H$; for $0.1 < T_H \leq 10$: $T_{\max} = 100T_H$; for $10 < T_H \leq 1000$: $T_{\max} = 10T_H$; and for $1000 < T_H < \infty$: $T_{\max} = T_H$. Using limits ℓ and T_{\max} might not be natural in applications, however it makes our comparison between algorithms more informative. Settings for *SGS*'s parameters T_{\max} and MFLIPS (F_{\max}) are shown in Table 5.2. For the search operators, the settings (i) $\Pr(O) = 0.9$ and $\Pr(NU) = 0.1$, where $O \in \{BM, BS, GM, GS\}$ and (ii) $\Pr(BM) = 0.25, \Pr(GM) = 0.25, \Pr(BS) = 0.25$, and $\Pr(GS) = 0.25$ were used for the different variants of *SGS* respectively.

Results for BNs from applications are shown in Table 5.1, Table 5.2 and in Figure 5.5. Table 5.1 shows results for Hugin. Rows in Table 5.2 show different BNs, while columns show results for *SGS* variants. For each algorithmic variant, there are two columns, T and S. T gives the average time of the 10 runs performed per BN, while S presents the number of successes. Finding an MPE counts as success.

In Figure 5.5 and Table 5.2, the results from the experiments that focus on the differences between search operators are presented. The main results are as follows. First, the results for *SGS* are comparable to those for Hugin. In some cases, notably for Barley, Pigs, and Pir3, Hugin is

faster, while for Link, Munin1, and Water, *SGS* is faster and has a perfect success score, both for *SGS/GM/NU* and for *SGS/Mix*. Second, the mixture variant *SGS/Mix* is consistently the best or second best *SGS* variant. Therefore, using all available search operators is an easy way to give very good performance. Third, which search operator is best depends on the BN at hand. For the Barley BN, the blanket operators *SGS/BS/NU* and *SGS/BM/NU* do not work well, while *SGS/GM/NU* does. The reason for this is probably that Barley has many small probabilities p in the range $1.0 \times 10^{-2} < p < 1.0 \times 10^{-5}$, giving underflow, and one is more likely to overcome these problems when the GMPE gain is used. For the Link and Munin1 BNs, *SGS/BS/NU* does not work well, while the other *SGS* variants do. The GMPE variants work well because of all the deterministic constraints in these BNs; the reason for the difference between the blanket variants is not well understood. The Pigs BN is a more extreme variant of Link and Munin1, in terms of the difference between performance of *SGS* variants. In the two last BNs, Pir3 and Water, all of the algorithms have perfect scores, and the inference times are not much different.

5.4.2 Application Bayesian Networks: Different Initialization Operators

In a second set of experiments with application BNs, the focus was on investigating the effect of the initialization operators. The experimental approach used is similar to the one described in Section 5.4.1, with the essential difference that initialization operators are used one at a time across all BNs, while the same *BM/GM/BS/GS* portfolio of search operators is employed for all initialization operators. Search operators were invoked probabilistically, with uniform probability. Note that *SGS/Mix* in Table 5.2 is exactly the same version of *SGS* as *SGS/Mix* in Table 5.3.

In Figure 5.6 and Table 5.3, the results from the experiments that focus on different initialization algorithms are presented. The main results are as follows. First, which initialization operator is best is in general very dependent on the BN at hand, however the mixture *UN/FS/FDP/BDP* performs consistently well. For the Barley BN, only *SGS/FDP* and *SGS/Mix* work well. This is due to Barley’s topology, which is close to being a forward tree, in which FDP works perfectly. For the Link BN, only *SGS/FS* and *SGS/Mix* work, because of all the deterministic constraints in these BNs. Munin1 is in several ways similar to Link, but here, surprisingly, *SGS/FDP* also works well. For the Pigs BN, only *SGS/FDP* and *SGS/Mix* work, again because of the BNs topology, which is

close to a forward tree. For the Pir3 BN, both of the dynamic programming operators work, while the other two initialization operators do not. The reason for this is Pir3’s close to tree-shaped topology; some parts of this BN are forward trees, while other parts are backward trees. Notice that the BDP variant is much better, and this is consistent with the topology. Water is the only BN in which all initialization operators have perfect scores. The reason for this is that this BN has a layered topology with both forward and backward tree aspects. There are many deterministic constraints which can be utilized by forward simulation, and since approximately 50% of the CPT entries in Water are zero, and the graph is the most highly connected among the tested application BNs, it is not surprising that *SGS/FS* is the winner.

Second, with the exception of Water, the uniform initialization (UN) performs poorly, and even for Water it is outperformed by forward simulation (FS). It appears quite easy to find an MPE in Water; in fact this is the only BN in which all initialization operators have perfect scores.

The fact that there is, in general, such a large difference in performance between different initialization operators makes us conclude the following. Firstly, initialization plays a key role when doing stochastic local search in application BNs: uniform initialization is successful in only one of the BNs. Secondly, using a portfolio of initialization algorithms is very fruitful. For some BNs, for instance those that have a tree topology or a topology which is almost a tree, it is easy to see that one initialization operator should outperform other operators. However, more generally, it is difficult to predict which initialization operator will find an MPE in the shortest time, and in such cases the portfolio approach pays off.

5.5 Conclusion and Future Work

The main contribution of this chapter is the *SGS* stochastic greedy search algorithm. *SGS* combines greedy search, stochastic search, and initialization algorithms. We have presented two *SGS* variants, standard *SGS* and operator-based *SGS*. We have shown that *SGS* outperforms Hugin on hard BNs constructed from satisfiability instances, and outperforms Hugin on certain application BNs. Core factors in the success of *SGS* are the initialization operators, the GMPE measure, and the approach of using a portfolio of operators for computation. The paradigm for constructing hard Bayesian networks and their use in benchmarking algorithms for computing MPEs in Bayesian networks is

presented in more detail in Chapter 6.

Areas for current and future work include the following. First, the important role of initialization and progress measures, especially in application networks, raises the question of more intelligently choosing the best initialization method for a given BN. Second, in local search, an essential question is how to escape from local minima. In this chapter, our answer has been using noise. Another approach is using crossover from genetic algorithms. Alternatively, the two measures of gain can be decomposed into their parent part and children part, and one could use these individually. More generally, there is potential for additional hybridization, for example combining stochastic local search with Hugin, or combining the FDP and BDP algorithms. Third, additional investigation of when to terminate, other than using a utility or time limit, is needed.

```

SGS(MTRIES, MFLIPS, P-NOISE, GAIN, C,  $\ell$ )
Input:  MTRIES  maximum # of restarts
          MFLIPS  maximum # of flips
          P-NOISE noise probability
          GAIN    GMPE or BLANKET
          C       criterion of choice
           $\ell$      limit
Output: b      estimate of MPE
          bool   success or not
Initialize(b)
for  $i \leftarrow 1$  to MTRIES
  Initialize(x) {stochastic initialization}
  if (Utility(x, GAIN) > Utility(b, GAIN)) b  $\leftarrow$  x
  if ( $\ell > 0$  and Utility(b, GAIN)  $\geq \ell$ ) return (true, b)
  for  $j \leftarrow 1$  to MFLIPS
    Noise  $\leftarrow$  Random01() < P-NOISE
    if (not Noise)
      ComputeGain(x,  $c$ , GAIN, C)
      if (Empty( $c$ ))
        if (P-NOISE = 0) break
        else Noise  $\leftarrow$  true
      else
        (Node, State)  $\leftarrow$  ChooseState( $c$ , C){pick from candidate array}
      endif
    endif
    if (Noise)
      Node  $\leftarrow$  pick node at random
      State  $\leftarrow$  pick state from node at random
    endif
    x[Node]  $\leftarrow$  State {change State of Node in explanation x}
    if (Utility(x, GAIN) > Utility(b, GAIN)) b  $\leftarrow$  x
    if ( $\ell > 0$  and Utility(b, GAIN)  $\geq \ell$ ) return (true, b)
  endfor
endfor
if ( $\ell = 0$ ) return (true, b) else return (false, b)

```

Figure 5.2: The *SGS* stochastic greedy search algorithm.

```

ComputeGain( $\mathbf{x}, c, \text{GAIN}, C$ )
  Input:  $\mathbf{x}$       explanation
            $c$       candidates
           GAIN    gain type
            $C$      criterion of choice
  Output:  $c$    candidates
if (GAIN = GMPE) Initialize( $c, 0$ ) else Initialize( $c, 1$ )
for each non-evidence node  $X_i \in \mathbf{X}$ 
   $o \leftarrow x_i$ 
  Old  $\leftarrow$  NodeChildrenProbability( $X_i, \mathbf{x}, \text{GAIN}$ )
  for each state  $s \in \Omega_{X_i} - x_i$ 
     $x_i \leftarrow s$  {try state  $s$  in  $i$ -th node}
    New  $\leftarrow$  NodeChildrenProbability( $X_i, \mathbf{x}, \text{GAIN}$ )
    if (GAIN = GMPE) Net  $\leftarrow$  New - Old
    else
      if (Old = 0)
        if (New = 0) Net  $\leftarrow$  1
        else
          Force( $c, i, s$ ) {force state  $s$  into candidate array  $c$ }
          return
        endif
      else
        Net  $\leftarrow$  New / Old
      endif
    endif
    Add( $c, i, s, \text{Net}$ ) {add state  $s$  to candidate array  $c$ }
  endfor
   $x_i \leftarrow o$  {reset state to original value}
endfor

```

Figure 5.3: The CHOOSE-NODE algorithm, used by *SGS*.

```

OSGS(MFLIPS, UTILITY, C,  $T_{\max}$ ,  $\ell$ ,  $i$ )
Input: MFLIPS    maximum # of flips
          UTILITY   SUM or PRODUCT
           $T_{\max}$     time limit
           $\ell$        limit
           $i$         # of initializations tried
Output: b      estimate of MPE
          bool    success or not
Instantiate(b, false,  $i$ , UTILITY)
start  $\leftarrow$  Clock()
while (Clock() - start <  $T_{\max}$ )
  Instantiate(x, true,  $i$ , UTILITY)
  for  $j \leftarrow 1$  to MFLIPS
    Search(x)
    if (Utility(x, GAIN) > Utility(b, GAIN)) b  $\leftarrow$  x
    if CheckLimit(Utility(b, UTILITY),  $\ell$ ) return (true, b)
  endfor
endfor
if ( $\ell = 0$ ) return (true, b) else return (false, b)

```

Figure 5.4: The operator-based *SGS* stochastic greedy search algorithm.

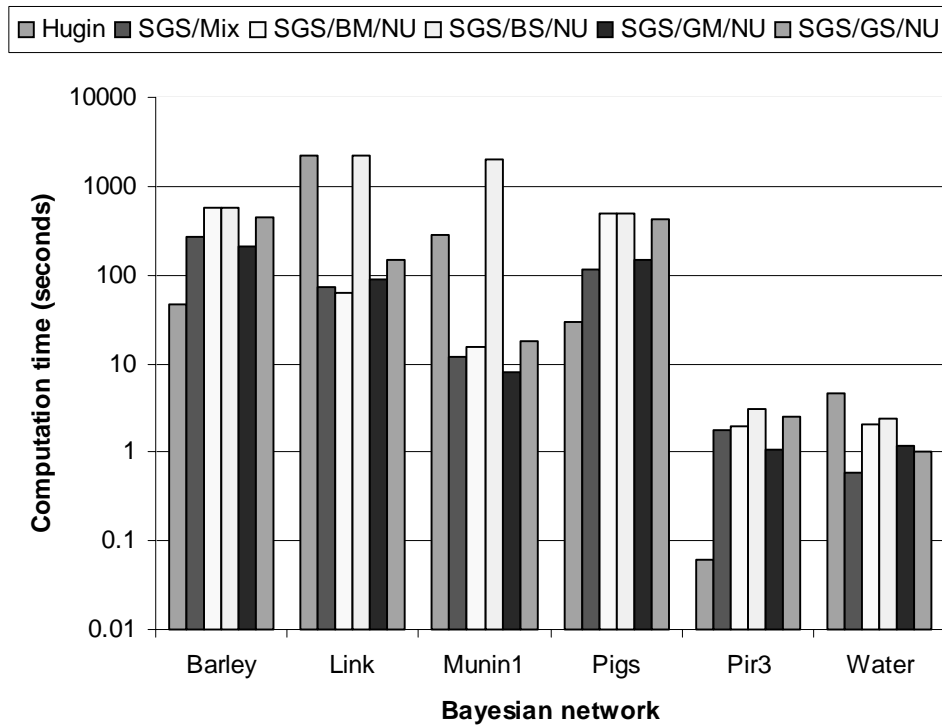
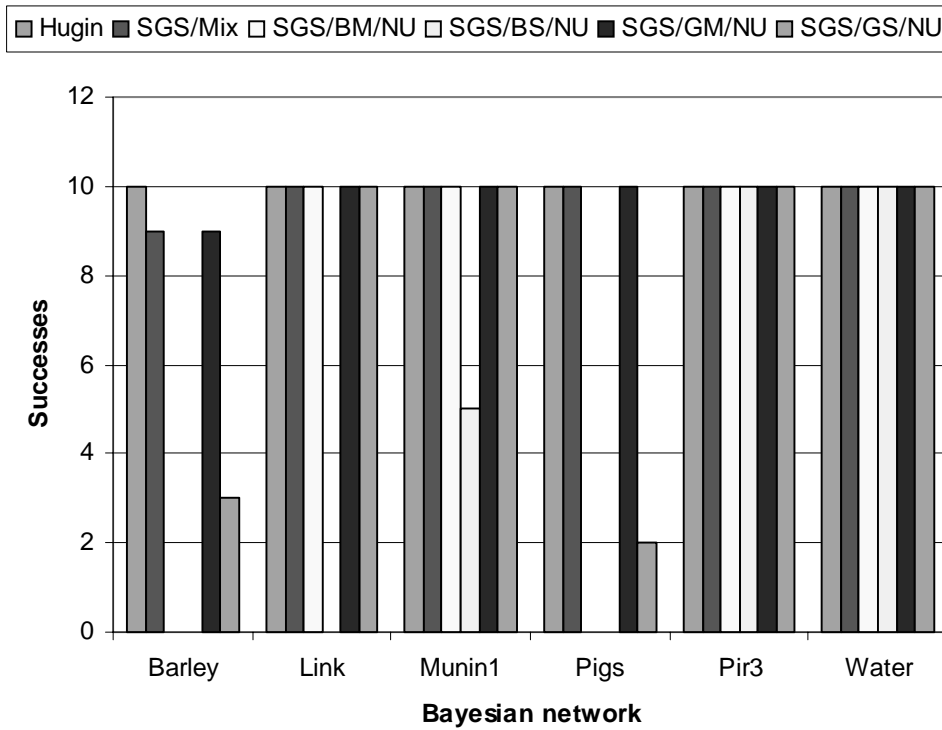


Figure 5.5: Effect of using different *SGS* search operators on application BNs. The MPE computation time (bottom, log scale) and success ratio (top) for Hugin and different variants of *SGS* is shown. The same portfolio of initialization operators is used for *SGS* in all cases.

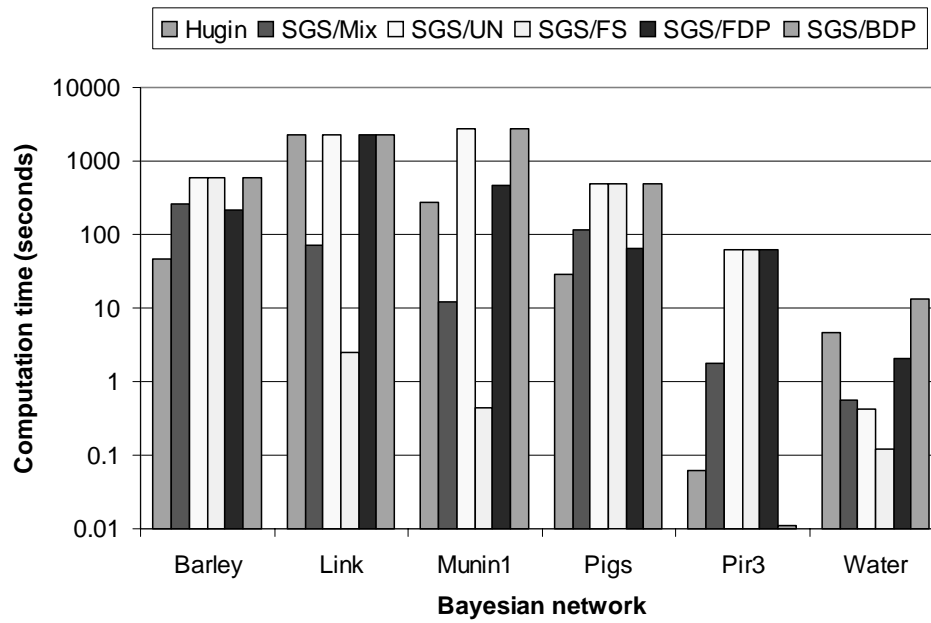
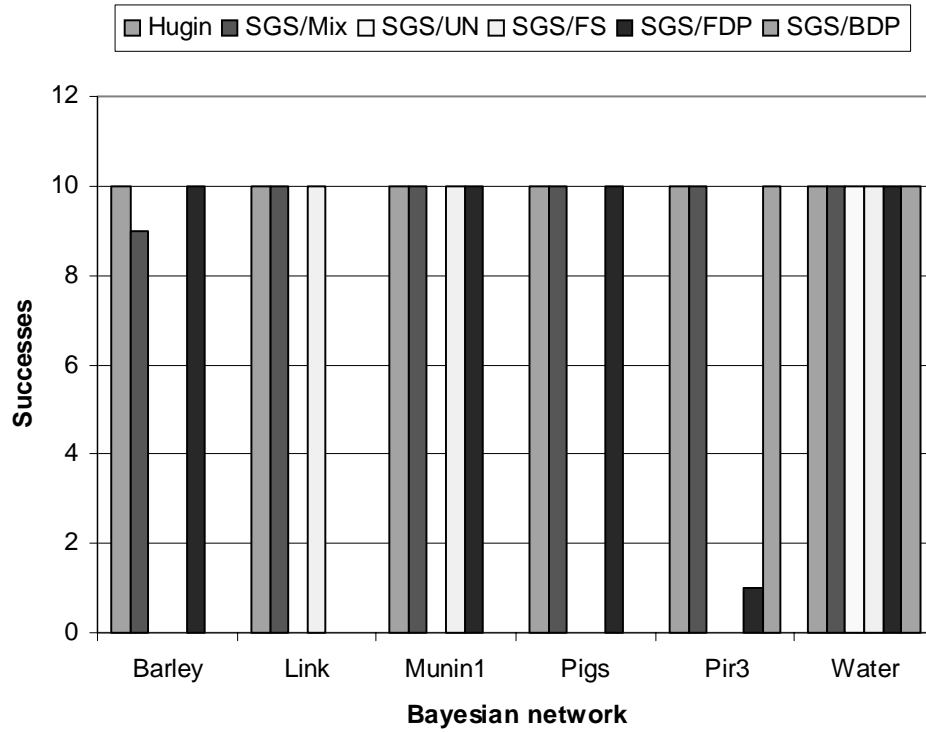


Figure 5.6: Effect of using different *SGS* initialization algorithms. The computation time (bottom) and success (top) for Hugin and different variants of *SGS* is shown. The same portfolio of search algorithms is used for *SGS* in all cases.

Chapter 6

Hard and Easy Bayesian Networks

Experimental work in Bayesian network inference can be performed using networks from applications, as was done in Chapter 5. While this approach is valuable, there are also limitations. First, the number of BNs per application is typically very small, typically one, sometimes up to five. It is impossible to obtain good statistics using such small samples. Second, as we have seen in Chapter 5, BNs vary considerably across applications in terms of the time MPE computation requires, and it is unclear how much one can learn from pooling BNs from different applications. Third, there is the possibility that existing BNs might have been engineered to give adequate performance in existing inference engines, and it seems very valuable to construct BNs that are not biased in this way.

Addressing these limitations associated with using BNs from applications for experiments, this chapter investigates a methodological generation of BN instances so that a systematic comparison of algorithms can be done. Generating Bayesian networks randomly, however, may result in easy inference problems that do not present a challenge to algorithms and may not reflect the hardness of realistic instances. Care has to be taken, therefore, when randomly generating networks, in order to control the hardness of the resulting inference problem.

This chapter presents an experimental paradigm for systematically generating increasingly hard instances used to compute the most probable explanations in Bayesian networks. As a knowledge representation formalism, Bayesian networks are quite expressive, and there are many parameters one can systematically vary in order to gain empirical insights. We study a few of these structural and distributional parameters of Bayesian networks and show how changing them (while maintaining network size) can change the hardness of the problem from a very simple inference problem to

one that existing algorithms cannot handle. Among the parameters we study are the ratio of the number of root nodes to the number of non-root nodes in the network, the irregularity of the graph and the distributional nature of the conditional probability tables.

We also focus on bounding or extreme cases, BNs which are extreme in the sense of being quite uniform along the dimension of interest, such as having only 0/1-probabilities. The advantage of such cases is that, if chosen carefully, they are very hard or easy for one of the algorithms we study. Our justification for focusing on such bounding cases is that unless one obtains results there, the less extreme cases will not be interesting, either. While the focus on extreme cases is less realistic from an application point of view, it makes the systematic experimentation much more informative.

The hardness of the networks is investigated experimentally using one of the most successful commercial inference algorithms, Hugin, along with the stochastic greedy search algorithm (SGS) that we have developed, and which is presented in Chapter 5. While both algorithms break down as the hardness of the problem increases, we show that they vary significantly along some of the dimensions and, surprisingly, that the performance of the stochastic search algorithm degrades more gracefully in many cases.

The rest of this chapter is organized as follows. Section 6.1 introduces in more detail the methodological points to be made. Section 6.2 presents the SAT and KD models we use for generating BNs, and the relations between them. Section 6.3 discusses the interaction between BN features and algorithmic hardness. Section 6.4 presents experimental evidence, while Section 6.5 concludes and discusses future work. Chapter 5 is closely related to this chapter, as it presents the *SGS* in detail.

6.1 Methodological Background

Essentially all the inference problems studied within the Bayesian network (BN) formalism are known to be computationally hard [Cooper, 1990, Roth, 1996, Shimony, 1994], and given their central role in diverse automated reasoning applications, developing heuristics for these problems is an important research problem. A significant component of this work has to be experimental and rely on the generation of instances, so that a systematic comparison of algorithms can be done. However, while worst case complexity results show inference in very simple (albeit large) networks

to be hard, generating networks randomly may tend to result in easy inference problems that do not present a challenge to the algorithms and may not reflect the hardness of realistic instances.

In this chapter we study the problem of computing the most probable explanation (MPE) in Bayesian networks and present an experimental paradigm for systematically generating increasingly hard instances for computing the MPE, as well as a careful study of some of the factors that contribute to the hardness of inference.

We study a few structural and distributional parameters of Bayesian networks and show how changing them (while maintaining network size) can change the hardness of the problem from a very simple inference problem to one that existing algorithms cannot handle. Among the parameters we study are the ratio of the number of root nodes to the number of non-root nodes in the network, the irregularity of the underlying graph, and the distributional nature of conditional distribution tables.

The hardness of the networks is investigated experimentally using two families of networks and two algorithmic approaches. Our selection of families of hard networks to study extends research on generating hard instances for satisfiability problems [Mitchell et al., 1992]. We show the relation between computing the MPE and identifying a satisfying assignment of a corresponding CNF formula and use this relation to build hard instances for MPE. As a result, we concentrate on two types of networks – those that are derived directly from instances of satisfiability, via a mapping we present, and a class of networks introduced in [Kask and Dechter, 1999], for which we show the relations to SAT-like networks. In both cases, generating random networks may result in very easy instances, but a careful selection of the parameters along the dimensions we discuss, even while fixing the size of the network, may result in networks that existing algorithms cannot handle.

Algorithmically, we investigate two very different approaches. We use the Hugin system, one of the best probabilistic inference packages available, which is based on clustering and propagation in join trees [Lauritzen and Spiegelhalter, 1988, Dawid, 1992]. Even though Hugin was originally developed for belief updating [Lauritzen and Spiegelhalter, 1988], it was later extended to handle MPE computation [Dawid, 1992]. The algorithm stays essentially the same, modulo the use of maximization rather than marginalization for MPE computation, however MPE computation can require multiple propagations. So a BN which is hard for Hugin MPE computation will also be hard

for Hugin belief updating, unless the hardness is only caused by a high number of propagations. The second algorithm we use is the stochastic search algorithm *SGS*, which we have developed.

Our findings allow us to identify clear trends in how to generate hard and easy instances for the MPE problem, while showing at the same time some interesting properties of the algorithms and their dependency on various conditions. Structurally, we identify similar trends for both types of networks and both algorithms. Let V be the number of root nodes in the network and C be the number of non-root nodes. The ratio C/V is an indication to the hardness parameter of the network; as this ratio grows, when fixing the number $N = C + V$ of nodes in the network, the MPE problem becomes harder. We identify a threshold phenomena, the value of which may depend on the setting of other parameters, above which the time to find the MPE grows exponentially. A second structural parameter we consider is the regularity of the underlying graph of the network. For the clustering and propagation based algorithm we show that, keeping the size of the network constant, the inference problem for regular graphs is harder than the problem for irregular graphs. Perhaps not surprisingly, this issue does not affect the stochastic search algorithm in a significant way.

Our distributional study is concerned with changing the nature of the CPT tables while keeping the structure of the network fixed. We observe a significant difference between the behaviors of the two algorithmic approaches we study. While the clustering and propagation based algorithm scales very badly when the CPT contains very extreme values (namely, the CPT simulates logical gates) it behaves much nicer when the CPTs are smoother. Quite the opposite behavior is observed for the stochastic algorithm, which is exponentially faster than Hugin when the CPTs have extreme values. In the smoother CPT, we found that the type of initialization used in the stochastic algorithm significantly affects the inference time.

6.2 Distributions of Bayesian Networks

Since inference in BNs is computationally hard, an important question is how to systematically study heuristics for this problem. One of the major problems one faces is that for many hard problems simply generating random instances yields fairly easy problem instances and may mislead this effort [Angluin and Valiant, 1979, Franco and Paull, 1983, Cheeseman et al., 1991].

This problem has been addressed in the context of satisfiability, in the seminal work of [Mitchell et al., 1992], where it was shown how to generate hard instances for 3SAT. These ideas can be generalized and used to generate hard instances for the MPE problem. The basic idea is to start with a CNF formula and generate a BN in which the MPE corresponds to a satisfying assignment of the original CNF. In this way hard instances of satisfiability correspond to hard instances of the MPE problem. Moreover, the threshold phenomena identified for instances of satisfiability, namely that the hardness of the problem depends on the ratio of the number of clauses to the number of variables, can be generalized to the case of Bayesian networks.

We generalize this parameter and consider the ratio C/V where V is the number of root nodes in the network and C is the number of non-root nodes. This turns out to be the natural generalization for the families of networks we study and, as in the case of Boolean satisfiability we exhibit a threshold phenomena, even when keeping $N = C + V$ fixed.

We consider two families of networks. First we present SAT-like networks, in which the mapping from the satisfiability problems is fairly direct. Then we discuss a more general construction, presented in [Kask and Dechter, 1999], which we show to be very related to the SAT networks and, as a result, can be studied from the same point of view.

We should qualify our results by pointing out that they only apply for particular algorithms, in our case Hugin and SGS. In the context of SAT, the following has been stated [Mitchell et al., 1992]: ‘[W]e present empirical results showing that random instances of satisfiability can be generated in such a way that easy and hard sets of instances (for a particular SAT procedure, anyway) are predictable in advance.’ Although there is significant empirical evidence that certain hard problem regions carry over between algorithms, there is as yet little theoretical support for this observation.

6.2.1 The SAT Model

Building on a construction from [Cooper, 1990, Roth, 1996] the basic idea is to start with a CNF formula and generate a Bayesian network for which an MPE corresponds to a satisfying assignment of the CNF. Thus, starting with a hard CNF yields a BN for which computing the MPE is hard. While the straight forward construction yields a hard, yet simple BN, in the sense that the variables are all Boolean and the conditional probability tables are simple, we also suggest ways to generalize

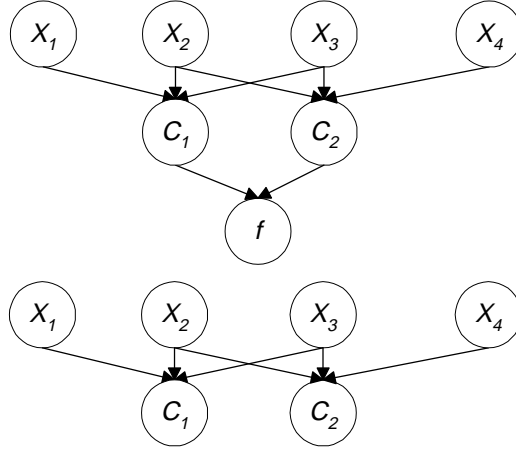


Figure 6.1: Example of constructing a BN from a CNF formula.

beyond that, and it is easy to verify that the resulting networks are still hard.

Figure 6.1 depicts the general idea of the construction. Given a 3CNF formula $f = \bigwedge_{i=1}^m C_i$ with clauses $C_i = X_{i_1} \vee X_{i_2} \vee X_{i_3}$ we construct a layered Bayesian network, in which one layer of nodes corresponds to the variables X_j , a second corresponds to the clauses C_i , and a third corresponds to the function value node, f . The variables X_j are all root nodes and are linked only to clause nodes. A variable X_j has an edge directed toward C_i iff X_j occurs in the clause C_i . All the clause nodes are linked to the function node f . The conditional probability tables are set so that $P(f = 1) > 0$ iff the assignment to the X_j 's satisfies the 3CNF formula. It is easy to see that this happens if for all i , the conditional probability table associated with the node C_i simulates an **or** gate of three inputs, and the table associated with the node f simulates an **and** gate over all the C_i 's. In order to generate a network that corresponds to a non monotone CNF (CNF formula with negation), the CPT needs to be generalized slightly in a straight forward way.

An example of the above construction over the variables X_1, X_2, X_3, X_4 is shown at the top in Figure 6.1. The corresponding CNF has two clauses with variables X_1, X_2, X_3 and X_2, X_3, X_4 respectively, and the CPT is constructed so as to encode the **or** on the corresponding literals (e.g., X_1 or \bar{X}_1).

In order to generate a BN for the MPE problem it is sufficient to *clamp* the clause nodes C_i to 1, and in this way drop the node f and the conjunction gate. An example of the resulting network is shown at the bottom of Figure 6.1. It is easy to verify that an assignment of values to the X_j 's

has a positive probability iff it satisfies the corresponding 3CNF formula. There may be many satisfying assignments, all with the same probability, making them all MPEs.

Satisfiability might seem like a limited problem to consider since (i) it is a decision problem, not an optimization problem like computing an MPE and (ii) it gives a particular bipartite BN topology. We provide counter-arguments to these two arguments in the following.

Concerning (i), even though satisfiability is obviously a decision problem, we note that decision problems are special cases of optimization problems. The optimization problem of finding an MPE of non-zero probability correspond to the problem of deciding whether the formula is satisfiable. And even within the SAT framework, regardless of the BN perspective, it has proven fruitful to view this decision problem as an optimization problem, known as the MAXSAT problem. MAXSAT is an optimization problem where one optimizes the number of satisfied clauses. So even though satisfiability is a decision problem, there is a strong connection to optimization problems, and emphasizing the difference between decision problems and optimization problems does not seem fruitful.

The advantage of focusing on this particular decision problem is that it is a bounding case where we expect a stochastic local search algorithm to perform well, and Hugin not so well. One can smooth and perturb the 0/1-probabilities in a SAT BN, and gradually make it more of a ‘realistic’ optimization problem. However, if one cannot show, experimentally, superior performance for one algorithm over another in the bounding case of SAT BNs, these more complicated experiments are not worthwhile.

Concerning (ii), we argue that even though the bipartite SAT BNs have a limited topology, this topology is, first of all, interesting in its own right. Several applications, including medical diagnosis as in the QMR-DT BN [Lin et al., 1990] and information theory [Gallager, 1962] [Frey, 1998], use BNs with this topology. Second, inference algorithms have been proposed for this topology [Henrion, 1991]. A third reason why the bipartite topology is interesting is that it provides a well-understood stepping stone for other topologies. As we detail below, one can regard the leaf nodes in a SAT BN as the non-root nodes in an arbitrary BN. Or, one can regard the two levels as a first step in the direction of considering multi-level BNs, and all BNs can certainly be organized into multiple levels [Castillo et al., 1997].

There are several ways to generate random 3CNF formulas. Assume one chooses the policy: work with V variables and C clauses; generate the clauses by selecting variables uniformly into clauses and negate each variable with probability 0.5 [Mitchell et al., 1992]. In this case, it turns out that there is a *phase transition* phenomena between easy problems and hard problems that depends on the ratio between the number of variables V and the number of clauses C , namely, C/V . Through extensive experimentation it has been established that for 3CNF formulas there is a phase transition around $C/V \approx 4.25$, and the hardest instances are found in the region around the phase transition, the *hard region* [Mitchell et al., 1992]. Via the construction above, the importance of this ratio carries over to BNs. The threshold value itself may change, though, if other parameters in the generation are changed. Notice that in our case, V is the number of root nodes in the network and C the number of non-root nodes in the network. C/V turns out to be an interesting hardness parameter even for more general networks.

In addition, we sometimes consider the ratio between the number of edges in the network E , and the number of nodes N . This measure is more general from the BN point of view, since it does not assume the bipartite structure of CNF-like BNs. In our case, a 3CNF on V variables and C clauses yields a network with $N = V + C$ variables and $E = 3C$ edges. A ratio of $C/V = k$ implies $E/N = 3k/(k + 1)$ (e.g., for $k = 4$, $E/N = 2.4$).

Note that we are not claiming that the ratio is constant between models. However, what we do claim is that with increasing C/V (or E/N) ratio for a given model (such as the SAT model), we expect to see a similar threshold phenomenon around some ratio, where the ratio most likely is model-dependent. Whether this is the case or not is an empirical question, and in order to gain some more insight regarding this, we turn to the KD model.

6.2.2 The KD Model

In this section we follow a suggestion for generating random Bayesian networks, suggested recently by Kask and Dechter [Kask and Dechter, 1999]: The procedure they suggest can be viewed as follows. Choose the following three parameters: N - the number of nodes in the network; V - the number of root nodes and P - the number of parents of a non-root node. Construct a networks as follows: Index the nodes from 1 to N , and iterate over the nodes. At the i -th step, pick the i -th

node and pick, uniformly at random P parents among the nodes indexed from $i + 1$ to N . Repeat until there are V nodes left.

This algorithm typically creates BNs with a tree-like topology. However, notice that if $C = N - V$, the number of non-root nodes, is much smaller than V , the graph is likely to be almost bipartite, with C non-root nodes and V root nodes. In this way, this model, which we will call the KD model, is a generalization of the bipartite SAT model. Instead of restricting nodes among the leaf nodes to have parents only among the root nodes, non-root nodes are allowed to have other nodes as children.

Kask and Dechter investigate experimentally the model using what we might call a constant node cardinality approach. In the expression $N = V + C$, N is kept constant, while V and C vary.

6.3 Hard and Easy Networks

In this section, we discuss several factors which might affect the hardness of the problem of computing an MPE. We present the factors and use this to motivate the experiments we performed and our expectations regarding effect various factors may have. In particular, we discuss the C/V ratio, the regularity of the underlying graph and the nature of the CPTs.

6.3.1 Threshold Phenomena C/V

As mentioned earlier, for the case of propositional satisfiability, a *phase transition* phenomena between hard instances and easy instances has been observed. This phenomena depends on the ratio C/V between the number of variables V and the number of clauses C in the CNF formula.

Given the way we have generated the SAT networks from SAT formulas, we have mapped V to be the number of root nodes in the network and C the number of non-root nodes in the network. We expect a similar threshold phenomena to occur here, namely, computing the MPE becomes significantly harder as the ratio increases. Although, given different parameter setting, the value of the threshold itself may vary.

The situation is somewhat more subtle for the KD network, which typically have a tree like structure. However, based on our earlier discussion we have observed that the KD network are essentially a generalization of the SAT networks and thus expect that C/V remains an important

	Children-regular (*/ r)	Children-irregular (*/ i)
Parent-regular (r / *)	Regular (r/r) Classical Gallager codes k CNF and Read- ℓ Regular KD networks	Irregular (r/i) Modern Gallager codes k CNF Irregular KD networks
Parent-irregular (i / *)	Irregular (i/r) Read- ℓ	Irregular (i/i) Many application BNs

Table 6.1: Regular and irregular Bayesian networks classified along the two orthogonal dimensions of children-regularity and parent-regularity. The heading x/y of a cell (such as r/r) means that parent-regularity is x, children-regularity is y.

parameter, and that we can see a threshold phenomena there too.

A third issue that is of interest here, given that we study these issues with two different algorithmic approaches, is the extent to which the hardness is algorithmic dependent.

6.3.2 The Nature of the CPTs

We consider three CPT types: Uniform discrete CPTs, Boolean CPTs and uniform continuous CPTs. (We have also considered adding low levels of noise to Boolean CPTs but this has turned out to give similar behavior to the Boolean cases.) These types cover the CPTs that are required to provide an exact mapping from SAT problems to the corresponding BNs as well as idealizations of CPTs that might occur in applications.

Given the nature of the algorithmic approaches we study - the clustering and propagation based algorithm Hugin versus the stochastic search SGS, the main concern here is the effect of the CPTs on the algorithms.

In the case of Boolean CPTs we expect that the number of MPEs would be relatively large, and therefore this would make the inference problem (keeping other parameters fixed) harder for Hugin to handle while, for the same reason, would make it easier for SGS. Hugin would have to repeatedly propagate until it arrives at one MPE, while SGS is more likely to find one of the MPEs from a good starting point for hill-climbing.

The other extreme case, of uniformly selected CPTs is expected to yield the opposite results. We expect that in this case there will be fewer MPEs, typically one. This will be advantageous to Hugin, having to propagate only once, but may make the problem of finding the MPE harder for a greedy search algorithm like SGS.

6.3.3 Regularity of Graphs

The issue of the regularity of the underlying graph of the network has received some attention in the context of BNs used in information theory [Gallager, 1962] [Luby et al., 1998]. In particular, when using iterated belief propagation to compute the MPE given a codeword transmitted over a noisy channel, irregular networks have been found to perform better than regular ones. Although Gallager’s original code (denoted classical Gallager codes in Table 6.1) enforce the constraint that each root node should have the same number of children, and each leaf node should have the same number of parents [Gallager, 1962], [Luby et al., 1998] provides a compelling argument for why lifting these assumptions may be beneficial for computing the MPE when decoding, giving modern Gallager code. The intuition is that, using a propagation based algorithm, high degree nodes may tend to get quicker to the “right” setting, given that they need to satisfy more constraints and this leads to a “wave effect” that helps lower degree nodes find their “right” setting.

In order to evaluate the effect of this structural property on computing the MPE we introduce the following terminology.

Definition 13 *A BN is parent-regular if all nodes with in-degree greater than zero have the same number of parents. A BN is children-regular if all nodes with out-degree greater than zero have the same number of children. A BN is regular if it is children-regular and parent-regular, else it is irregular.*

Table 6.1 summarizes some families of BNs and how they can be classified into the four classes, using the two orthogonal dimensions of children-regularity and parent-regularity. Most entries in this table should be well-known or explained already; the notion of read- ℓ means that a variable is used ℓ times in a clause. Note that regularity could easily be made more gradual; for example one could use variance in in-degree and out-degree a measure of regularity. With this more general measure, high variance means irregular, low variance means regular. Here, however, we present experiments only with two extreme cases. We investigate the effect of regularity by keeping the number of nodes and edges fixed, and varying how edges are attached to nodes with respect to children-regularity. The construction algorithms of Section 6.2 have been augmented to construct irregular or regular BNs.

We are concerned about two issues. First, the general effect of changing the regularity on the hardness of computing the MPE and second, the extent to which the different algorithms behave differently under these conditions. Our expectation is that, if indeed regularity makes the inference problem harder, an algorithmic approach that depends more on the BN topology, such as Hugin, would be affected more significantly than an algorithm like SGS.

6.4 Experiments

In this section we perform experiments with *SGS* and Hugin, using BNs from KD and SAT distributions, and varying the various parameters that control which BNs are generated from these models.

In Section 6.4.1, we focus on the effect of varying the ratio in BNs where the CPTs are Boolean functions. We show that certain BNs with a high ratio are hard for Hugin, but easy for *SGS*. Section 6.4.2 presents experimental results that lifting the assumption of Boolean CPTs makes certain BNs harder for *SGS*, while they get easier for Hugin. Section 6.4.3 investigates the role of regularity.

In the present experiments, we restrict ourselves to nodes with two states, and Boolean CPTs are OR-gates. The methodology used is to generate a number of BN instances according to the SAT or KD models, run *SGS* and Hugin on these instances, and record statistics such as median, mean μ , and standard deviation σ for the time (in seconds) to compute an MPE.

Unless otherwise noted, the termination criterion we use for *SGS* is that an MPE has been found or a pre-specified, high time-limit has been reached. This might not be the way one would use *SGS* in an application, where one might want to trade off speed and solution quality in a more flexible way, however our approach makes comparison between *SGS* and Hugin easier. For SAT BNs, *SGS* will generally find a satisfiable assignment in a very short time, and we let the algorithm run until a satisfying assignment has been identified. And in other BNs, we generally let *SGS* run until an MPE has been found.

6.4.1 Ratio Experiments

Using the construction presented in Section 6.2.1, 3SAT BNs of varying hardness were generated at random. Important BN parameters are number of variables V , number of clauses C , number of

BN parameters				Hugin		SGS/GM		Comparison
V	C	C/V	E/N	μ_H	σ_H	μ_G	σ_G	μ_G/μ_H
20	40	2	2.00	0.155	0.0536	0.00126	0.00432	8.13×10^{-3}
30	60	2	2.00	3.59	2.19	0.00692	0.00849	1.93×10^{-3}
35	70	2	2.00	17.1	13.4	0.00560	0.00987	3.27×10^{-4}
40	80	2	2.00	90.7	65.4	0.00596	0.00770	6.57×10^{-5}
20	60	3	2.25	0.577	0.263	0.01316	0.02304	2.28×10^{-2}
30	90	3	2.25	28.5	26.8	0.03122	0.05367	1.10×10^{-3}
35	105	3	2.25	234	154	0.06370	0.13364	2.72×10^{-4}
40	120	3	2.25	1425	1110	0.07972	0.11033	5.59×10^{-5}
20	80	4	2.40	1.20	0.495	0.12160	0.28286	1.00×10^{-1}
30	120	4	2.40	101.8	65.5	0.52440	1.01343	5.15×10^{-3}
35	140	4	2.40	1116	908	1.07782	1.80301	9.66×10^{-4}

Table 6.2: Speed (in seconds) of Hugin and SGS for MPE computation on Bayesian network translated from CNF formulas. Note that SGS is typically several orders of magnitude faster than Hugin.

edges E , and number of nodes N . The C/V -ratio and the E/N -ratio are both measures of hardness of a BN; as a rule of thumb we have that the larger these ratios are, the harder the network is on the average. The constructed BNs consist of 20, 30, 35, and 40 variables, and have a C/V -ratio ranging from 2 to 4, see Table 6.2. For *SGS*, 10 MTRIES, $10(C + V)$ MFLIPS, and a noise level of $P\text{-NOISE} = 0.1$ was used.

Results are shown in Table 6.2, where each row summarizes 50 deterministic instances. The mean and standard deviation of MPE computation time is presented for Hugin and one variant of *SGS*. The results for MPE computation are in seconds. The Comparison column summarizes the results by giving ratios for means of computation times of the two algorithms. Although we have experimented with 0/1 instances as well as smoothed instances we present here results only for the 0/1 cases. Also, we only present results for instances which Hugin was able to process. (Hugin was not able to process, at least not in reasonable amounts of time, larger instances than what is presented). The reason for focusing on instances which Hugin could process is that for these cases we know for sure that some non-zero MPE exists. This methodology is the same as used in the seminal research on satisfiability [Selman et al., 1992, Selman et al., 1994].

For Hugin, there are two main effects, as predicted. First, as the C/V -ratio increases towards the hard region, the inference time increases super-linearly. Second, as the number of variables and clauses increases, the inference time increases, too.

BN parameters				Hugin			
V+C	V	C	C/V	median	μ_H	σ_H	BNs
256	146	110	0.75	0.1560	0.1579	0.0141	100
256	136	120	0.88	0.2030	0.2089	0.0497	100
256	126	130	1.03	0.2810	0.3152	0.1089	100
256	116	140	1.21	0.6880	0.8667	0.7101	100
256	106	150	1.42	4.2035	6.6136	7.2083	100
256	104	152	1.46	8.0390	10.9292	10.2812	100
256	102	154	1.51	12.0800	19.8823	22.0354	100
256	100	156	1.56	20.4200	35.3964	45.0436	100
256	98	158	1.61	29.8450	50.8568	53.3685	98
256	96	160	1.67	55.500	96.959	211.920	99
256	86	170	1.98	208.400	806.800	1400.750	5

Table 6.3: Time (seconds) for Hugin to compute the MPE for various parameter settings of the KD model.

For *SGS*, an MPE was found in all cases, so only the search times are presented in Table 6.2. *SGS* with the GMPE measure outperforms Hugin by several orders of magnitude on these BNs. This is summarized in the μ_G/μ_H column in Table 6.2. In this column we also see that the advantage of using *SGS* increases with increasing problem size, while there is no such clear trend for *C/V*-ratio.

In a related experiment, we used $V = 30$ variables, and varied the number of clauses from $C = 60$ to $C = 126$. Again, we use only satisfiable BNs, and run *SGS* until an MPE is found or until the time limit T_{\max} is reached. In Figure 6.2, the results from these experiments are shown. Note that *SGS/GM/NU* consistently finds MPEs in these BNs, while *SGS/BM/NU* does not. This corresponds well with the superior performance of *SGS/GM/NU* compared to *SGS/BM/NU* in Figure 6.2. A second main point is that *SGS/GM/NU* is two to three orders of magnitude faster than Hugin on these networks, while *SGS/BM/NU* is faster than Hugin on networks with $C/V < 3.1$; after that point Hugin is faster.

6.4.2 CPT Experiments

Here we lift the assumption that CPTs are Boolean functions. In particular, we pick CPT values from a uniform continuous distribution at random. In Table 6.3, we present results for Hugin on KD BNs with uniform continuous CPTs. Note that this is different from experiments with SAT in Table 6.2, since here we keep $V + C$ constant rather than V . (The effect of this is a non-linear

BN parameters			Regular SAT BNs				Irregular SAT BNs				Ratio
V	C	C/V	m	$\mu_S^{r/r}$	$\sigma_S^{r/r}$	S	m	$\mu_S^{i/r}$	$\sigma_H^{i/r}$	S	$\mu_S^{r/r} / \mu_S^{i/i}$
30	60	2.0	16.70	19.4	11.5	100	3.67	3.10	2.10	100	3.13
30	66	2.2	29.00	33.1	16.4	100	7.11	8.21	5.01	100	4.04
30	72	2.4	42.84	45.6	19.8	100	9.94	11.61	6.75	100	3.93
30	78	2.6	58.41	65.8	31.8	100	14.60	18.38	12.52	100	3.58
30	84	2.8	101.45	109.8	59.7	100	21.43	26.31	17.98	100	4.17
30	90	3.0	121.05	131.0	66.3	100	29.41	35.32	21.36	100	3.71
30	96	3.2	156.45	158.4	65.4	100	45.64	53.78	33.90	100	2.95
30	102	3.4	168.30	192.0	97.4	100	58.74	67.36	42.79	100	2.85
30	108	3.6	178.55	212.1	114.1	100	77.51	85.76	45.37	96	2.47
30	114	3.8	226.90	264.3	152.4	99	89.73	107.28	66.77	97	2.46
30	120	4.0	257.00	306.1	168.4	96	107.55	121.73	66.86	88	2.51
30	126	4.2	318.25	358.3	165.6	86	125.35	155.18	97.26	68	2.31

Table 6.4: The effect of regular and irregular SAT BNs on Hugin computation time (seconds).

change in the C/V -ratio as a function of a change in C .) Also note that there are two parents per non-root node rather than three as is used in 3SAT. In Figure 6.3, the mean computation time μ_H for Hugin on these networks is depicted. We note the sharp phase transition around the ratio $C/V \approx 1.7$. Also note that the KD BNs used by Kask and Dechter [Kask and Dechter, 1999], with $C/V \leq 0.75$ for the case of $N = 256$, are actually quite easy for Hugin to solve.

In other experiments, we confirmed the expectation that Hugin outperforms *SGS* on BNs with uniform continuous nodes, and also confirmed that these nodes made Hugin run faster than what it does on BNs with Boolean CPTs.

6.4.3 Regularity Experiments

To answer our questions about regularity, we performed experiments as summarized in Table 6.4. BNs were created from the SAT distribution, varying the C/V -ratio, as earlier, but also varying whether they were irregular (corresponding to k SAT formulas) or regular (corresponding to k SAT and Read- ℓ formulas).

Results from the Hugin experiments are presented in Table 6.4 and in Figure 6.4. The main effect is that the more regular BNs consistently require more time for computation. That is, BNs that are regular are slower than those that are irregular. This is summarized in the ratio between the computation means in the regular case $\mu_H^{r/r}$ and in the irregular case $\mu_H^{i/r}$. In particular, we

BN parameters			Regular SAT BNs				Irregular SAT BNs				Ratio
V	C	C/V	m	$\mu_S^{r/r}$	$\sigma_S^{r/r}$	S	m	$\mu_S^{i/r}$	$\sigma_H^{i/r}$	S	$\mu_S^{r/r} / \mu_S^{i/i}$
30	60	2.0	0.14	0.21	0.19	100	0.17	0.25	0.28	100	0.84
30	66	2.2	0.30	0.43	0.44	100	0.20	0.38	0.45	100	1.13
30	72	2.4	0.47	0.93	1.59	100	0.61	1.20	1.58	100	0.78
30	78	2.6	0.90	1.79	2.42	100	1.50	2.5	3.9	100	0.72
30	84	2.8	1.60	3.50	5.50	100	2.10	5.4	7.4	100	0.65
30	90	3.0	4.10	11.70	21.30	100	5.70	13.0	18.0	100	0.90
30	96	3.2	10.50	25.10	35.20	100	13.10	32.2	43.4	100	0.78
30	102	3.4	19.50	41.40	46.30	100	48.70	70.0	57.4	100	0.59
30	108	3.6	72.70	80.50	56.40	100	84.30	85.8	58.6	96	0.85
30	114	3.8	150.00	100.5	59.60	99	150.00	110.6	55.4	96	0.91
30	120	4.0	150.00	133.20	39.40	96	150.00	129.9	42.5	88	1.02
30	126	4.2	150.00	134.20	37.80	86	150.00	134.9	35.9	68	0.99

Table 6.5: The effect of regular and irregular SAT BNs on SGS/BM/NU computation time (seconds).

have that $2.3 < \mu_H^r / \mu_H^i < 4.2$. Interestingly, this ratio decreases as the C/V -ratio increases.

A second effect is that the number of BNs that could be processed under the regular condition is greater than that number under the irregular condition.

In Table 6.5 and Figure 6.4, we investigate the effect of regularity on *SGS*. In these experiments, we use only the BNs which can be processed by Hugin from Table 6.4. So for all BNs up to the 3.4 ratio, there are 100 BNs, while for higher ratios there are fewer BNs. In this case, difference between speed of inference is much smaller, and the ratio is, if anything, reversed compared to Hugin.

In summary, we have showed that regularity is important not only for the iterated belief propagation algorithm when used on information theory BNs. In particular, we have showed empirically that regular BNs are slower to process for Hugin, while for *SGS* the difference between regular and irregular BNs is very small.

6.5 Conclusion and Future Work

This chapter presents an experimental paradigm for constructing hard Bayesian networks and their role in benchmarking algorithms for computing MPEs. We have studied a range of structural and distributional parameterization of BNs and showed how they affect the hardness of computing

MPEs. We have shown that varying these parameters, even when keeping the network size fixed, may make the networks hard to handle for state-of-the-art algorithms like Hugin. In addition, our study compares two distinct algorithmic approaches and show how different parameters values affect them differently to the extent that their performance differs exponentially in the problem size.

The contribution of this work is thus two fold: Methodologically, we believe this line of work to be essential so that valid experimental evaluation of algorithms can be performed. Algorithmically, this work helps in developing an understanding of the suitability of different algorithmic approaches in different settings.

This work can be extended in many directions. Most important is to develop a better understanding to intermediate cases. While we have studied extreme cases in a few dimensions, mainly to exhibit that threshold phenomena exists, it is essential to perform similar studies for less extreme cases along these and other dimensions.

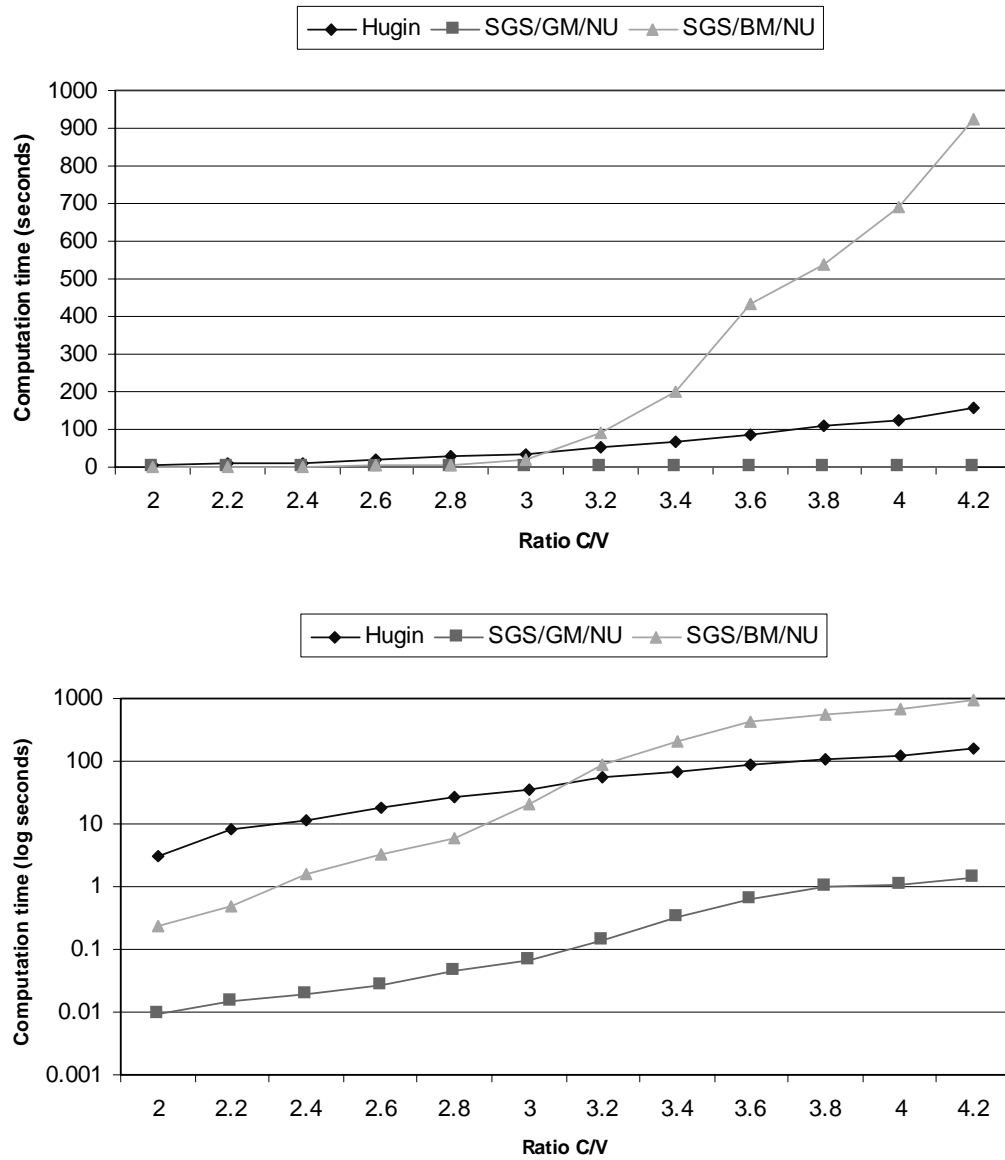


Figure 6.2: MPE computation time of Hugin and SGS on SAT BNs with $V = 30$ variables and $C = 60$ to $C = 126$ clauses.

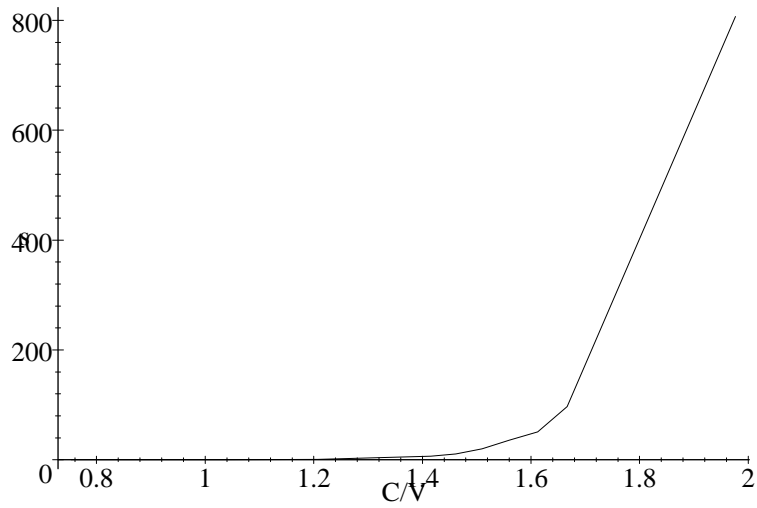


Figure 6.3: Computation time (seconds) as function of C/V ratio in Hugin for KD networks. Note the phase transition around $C/V \approx 1.7$, supporting our claim that other models than the SAT model have phase transitions.

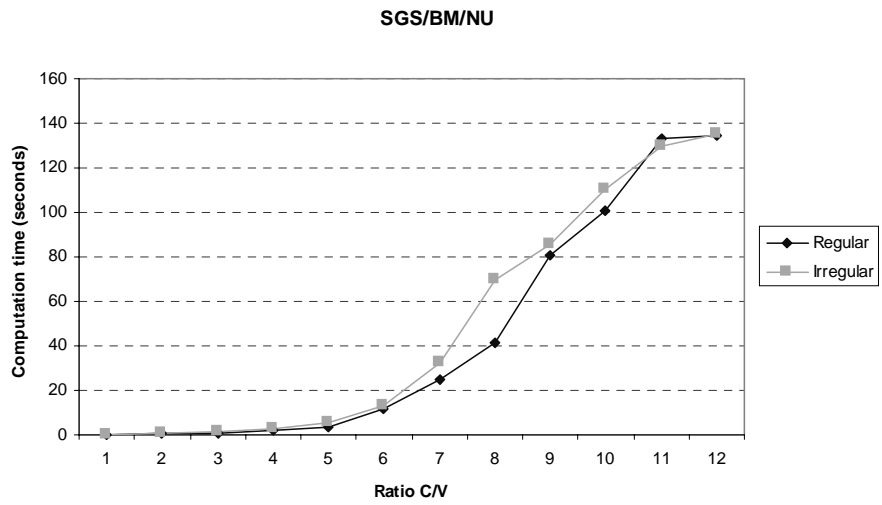
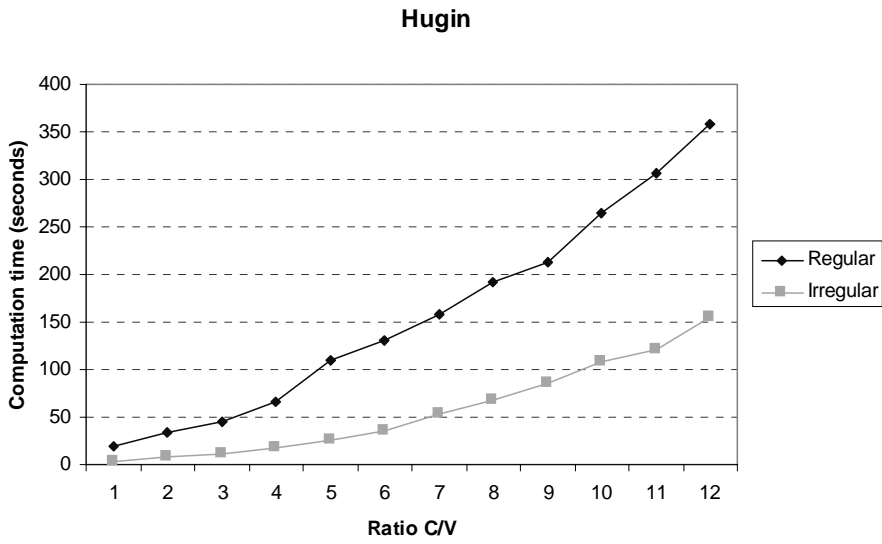


Figure 6.4: Computation time (seconds) as function of C/V ratio for irregular and regular SAT networks. Results for Hugin are shown at the top, results for SGS to the bottom.

Chapter 7

Abstraction for Computing the Most Probable Explanation

Abstraction hierarchies have seen widespread use within many areas of artificial intelligence. There are several purposes of abstraction, including (i) improving human comprehension and interaction [Chang and Fung, 1991], (ii) facilitating learning and knowledge acquisition [Koza, 1989], and (iii) improving the speed or quality of search [Wellman and Liu, 1994] [Mengshoel and Wilkins, 1998a].

Improving the speed or quality of search for a most probable explanation in a Bayesian network (BN) [Pearl, 1988] is the main motivation for the research reported in this chapter. Exactly computing a most probable explanation (MPE) is computationally hard [Shimony, 1994], and the large node state spaces that occur frequently in application BNs have proven to be a major factor in slowing down the speed of inference. In addition to being directly used, large state spaces can be caused by a discretization of continuous nodes.

To attack the problem of large state spaces slowing down the speed of inference, we abstract these large node state spaces. We introduce the perspective that an abstracted state in a search space can be regarded as a noisy approximation of the sets of states that it abstracts. This perspective is valuable because stochastic search algorithms such as genetic algorithms (GAs) [Goldberg, 1989c] can be augmented with abstraction and refinement when computing a most probable explanation [Mengshoel and Wilkins, 1998a], and GAs have proven robust even when using noisy fitness functions [Goldberg, 1989c] [Miller, 1997].

Our approach relies on an ability to construct abstraction hierarchies of high quality, and this is a main focus of the present chapter. This chapter synthesizes and extends research from several areas of artificial intelligence. In particular, we extend previous work on BN abstraction and

refinement [Chang and Fung, 1991] [Wellman and Liu, 1994] [Liu and Wellman, 1996] [Mengshoel and Wilkins, 1998a] by focusing on MPE computation, identifying quality criteria for abstraction hierarchies, developing algorithms for abstraction hierarchy construction, and empirically testing them. For a stochastic searcher such as a GA, the goal of using abstraction and refinement for search is more reliable convergence or increased speed of handling more abstract (hence coarser) individuals, and future research should investigate these issues.

The remainder of this chapter is organized as follows. Concepts related to Bayesian network and GA approximation are presented in Section 7.1. We focus on abstraction and refinement in Bayesian networks in Section 7.2. We discuss abstraction for the MPE task in Section 7.3, and we investigate quality measures for abstraction in Section 7.4. Abstraction algorithms are presented in Section 7.5, and we give experimental results for them in Section 7.6. Section 7.7 concludes and outlines future research. Appendix A gives further details about abstraction and refinement.

7.1 Approximation, Bayesian Networks, and Genetic Algorithms

The central idea in BN approximation is, given an explanation \mathbf{x} to, give an approximate value of $\Pr(\mathbf{x})$ rather than $\Pr(\mathbf{x})$ itself. As detailed later in this chapter, we approximate the explanation \mathbf{x} by another explanation \mathbf{y} , and then use $\Pr(\mathbf{y})$ as an approximation to $\Pr(\mathbf{x})$. For BNs, we will distinguish between abstraction and aggregation, which both have been described in the BN literature [Chang and Fung, 1991] [Wellman and Liu, 1994] [Liu and Wellman, 1996]. *Abstraction* is essentially to replace several node states with one node state. Abstraction is also known as state-space abstraction [Wellman and Liu, 1994] [Liu and Wellman, 1996], coarsening [Chang and Fung, 1991], or behavioral abstraction [Genesereth, 1984]. *Aggregation* is essentially to replace several nodes with one node. Aggregation is also known as structural abstraction [Wellman and Liu, 1994] [Genesereth, 1984] or hierarchical abstraction [Srinivas, 1994]. The inverse operations of abstraction and aggregation, refinement and decomposition, are also of interest; these will be discussed below as will other issues related to abstraction and aggregation in Bayesian networks.

Previous research establishes that GAs can perform well using approximate fitness functions [Grefenstette and Fitzpatrick, 1985] [Miller, 1997] [Chapman and Saitou, 1994]. An abstracted BN is an approximation of the original BN, and is therefore an approximate fitness function. The

advantage of abstraction is that the search space is made smaller and hence easier to search. The price to pay is inaccuracy, which can be considered to be noise, however previous research has shown that GAs are noise-tolerant [Grefenstette and Fitzpatrick, 1985]. In the rest of this section we discuss approximation in Bayesian networks as well as genetic algorithms, approximations, and noise.

7.1.1 Abstraction and Refinement

Chang and Fung introduced the two operations of refine and coarsen for discrete Bayesian networks (BNs) [Chang and Fung, 1991]. Coarsen eliminates states for a node (it is an abstraction operation), while refine introduces new states for a node (it is a refinement operation). Both operations, which are discussed in more detail in Section 7.2, take as input a target node and a desired refinement or coarsening, and then output a revised conditional probability distribution for the target node and for the target node’s children. Both operations are based on constraints on the Markov blanket of the target node. Two classes of operations are described: external operations and internal operations. External operations change the BN topology by using Shachter’s operations [Shachter, 1988]. Internal operations, on the other hand, maintain the BN topology. In the following we focus on the internal abstraction operation.

Wellman and Liu also consider abstraction of a node X , and in particular how the conditional probability tables related to a parent P and a child C are affected. Updating of $\Pr(X | P)$ is similar to Chang and Fung, while updating of $\Pr(C | X)$ is different. Let the states $x_i, \dots, x_j \in \Omega_X$ be abstracted to $x'_{ij} \in \Omega_{X'}$. Wellman and Liu’s abstraction operation (WL abstraction) approximates the conditional probability distribution and is defined as:

$$\Pr(c | x'_{ij}) = \frac{\sum_{l=i}^j \Pr(c | x_l)}{j - i + 1},$$

where $|C(x')| = j - i + 1$. In other words, the $j - i + 1$ states x_i, \dots, x_j in node X are abstracted into one state x'_{ij} in node X' . The approach here is to approximate by averaging over all the conditionals $\Pr(c | x)$ of X .

7.1.2 Aggregation and Decomposition

Previous research on aggregation has been less extensive than that on abstraction, even though in some sense the issues involved are more comprehensive: Aggregation makes ‘larger’ changes to a Bayesian network.

Srinivas investigates the relationship between model-based reasoning and BNs [Srinivas, 1994]. Specifically, he constructed a translation algorithm that creates a BN from a hierarchical functional schematic, and describes how clustering [Lauritzen and Spiegelhalter, 1988] [Jensen et al., 1990b] can be modified to exploit the hierarchy. This work has been extended to include computation of repair plans [Srinivas and Horvitz, 1995]. The basic functional schematic unit is a component C . A component has input, output, and a mode; the mode represents operational status of the unit (e.g. correct, stuck-at-1, or stuck-at-0 for a transistor). Unless it is atomic, a component is recursively decomposed into subcomponents C_1, \dots, C_n . The translation scheme takes a functional schematic and creates a BN consisting of these nodes: I_i represents input i , O represents the output, M_i represents mode i , and X_i represents internal variable i . Internal variables are subcomponent input and output that are not inputs and outputs of the component. At the level of subcomponents C_1, \dots, C_n , we have a BN representing the joint distribution $\Pr(I_1, \dots, I_m, O, M_1, \dots, M_n, X_1, \dots, X_k)$. At a higher level, C is considered an aggregate and the corresponding BN represents $\Pr(I_1, \dots, I_m, O, M)$. Srinivas describes how the higher level distribution is computed from the lower level distribution using Shachter’s topological transformation operations for marginalization [Shachter, 1988].

7.1.3 Approximation and Genetic Algorithms

Approximations have also been explored within the GA community. Approximations can be done at a low level, namely at the level of the GA strings themselves. It has been observed that using individuals that are as fine-grained as they can be might not be optimal. For example, this was the result of using a GA for structural mechanical design [Chapman and Saitou, 1994]. In the research of Chapman et al., a GA was used for conceptual design, in particular structural topology design in mechanical design [Chapman and Saitou, 1994]. A hierarchical subdivision scheme was applied, where an initial optimum was found using a coarse discretization. This initial optimum was subdivided into four subdesigns, and each was optimized pseudo-independently using the same

string length as initially used. The four populations corresponding to the respective four quadrants were initialized with mutated versions of the coarse optimum (the high mutation rate 0.15 was used). Fitness evaluation was done globally, i.e. by combining an individual in the current quadrant with the best individuals in the other three quadrants, and sequentially going through the four quadrants. Compared to not using hierarchical subdivision, a similar conceptual design was found using fewer fitness evaluations (13,500 versus 18,000), where many of the fitness evaluations were less expensive for the hierarchical subdivision scheme.

7.1.4 Noise and Genetic Algorithms

For GAs, there is a trade-off involved in fitness function evaluation. GA fitness function evaluation can be slow and accurate on the one hand or fast and approximate on the other [Grefenstette and Fitzpatrick, 1985]:

Given a fixed amount of computation time, is it better to devote substantial time to getting highly accurate evaluations, or to obtain quick, rough evaluations and run the GA for many more generations?

One form of approximate fitness function evaluation is Monte Carlo sampling. A central question here is how many samples to perform per fitness function evaluation. Grefenstette and Fitzpatrick experimentally decided the optimal number of pixels to be sampled in an image, optimal meaning the number that gave best GA performance [Grefenstette and Fitzpatrick, 1985]. A slightly different perspective is that Monte Carlo sampling induces a noisy fitness function. The impact of such noise on convergence, population sizing, and sampling has been investigated [Miller, 1997].

7.2 Preliminaries: Abstraction and Refinement Operators

In this section, we introduce terminology related to abstraction, first the notion of an abstraction hierarchy. Based on that, we can introduce the terminology that a node is an abstraction (refinement) of another node, and a BN is an abstraction (refinement) of another BN, and similarly for explanations.

7.2.1 Abstraction Hierarchy

An abstraction hierarchy is a DAG which structures the state space of a BN node.

Definition 14 *Let X be a BN node with $\Omega_X = \{x_1, \dots, x_k\}$. An abstraction hierarchy for X is a DAG that is a tree where each leaf node represents some x_i , the root node represents the disjunction of all states $x_1 \vee \dots \vee x_k$, and interior node number i represents a disjunction of j states, $x_{i_1} \vee \dots \vee x_{i_j}$.*

In the following definition we focus on the relations between states in an abstraction hierarchy.

Definition 15 *Let H be an abstraction hierarchy. A ground state is a leaf in H . An abstract state is an interior or a root node in H . A state \check{s} directly refines another state s in H ($\check{s} \in \rho(s)$) if there is a directed edge from \check{s} to s . A state s directly abstracts another state \check{s} in H ($s \in \alpha(\check{s})$) if $\check{s} \in \rho(s)$. A state \check{s} refines a state s in H ($\check{s} \in \rho^*(s)$) if there is a directed path from \check{s} to s or if $s = \check{s}$. A state s abstracts a state \check{s} in H ($s \in \alpha^*(\check{s})$) if $\check{s} \in \rho^*(s)$ in H .*

We pronounce \check{s} and \hat{s} ‘s down’ and ‘s up’ respectively, referring to the abstraction hierarchy. Note that a state s both abstracts and refines itself by these definitions. When we want to exclude this possibility, the terminology strict abstraction and strict refinement is used. In order to simplify the notation, we often do not explicitly mention the abstraction hierarchy H in the following.

Note that an arbitrary set of states $S = \{s_1, \dots, s_k\}$ might or might not be a valid state space. A set of states S is a valid state space if every leaf node in its abstraction hierarchy H has exactly one abstract state in the set S .

The *level* of a state in an abstraction hierarchy is defined inductively as the level of its parent state plus one. The level of the root node is zero. Since we assume there is only one root node, the notion of level is well-defined.

7.2.2 Abstract State Spaces and Nodes

Several of the definitions of Chapter 2 can now be extended to accommodate abstractions and refinements. In particular, we can now think about a state space as abstracting another state space, and a BN node as abstracting another BN node. Also, we need to introduce reasonable ways of changing CPTs when abstracting or refining.

Definition 16 *A state space $S = \{s_1, \dots, s_m\}$ abstracts a state space $\check{S} = \{\check{s}_1, \dots, \check{s}_n\}$ if they have the same abstraction hierarchy H and for all $\check{s} \in \check{S}$ there exists some $s \in S$ such that $\check{s} \in \rho^*(s)$ in H .*

It is sometimes more convenient to talk about a node rather than a node and its state space, thus we will often not mention the state space explicitly. Often we will also consider different nodes, even though they share the same abstraction hierarchy and it is only their state space which differs. An example of this is shown in Figure 7.1, where R_1 , R_2 , and R_3 are essentially variants of the same node.

Figure 7.1 shows a simple example of abstraction and refinement in agent monitoring. Suppose an agent can at any one time be in one among a number of regions (here along a line), and suppose there are 16 regions at the ground level. Let these ground level regions be named from $r_{1,1}$ through $r_{1,16}$. Let the BN node R_1 range over states corresponding to these regions or segments, representing the segment in which the agent is. Then neighboring states can be abstracted, for example states $r_{1,1}$ and $r_{1,2}$ can be abstracted into $r_{2,1}$ as shown in Figure 7.1. Figure 7.2 contains an abstraction hierarchy for this example.

We are in this section primarily concerned with predefined and hierarchical abstraction and refinement. By predefined we mean that the abstraction hierarchy is given. This assumption is lifted in later sections, where we focus on constructing abstraction hierarchies. An example of a domain where a predefined abstraction hierarchy is natural is in the discretization of a continuous, real-valued space which occurs in spatial abstraction. Abstraction, then, amounts to ‘merging’ adjacent states, while refinement is the ‘splitting’ of a state into its component, adjacent, states. In the context of spatial abstraction, hierarchical abstraction means that, once states u_{i_1} and u_{i_2} have been picked, there is no ambiguity in how to abstract.

Several of the definitions of the previous section can now be extended to accommodate abstractions and refinements.

Definition 17 *Let H be an abstraction hierarchy. A ground state space (or ground node) with respect to H is a state space Ω_X (or a node X) where all states are leaf nodes in H . An abstract state space (or node) has some state that is not a leaf in H .*

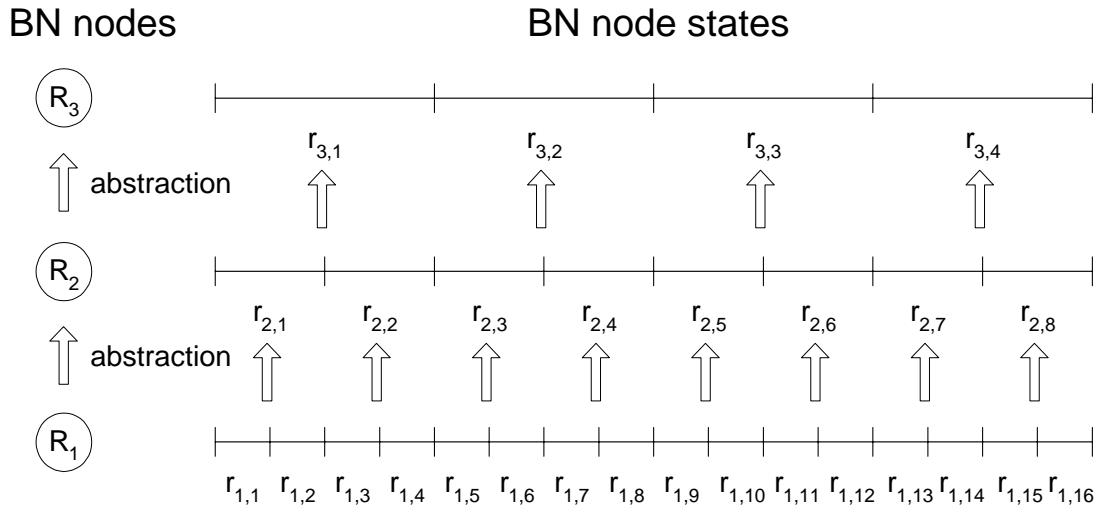


Figure 7.1: Example of state space abstraction and refinement. Node R_1 contains the original states $r_{1,1}$ to $r_{1,16}$. Node R_2 contains states $r_{2,1}$ to $r_{2,8}$, where any state $r_{2,i}$ is an abstraction of two states $r_{1,2i-1}$ and $r_{1,2i}$ in R_1 .

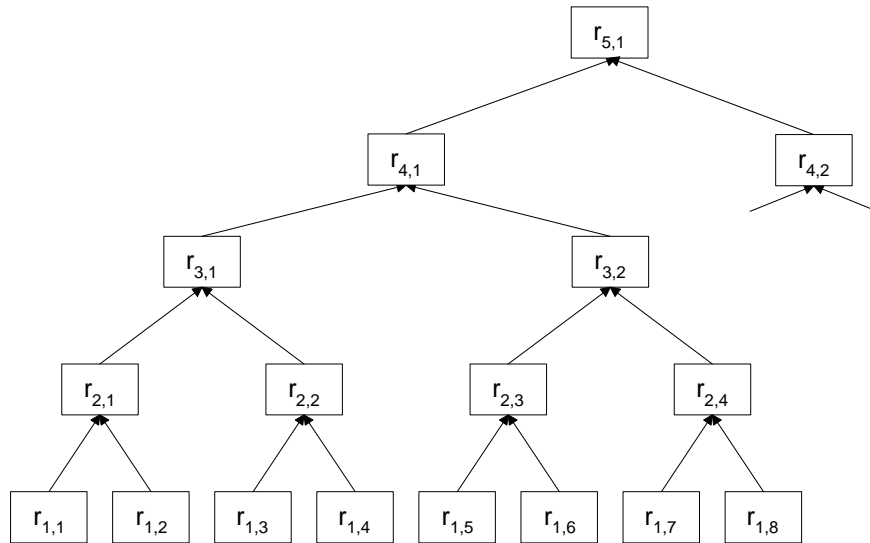


Figure 7.2: Abstraction hierarchy for an agent monitoring node R_i . The abstraction hierarchy rooted at $r_{4,2}$ is similar to the hierarchy rooted at $r_{4,1}$, but has been left out of this figure.

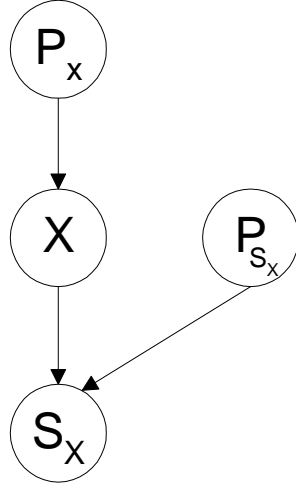


Figure 7.3: Bayesian network showing abstraction and refinement principles.

We adopt the internal abstraction and refinement operators for conditional probability introduced by Chang and Fung [Chang and Fung, 1991]. In our variant of their scheme, the inputs to abstraction and refinement are:

- a node X whose state space Ω_X is to be refined or abstracted.
- a new state space Ω'_X .
- an abstraction hierarchy H , which describes which states $x \in \Omega_X$ are related to which states $x' \in \Omega'_X$.

The outputs are the following:

- a new conditional distribution for X , $\Pr'(X \mid \Pi_X)$
- a new conditional distribution for successors of X , $\Pr'(S_X \mid \Pi_{S_X})$, where S_X is a successor (child) of X , $S_X \in \Psi_X$.

The central idea of Chang and Fung is to keep the effect of abstraction and refinement localized, in particular by keeping the joint distribution of the Markov blanket of the abstracted node intact, if possible. The topology of the BN shown in Figure 7.3 illustrates this; the node X and its Markov

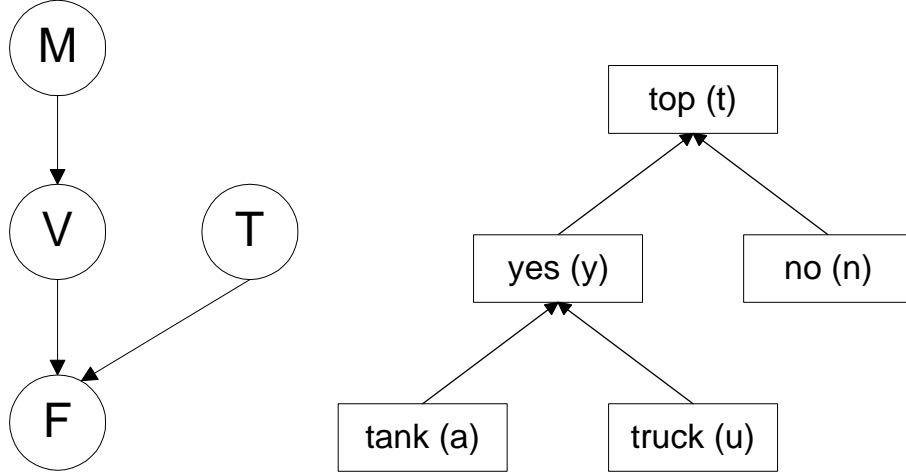


Figure 7.4: Bayesian network with abstraction hierarchy. The Bayesian network, whose node V is abstracted, is shown to the left, the abstraction hierarchy is shown to the right.

blanket is depicted. Their idea leads to the following two types of constraints on abstraction. The *parent constraint* is

$$\Pr(\hat{x} \mid \pi_X) = \sum_{x \in \rho(\hat{x})} \Pr(x \mid \pi_X), \quad (7.1)$$

while the *child constraint* is

$$\Pr(s_X \mid \hat{\pi}_{S_X}) \Pr(\hat{x} \mid \pi_X) = \sum_{x \in \rho(\hat{x})} \Pr(s_X \mid \pi_{S_X}) \Pr(x \mid \pi_X). \quad (7.2)$$

Here, $s_X \in \Omega_{S_X}$, where we recall that $S_X \in \Psi_X$. An original state is x , while the abstracted state is \hat{x} , where we have $x \in \pi_{S_X}$ and $\hat{x} \in \hat{\pi}_{S_X}$.

There are several ways to use the parent and child constraints for abstraction and refinement. Given an abstraction hierarchy, they can be used to compute CPTs for an abstract node. The constraints are used that way in this section; in this case the following form of (7.2) is useful:

$$\Pr(s_X \mid \hat{\pi}_{S_X}) = \frac{\sum_{x \in \rho(\hat{x})} \Pr(s_X \mid \pi_{S_X}) \Pr(x \mid \pi_X)}{\Pr(\hat{x} \mid \pi_X)}.$$

Given an abstraction hierarchy and the CPT entry $\Pr(s_X \mid \pi_{S_X})$, we can compute $\Pr(s_X \mid \hat{\pi}_{S_X})$.

When constructing an abstraction hierarchy, we argue that among the parent constraint (7.1) and the child constraint (7.2), the parent constraint is most important, since the parent probabilities $\Pr(x | \pi_X)$ and $\Pr(\hat{x} | \pi_X)$ also participate in the child constraint.

An example Bayesian network, borrowed from [Chang and Fung, 1991, Figure 7] and used in the following is shown in Figure 7.4. Here, M stands for military unit type, with $\Omega_M = \{b, c\}$. V stands for a vehicle in a particular place at a particular time, with $\Omega_V = \{y, n\}$ (yes or no) or $\Omega_V = \{a, u, n\}$ (tank, truck, or no). These two state spaces are represented as an abstraction hierarchy in Figure 7.4. T stands for terrain conditions, with $\Omega_T = \{g, b\}$ (good or bad). F is feature, $\Omega_F = \{a, b, o\}$. In this network, V 's state space can be abstracted from $\Omega_V = \{a, u, n\}$ to $\Omega_V = \{y, n\}$, so $\rho(\hat{x}) = \rho(y) = \{a, u\}$, while $\alpha(\{a, u\}) = y$; see the abstraction hierarchy in Figure 7.4. Conditional probability tables for this example are shown in Table 7.1 and Table 7.2.

As an example, let us consider how to compute probability values in two closely related cases, Case (i) and Case (ii). Case (i) and Case (ii) are for one particular abstraction hierarchy. Case (i): Suppose $V = y$, $M = b$, $F = a$, and $T = g$ for the second child constraint above. For the right hand side of the child constraint we then get:

$$\begin{aligned} \frac{\sum_{x \in \rho(\hat{x})} \Pr(a | x, g) \Pr(x | b)}{\Pr(y | b)} &= \frac{(\Pr(a | a, g) \Pr(a | b) + \Pr(a | u, g) \Pr(u | b))}{\Pr(y | b)} & (7.3) \\ &= \frac{(0.4 \times 0.3 + 0.6 \times 0.5)}{0.8} \\ &= 0.525. \end{aligned}$$

Case (ii): Let $M = c$, with the other nodes instantiated as above (i.e. $V = y$, $F = a$, and $T = g$):

$$\begin{aligned} \frac{\sum_{x \in \rho(\hat{x})} \Pr(a | x, g) \Pr(x | c)}{\Pr(y | c)} &= \frac{(\Pr(a | a, g) \Pr(a | c) + \Pr(a | u, g) \Pr(u | c))}{\Pr(y | b)} & (7.4) \\ &= 0.55. \end{aligned}$$

Now take the average of the values computed in Equation 7.3 and Equation 7.4; this gives the abstracted value

Pr($F V, T$)						
$V:$	a		u		n	
$T:$	g	b	g	b	g	b
a	0.4	0.1	0.6	0.4	0.1	0.2
b	0.5	0.5	0.3	0.2	0.1	0.2
o	0.1	0.4	0.1	0.4	0.8	0.6

Pr($F V, T$)				
$V:$	y		n	
$T:$	g	b	g	b
a	0.5375	0.30625	0.10	0.20
b	0.3625	0.29375	0.10	0.20
o	0.1000	0.40000	0.80	0.60

Table 7.1: To the left, F 's conditional probability table is shown before abstraction of $\Omega_V = a, u, n$ to $\Omega_V = y, n$, to the right it is shown after abstraction.

Pr($V M$)		
$M:$	b	c
a	0.3	0.1
u	0.5	0.3
n	0.2	0.6

Pr($V M$)		
$M:$	b	c
y	0.8	0.4
n	0.2	0.6

Table 7.2: To the left V 's CPT is shown before abstraction of $\Omega_V = a, u, n$ to $\Omega_V = y, n$, to the right it is shown after abstraction. Notice how these distributions are changed.

$$\Pr(a | y, g) = \frac{0.525 + 0.55}{2} = 0.5375 \tag{7.5}$$

as shown in Table 7.1, more specifically in the CPT to the right, in the column with $V = y$ and $T = g$, and the row with $F = a$. The other conditional probability values in this CPT can be computed in a similar way.

In addition to abstraction operators, refinement operators have been introduced and can be used to perform what we denote *straight refinement* (see Appendix A). An alternative to using the refinement operators is to perform what we denote *refinement by abstraction*. Refinement by abstraction just means that if we have an abstract node \hat{V} , and want to refine to node V , we can do this by abstracting V from the ground node \check{V} , $V = \alpha^*(\check{V})$. Refinement by abstraction is always at least as good as straight refinement, and in the following we restrict ourselves to refinement by abstraction for that reason.

7.2.3 Abstract BNs and Explanations

The notions of BNs and explanations can be extended into abstract BNs and abstract explanations as follows.

Definition 18 *A Bayesian network with abstraction hierarchies is a tuple $(\mathbf{V}, \mathbf{E}, \mathbf{Pr}, \mathbf{H})$, where $(\mathbf{V}, \mathbf{E}, \mathbf{Pr})$ is a Bayesian network, \mathbf{H} a set of abstraction hierarchies.*

We are interested in abstraction because it can be done in polynomial time, by iterating over all the nodes in a BN, using the appropriate abstraction operator(s) as described above.

We can introduce abstraction relations between BNs, similar to for nodes and state spaces. Given an abstracted Bayesian network $\hat{B} \in \alpha^*(B)$, we can consider the situation where an explanation $\hat{\mathbf{x}}$ in \hat{B} abstracts an explanation \mathbf{x} in B , so these explanations are closely related through the abstraction hierarchies.

Definition 19 *A ground explanation is an explanation where all states are ground. An abstract explanation is an explanation where some state is abstract. An explanation $\hat{\mathbf{x}}$ abstracts an explanation \mathbf{x} , $\hat{\mathbf{x}} \in \alpha^*(\mathbf{x})$, if $\hat{\mathbf{x}}$ is an explanation in \hat{B} , \mathbf{x} is an explanation in B , and $\hat{B} \in \alpha^*(B)$.*

Along similar lines, we have that an explanation $\check{\mathbf{x}}$ is said to refine another explanation \mathbf{x} , $\check{\mathbf{x}} \in \rho^*(\mathbf{x})$.

Having introduced the terminology for abstraction and refinement, we turn to how it can and should be done for the purposes of the MPE task in the next sections.

7.3 Abstraction for the MPE Task

Suppose we have a BN A which is an abstraction of another BN R , or $A \in \alpha^*(R)$. There are two ways in which BN A is of interest, despite the fact that R is (at least by our assumptions here) a more accurate model. (i) First, A can in some cases be an end-result in itself; for example the BN R may contain many distinctions that are not considered relevant to the particular task at hand. In this case, the abstract BN A summarizes aspects of the refined BN R , or one is willing to use an abstracted BN, despite loss in accuracy, because it is faster. (ii) Second, the abstract BN can be considered to be a stepping-stone for the refined BN. In this case, the abstract BN is used to

predict certain aspects of the refined BN. For instance, ideally we would like an MPE in the refined BN to abstract to an explanation that is an MPE in the abstract BN, and vice versa. In both (i) and (ii), the quality of abstractions matters, and advantages and disadvantages of abstraction is what we turn to next.

7.3.1 State Space Size

A first advantage of abstraction is that the search space in an abstracted BN will be smaller than that from which it is abstracted. Suppose a BN contains nodes V_1, \dots, V_n . Then the cardinality of the search space of the original BN is:

$$|\Omega_{V_1}| \times \dots \times |\Omega_{V_n}|,$$

while the search space of an abstracted BN is

$$|\Omega_{\hat{V}_1}| \times \dots \times |\Omega_{\hat{V}_n}|,$$

where \hat{V}_i is the abstracted node of the node V_i in the original BN. If a BN contains ten nodes, each with ten states, and all abstracted nodes contain two states, this gives a ratio between the BN state spaces of:

$$\frac{2^{10}}{10^{10}} = \frac{1024}{1\,000\,000\,000\,000} = 0.0000001024,$$

which is a substantial search space reduction.

The second advantage of using an abstracted BN is that the operations on explanations, or individuals in the GA context, will be faster. This is mainly because fitness of individuals can be computed faster but also because other operations might be faster, since individuals have smaller state spaces.

Let \mathbf{x} and $\tilde{\mathbf{x}}$ be explanations, and suppose $\tilde{\mathbf{x}}$ refines \mathbf{x} . Using Equation 2.1 we can compute both $\Pr(\tilde{\mathbf{x}})$ and $\Pr(\mathbf{x})$, and $\Pr(\mathbf{x})$ estimates $\Pr(\tilde{\mathbf{x}})$. This estimate places several explanations into the same equivalence class as far as their probability goes. The number of explanations in each

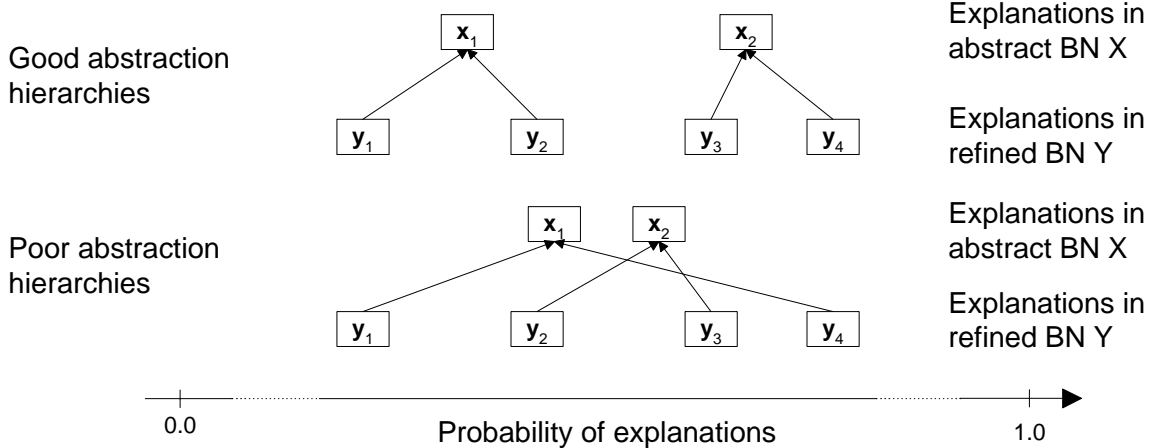


Figure 7.5: Ordering of explanations according to their probability. In the top part of the figure, an example ordering for high quality abstraction hierarchies is shown. In the bottom part, an example ordering for low quality abstraction is shown.

equivalence class, which is just the difference in cardinality for each node’s state space, needs to be accounted for. For this, consider an abstract explanation \mathbf{x} in BN B , and the explanations $\check{\mathbf{x}}_1, \dots, \check{\mathbf{x}}_g$ it refines to in BN \check{B} , where $\check{B} \in \rho^*(B)$. Here g can be computed as follows:

$$g = \prod_{i=1}^n \frac{|\Omega_{\check{X}_i}|}{|\Omega_{X_i}|} = \prod_{i=1}^n g_i, \quad (7.6)$$

Recall that \check{X}_i is refined from X_i , and so $|\Omega_{\check{X}_i}| \geq |\Omega_{X_i}|$, and therefore $g \geq 1$. The measure g indicates how much the search space is expanded by refinement.

Two closely related perspectives on abstraction quality exist, one focuses on the abstraction hierarchies themselves, the other focuses on the explanations. Ideally, the explanation perspective should drive the abstraction hierarchy perspective. Abstraction hierarchies are good, for our purposes, if they make it easier to compute an MPE. In the following subsection, we focus on how to maintain, if possible, MPEs when doing abstraction.

7.3.2 Quality of Abstraction

For computing the MPE, correctness of the explanation and not necessarily correctness of the probability of the explanation is important. One can argue that for decision making purposes, the most important question is whether a system computes the correct MPE, and whether the

probability is correct is less important. More formally, let \mathbf{x} and \mathbf{y} be explanations, and let $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ be their respective abstractions. What we want to avoid is $\Pr(\mathbf{x}) > \Pr(\mathbf{y})$ while $\Pr(\hat{\mathbf{x}}) < \Pr(\hat{\mathbf{y}})$, or $\Pr(\hat{\mathbf{x}}) > \Pr(\hat{\mathbf{y}})$ while $\Pr(\mathbf{x}) < \Pr(\mathbf{y})$. In order to avoid this, a possible gold standard is this: If $\Pr(\mathbf{x}_1) \geq \Pr(\mathbf{x}_2) \geq \dots \geq \Pr(\mathbf{x}_m)$, then $\Pr(\hat{\mathbf{x}}_1) \geq \Pr(\hat{\mathbf{x}}_2) \geq \dots \geq \Pr(\hat{\mathbf{x}}_m)$. Here, m is the number of explanations in the BN. That is, the abstraction process should not change the ordering, according to probability, of the explanations. This principle is illustrated in Figure 7.5. A question is whether one should require this for all explanations rather than just for explanations of high probability. A danger of focusing on the quality of high-probability explanations only is that they might be inconsistent with evidence, and in such cases lower-probability explanations will be MPEs. On the other hand, if low probability explanations are MPEs, this means there is evidence and the search space is smaller. So focusing on the quality of low-probability explanations as much as that of high-probability explanations seems unnecessary. For low-probability explanations, one can afford to refine to lower-level states in the abstraction hierarchy.

An alternative perspective on the quality of abstraction is to focus on probabilities of abstracted explanations, and this is what we do next. We distinguish between two qualities of an abstract explanation \mathbf{x} . The *summarizing* quality of \mathbf{x} concerns how close $\Pr(\mathbf{x})$ is to $\Pr(\check{\mathbf{x}}_1) + \dots + \Pr(\check{\mathbf{x}}_g)$. Clearly,

$$\Pr(\mathbf{x}) = \Pr(\check{\mathbf{x}}_1) + \dots + \Pr(\check{\mathbf{x}}_g) \quad (7.7)$$

is the gold standard, we call this perfect summarizing abstraction. The *predictive* quality of \mathbf{x} concerns how closely $\Pr(\mathbf{x})$ predicts the probability of $\check{\mathbf{x}}_i$, $\Pr(\check{\mathbf{x}}_i)$, where $i \in [1, g]$, and g is the total number of explanations which \mathbf{x} refines to in \check{B} . We introduce the notion of an estimate $\widehat{\Pr}(\check{\mathbf{x}})$ of the true probability $\Pr(\mathbf{x}_i)$:

$$\widehat{\Pr}(\check{\mathbf{x}}) = \Pr(\mathbf{x})/g. \quad (7.8)$$

Clearly,

$$\widehat{\Pr}(\check{\mathbf{x}}) = \Pr(\check{\mathbf{x}}_i) \quad (7.9)$$

for all $i \in [1, g]$ is the ideal situation, we call this perfect predictive abstraction. More realistic, when Equation 7.9 cannot be attained, is to minimize the deviation

$$\sum_{i=1}^g \left(\widehat{\Pr}(\tilde{\mathbf{x}}) - \Pr(\tilde{\mathbf{x}}_i) \right)^2 = \sum_{i=1}^g \left(\frac{\Pr(\mathbf{x})}{g} - \Pr(\tilde{\mathbf{x}}_i) \right)^2$$

for a given abstract explanation \mathbf{x} , with $\tilde{\mathbf{x}}_i \in \rho^*(\mathbf{x})$. Now, we need to do this for all m explanations in an abstracted BN, which gives us

$$\sum_{j=1}^m \sum_{i=1}^g \left(\widehat{\Pr}(\tilde{\mathbf{x}}_j) - \Pr(\tilde{\mathbf{x}}_{j_i}) \right)^2 = \sum_{j=1}^m \sum_{i=1}^g \left(\frac{\Pr(\mathbf{x}_j)}{g} - \Pr(\tilde{\mathbf{x}}_{j_i}) \right)^2, \quad (7.10)$$

where \mathbf{x}_j is the abstracted BN's j -th explanation, and $\tilde{\mathbf{x}}_{j_i}$ is the i -th explanation which \mathbf{x}_j refines to. Now, we want to minimize the sum in Equation 7.10. Clearly, enumerating all possible BNs and computing this sum explicitly over all explanations in order to minimize is computationally prohibitive. Still, the above equations provide idealized conditions which we can try to approximate, and we will return to this later in this chapter. If we can estimate the explanation probability reliably, then we can also find the MPE. Therefore, we focus on minimizing probability error in the following.

7.3.3 Difficulty of Obtaining Correctness

The problem with the quality measures and the gold standards above is that they cannot be efficiently computed, since computing the MPE is NP-hard. And even if it could be computed in $O(1)$ time, there is an exponential number of explanations in a BN, so for BNs of interesting size, one cannot compute. However, what one can do is to exploit the BN structure, using various heuristic methods of computing abstracted BNs and hence abstract explanations, and then perform measurements over the abstract and refined explanations, seeing how well they perform with respect to the quality criteria. This is the approach taken in this chapter. We will return to this issue in Section 7.6, which contains empirical results.

However, before we get to that we need to consider the basis of our abstraction algorithms. In particular, we consider abstraction as a noisy process, and decompose explanation noise into node noise and network noise. These topics are explored in the next section.

Hierarchy:	A, B, C	A	B	C	A, B, C
Level:	2	1	1	1	0
x_1	0.1	0.15	0.2	0.2	0.25
x_2	0.2	0.15	0.2	0.2	0.25
x_3	0.3	0.35	0.3	0.2	0.25
x_4	0.4	0.35	0.3	0.4	0.25
$\phi_{j,k}^2$	0	0.0025	0.005	0.005	0.0125
φ_j^2	0.0125	0.01	0.0025	0.0075	0

Table 7.3: Abstraction hierarchies A , B , and C with internal and external node variances for different levels.

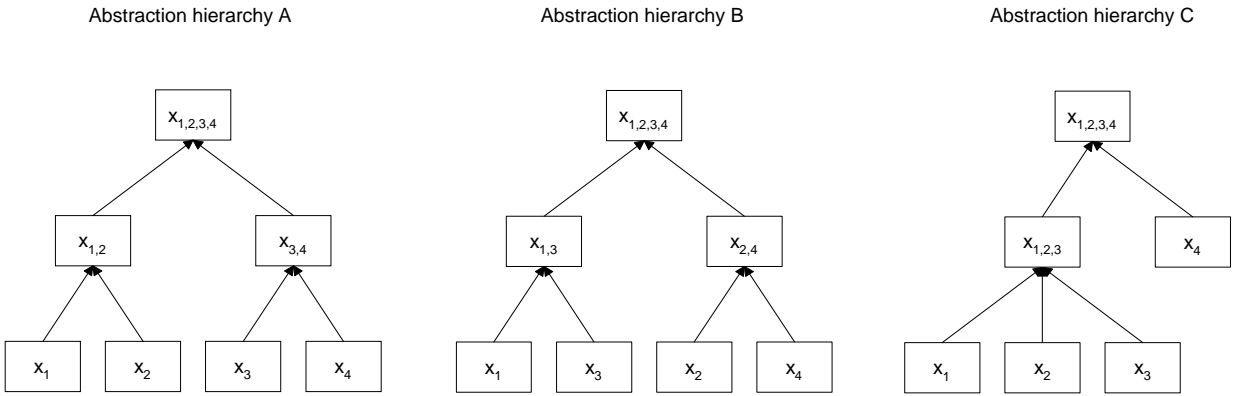


Figure 7.6: Three possible abstraction hierarchies A , B , and C constructed for a BN node X .

7.4 Abstraction Quality

Equation 7.7 and Equation 7.9 describe gold standards for abstraction. However, even $\widehat{\Pr}(\tilde{\mathbf{x}}) \approx \Pr(\tilde{\mathbf{x}}_i)$ is tolerable, and in general the question is what the difference $\Delta p(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}_i) = |\widehat{\Pr}(\tilde{\mathbf{x}}) - \Pr(\tilde{\mathbf{x}}_i)|$ is. The difference $\Delta p(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}_i)$ can be regarded as noise, explanation noise. Note that these notions can easily be generalized from explanations and probabilities to arbitrary individuals and fitness functions for GAs and stochastic search algorithms in general. In the following we discuss node noise and network noise; the focus is on individual nodes and the complete network respectively.

7.4.1 Node Noise Variance

The notion of node noise variance is probably best understood by means of an example. Suppose we have a BN consisting of one node X , with ground states x_1, x_2, x_3 , and x_4 . Let $\Pr(x_1) =$

0.1, $\Pr(x_2) = 0.2$, $\Pr(x_3) = 0.3$, and $\Pr(x_4) = 0.4$, see Table 7.3. Three possible ways of constructing abstraction hierarchies are illustrated in Figure 7.6. A first abstraction hierarchy A is where x_1 and x_2 are abstracted into $x_{1,2}$ with $\Pr(x_{1,2}) = 0.3$, x_3 and x_4 into $x_{3,4}$ with $\Pr(x_{3,4}) = 0.7$, and root state $x_{1,2,3,4}$ with $\Pr(x_{1,2,3,4}) = 1$. Using Equation 7.6 and Equation 7.9, we can now estimate probabilities, see Table 7.3 for results.

Now we introduce the notions of external and internal variance, $\phi_{j,k}^2$ and φ_j^2 respectively:

$$\begin{aligned}\phi_{j,k}^2 &= \frac{1}{s} \sum_{i=1}^s \left(\widehat{\Pr}(x_i | \pi_{X_i}) - \widehat{\Pr}(y_i | \pi_{Y_i}) \right)^2 \\ \varphi_j^2 &= \frac{1}{s} \sum_{i=1}^s \left(\widehat{\Pr}(x_i | \pi_{X_i}) - \frac{1}{s} \right)^2,\end{aligned}$$

Here s is the number of explanations and the conditional probability distributions $\widehat{\Pr}(x_i | \pi_{X_i})$ and $\widehat{\Pr}(y_i | \pi_{Y_i})$ are from levels j and k respectively, with $k < j$. External variance compares conditional probabilities at two levels, while internal variance is a measure of the deviation from a uniform distribution at a particular level. We want external variance to be as low as possible, to minimize difference between abstraction levels, and internal variance to be as high as possible, to separate between states at the same abstraction level as much as possible. In the following external variance $\phi_{j,k}^2$ is always with respect to the ground level, in which case we leave implicit that level and just say ϕ_j^2 . An example computation for abstraction hierarchy A is:

$$\phi_{1,2}^2 = \frac{(0.15 - 0.1)^2 + (0.15 - 0.2)^2 + (0.35 - 0.3)^2 + (0.35 - 0.4)^2}{4} = 0.0025.$$

A second abstraction hierarchy B is $x_{1,3}$ with $\Pr(x_{1,3}) = 0.4$, and $x_{2,4}$ with $\Pr(x_{2,4}) = 0.6$. For computing the MPE, one could argue that it is important to have small external variance for states with more probability mass, and thus introduce an unbalanced abstraction hierarchy C : At level 1 we have nodes x_4 and $x_{1,2,3}$ with $\Pr(x_4) = 0.4$ and $\Pr(x_{1,2,3}) = 0.6$. See Table 7.3 for computations of external and internal variances.

The idea with variance is that a better abstraction hierarchy has generally low external variance $\phi_{j,k}^2$ and high internal variance φ_j^2 . These quality measures can be used to compare different abstraction hierarchies for a given node. In Table 7.3 internal and external variance is presented

for the three example hierarchies. A is best, C second best, and B third best. A is better than B because the difference between the estimated probability and the true probability for x_1 (say) is smaller for A at the first abstraction level (with $x_{1,2}$ and $x_{3,4}$) than it is for B (with $x_{1,3}$ and $x_{2,4}$). In our terminology, the external variance is smaller for A .

7.4.2 Bounding the Node Noise

In algorithms for constructing abstraction hierarchies, one explicitly or implicitly needs to make trade-off between various parameters such as the error c introduced when abstracting, the degree k of nodes in the abstraction hierarchy, and the number of levels ℓ in the abstraction hierarchy. Here, we investigate how these parameters are related under certain idealized conditions. These idealized conditions are (i) we assume that the Markov blanket is maintained perfectly, and (ii) the in-degree of non-leaf nodes in the abstraction hierarchy is a constant k .

Suppose that $\hat{\mathbf{x}}$ is a direct abstraction of \mathbf{x} , and that they only differ with respect to one node X . And further suppose that the difference in conditional probabilities for one particular parent instantiation is bounded by a constant c . That is, in the parent constraint of Equation 7.1, one of the states, x_m , is such that probabilities of all other states x are bounded as follows:

$$\Pr(x_m | \pi_X) - \frac{c}{2} \leq \Pr(x | \pi_X) \leq \Pr(x_m | \pi_X) + \frac{c}{2}.$$

See the calculation in Equation 7.5 for an example. Suppose that k states are being abstracted here. Then the sum will be bounded

$$k \left(\Pr(x_m | \pi_X) - \frac{c}{2} \right) \leq \sum_{x \in \rho(\hat{x})} \Pr(x | \pi_X) \leq k \left(\Pr(x_m | \pi_X) + \frac{c}{2} \right).$$

Substituting using Equation 7.1 we attain

$$k \left(\Pr(x_m | \pi_X) - \frac{c}{2} \right) \leq \Pr(x' | \pi_X) \leq k \left(\Pr(x_m | \pi_X) + \frac{c}{2} \right).$$

Suppose there are multiple levels of abstraction, in particular that x' has parent x'' in the

abstraction hierarchy, giving:

$$k \left(\Pr(x'_m | \pi_X) - \frac{c}{2} \right) \leq \Pr(x'' | \pi_X) \leq k \left(\Pr(x'_m | \pi_X) + \frac{c}{2} \right).$$

Combining these two equations, we get

$$k^2 \Pr(x_m | \pi_X) - \frac{c}{2}(k + k^2) \leq \Pr(x'' | \pi_X) \leq k^2 \Pr(x_m | \pi_X) + \frac{c}{2}(k + k^2).$$

In general we have

$$k^\ell \Pr(x_m | \pi_X) - d \leq \Pr(x^\ell | \pi_X) \leq k^\ell \Pr(x_m | \pi_X) + d,$$

where

$$d = \frac{c(k^{\ell+1} - k)}{2(k - 1)}. \quad (7.11)$$

Here, c is the error, k is the degree of nodes in the abstraction hierarchy, and ℓ is the number of levels. We want to make the bound as tight as possible, in other words minimize d . In order to do that, we need to minimize c , k , and ℓ . Obviously, there are trade-offs involved in doing so. For example, having a small number of levels will give a high node degree. Typically, different algorithms for hierarchy construction make different trade-offs for these parameters, as we will see in the following.

7.4.3 Network Noise

So far, we have considered BN nodes, however the interaction between nodes in the BN is also of interest when computing an MPE. We can make distinction between internal variance and external variance here as well. Network internal variance is for one particular BN. Network external variance, on the other hand, compares BNs at different abstraction levels.

The question is, how does the abstraction process affect this? Using states high in up in abstraction hierarchies, the network internal variance will be small. As we move down in the abstraction hierarchies of a BN, the network internal variance will increase, and it will be at its maximum

when the network is ground. Network external variance, however, will be high at high levels of abstraction, and low at low levels of abstraction.

A key point is also that the node variances (internal and external) carry over to the network variances. High node external variance gives high network external variance, and high node internal variance gives high network internal variance. As an approximation, one can assume that a parametric probability distribution is followed.

7.5 Abstraction Algorithms

So far, we have assumed that abstraction hierarchies exist; this section presents algorithms for constructing them. We present three algorithms: minimal distance abstraction, minimal error abstraction, and random abstraction. The latter is a base-line algorithm to check that the ‘real’ algorithms do in fact help beyond constructing abstraction hierarchies at random.

Abstraction hierarchies are constructed off-line, and then used on-line during computation. Clearly, it is computationally hard to have algorithms optimize all of the abstraction hierarchy parameters introduced in the previous section, however as long as ‘reasonably’ good abstraction hierarchies are constructed, noise-resilient searchers such as GAs can still make good use of the hierarchies. Still, we would like to construct abstraction hierarchies of good quality.

The following greedy algorithm, which we call *minimal distance abstraction* (MinDistance), is one algorithm for abstraction hierarchy construction, see Figure 7.7. The algorithm works by picking two states s and s' of a node S for abstraction, minimizing the distance $d(s, s')$.

Various distance measures may be used. An obvious distance measure is sum of squares. For node S , we sum over all conditional distributions for the states s and s' as follows:

$$d(s, s') = \sum_{\pi_S} (\Pr(s | \pi_S) - \Pr(s' | \pi_S))^2, \quad (7.12)$$

where π_S is a parent instantiation of S . In other words, π_S is a parent instantiations, and we sum the squared difference over all parent instantiations for the node S . Now, do this for all pairs of states s, s' . For example, if $\Omega_S = \{s_1, s_2, s_3\}$, we compute (7.12) for all possible pairs (s_1, s_2) , (s_1, s_3) , and (s_2, s_3) . Using this measure, we find d_{\min} , v_1 , and v_2 , where d_{\min} is the minimal

```

MinDistance( $S$ )
  Input:  $S$  BN node
  Output:  $A$  abstraction hierarchy
   $A \leftarrow$  empty hierarchy
   $v_1 \leftarrow s_1$ 
   $v_2 \leftarrow s_2$ 
  repeat
    for each state  $s \in \Omega_V$ 
      for each state  $s' \in \Omega_V - s$ 
         $d \leftarrow d(s, s')$ 
        if ( $d < d_{\min}$ )
           $d_{\min} \leftarrow d$ 
           $v_1 \leftarrow s$ 
           $v_2 \leftarrow s'$ 
        endif
      endfor
    endfor
     $a \leftarrow$  abstract  $v_1$  and  $v_2$ 
     $A \leftarrow A$  updated with  $a$ 
  until only one state in  $S$ 

```

Figure 7.7: The MinDistance abstraction hierarchy construction algorithm.

distance and v_1 and v_2 are the states among all possible states that give d_{\min} .

A weakness of minimal distance abstraction is that there might be several states with very similar conditional distributions, and so one might want to merge all of these rather than enforcing a binary out-degree on internal nodes in the abstraction hierarchy. This observation leads to an alternative algorithm, *minimal error abstraction* (MinError), which is based on MinDistance: Start with the two states s and s' which are closest according to the distance measure $d(s, s')$. In addition, add all states which are within a distance c , where c is a constant.

Finally, there is *random abstraction* (Random), which just picks states at random and merge them into an abstract state. This process is repeated until just one state, the root state, is left.

Note that there are several other reasonable algorithms for abstraction hierarchy construction; indeed part of the challenge here is to strike a balance between optimizing the different parameters that have an impact on the quality of abstraction. For example, one can try to make the abstraction hierarchy as balanced as possible, but vary the out-degree of abstract states. The idea with this algorithm is to minimize the height of the abstraction hierarchy. Or one can start by picking the state s whose conditional distribution is smallest on average. Merge to s all other states within

Algorithm	BN node	External variance ϕ^2	Internal variance φ^2	Level ℓ
Random	V_1	0.160	0.206	4
MinDistance	V_1	0.128	0.291	6
MinError	V_1	0.128	0.291	6
Random	V_2	0.315	0.457	4
MinDistance	V_2	0.185	0.508	6
MinError	V_2	0.185	0.508	6
Random	V_3	0.444	0.559	3
MinDistance	V_3	0.261	0.568	5
MinError	V_3	0.249	0.666	3
Random	V_4	0.635	0.756	4
MinDistance	V_4	0.227	0.786	9
MinError	V_4	0.169	0.762	6
Random	V_5	0.160	0.173	3
MinDistance	V_5	0.097	0.184	4
MinError	V_5	0.097	0.184	4
Random	V_6	0.169	0.137	3
MinDistance	V_6	0.137	0.296	5
MinError	V_6	0.137	0.296	5

Table 7.4: Results from using different approaches to constructing abstraction hierarchies on different BN nodes. External variance is between the given abstraction level and the ground level.

a certain tolerance, and create an abstracted state from all these states. If no other states than s exists within the tolerance, increase it and try again. The idea with this algorithm is to place states with higher average probability higher up in the abstraction hierarchy.

In order to get a better understanding of how these algorithms behave, we perform experiments as follows.

7.6 Abstraction Experiments

This section presents experiments performed using the abstraction construction algorithms. First, we present experiments that focus on nodes, second experiments that focus on MPEs.

7.6.1 Node Experiments

Here, we characterize abstraction hierarchies constructed using the algorithms of the previous section using the quality measures introduced above. So the types of hierarchies are (i) random

abstraction hierarchies, (ii) minimal distance hierarchies, and (iii) minimal error hierarchies.

In order to work with realistic data, we consider the application BNs Barley and Munin1. Barley has 48 nodes and 84 edges, Munin1 has 189 nodes and 282 edges. Barley is a BN model of Barley crop yields [Kristensen and Rasmussen, 1997b] [Kristensen and Rasmussen, 1997a]. Munin1 is part of an expert electromyography assistant [Andreassen et al., 1987]. These are BNs where many nodes have a high number of states, so that abstraction could be useful. Experiments are performed on nodes V_1, V_2, V_3, V_4, V_5 , and V_6 , see Table 7.4. The node V_1 is Munin1’s R_MEDDD2_AMP_WD node, V_2 is Munin1’s R_APB_MUSIZE, V_3 is Munin1’s R_MYOP_MYDY_APB_MUSIZE, V_4 is Barley’s s2528, V_5 is Barley’s dgv1059, and V_6 is Barley’s aks_vgt.

Results are also presented in Table 7.4. In each row we have a construction algorithm and node combination. The three result columns are ϕ^2 , φ^2 , and ℓ . The numbers for external variance ϕ^2 and internal variance φ^2 are averaged over all levels, and then normalized. Note that the noise numbers are computed over $\ell - 1$ levels; the ground level is omitted. For example, the row ‘MinError, V_3 ’ has 3 levels and mean external variance for the 2 non-ground levels of $6.61744e^{-023}$ and 0.49975 respectively. This gives

$$\phi_{j,k}^2 = (6.61744e^{-023} + 0.49975)/2 = 0.249,$$

as can be found in the table.

The main trends in the results are as follows. MinDistance and MinError generally outperforms Random when it comes to external variance ϕ^2 and internal variance φ^2 . Recall from an earlier section that external variance should be as small as possible, while internal variance should be as large as possible. This means that the MinDistance and MinError algorithms are constructing good hierarchies in terms of these measures. MinDistance and MinError perform quite similarly, except for V_3 and V_4 . Here, MinError is better for external variance, however for internal variance MinDistance is unexpectedly better in the V_4 case.

Concerning the number of levels in the abstraction hierarchies, ℓ , we see that for V_3 and V_4 , MinError gives lower abstraction hierarchies, that is abstraction hierarchies with smaller ℓ -values. This is as expected. Random gives abstraction hierarchies with the smallest ℓ -values, and again this is as expected.

Algorithm	Refined BN			Abstracted BN			
	ℓ	$T(\mathbf{y}_{\text{MPE}})$	$\text{Pr}(\mathbf{y}_{\text{MPE}})$	ℓ	$T(\mathbf{x}_{\text{MPE}})$	$\text{Pr}(\mathbf{x}_{\text{MPE}})$	$\text{Pr}(\hat{\mathbf{y}}_{\text{MPE}})$
Random	2	6.3	1.3×10^{-6}	1	0.031	1.7×10^{-4}	1.5×10^{-8}
Random	3	69	3.7×10^{-7}	2	6.3	1.3×10^{-6}	6.2×10^{-15}
MinDistance	2	0.12	1.1×10^{-5}	1	0.016	2.0×10^{-3}	4.2×10^{-5}
MinDistance	3	3.0	1.7×10^{-6}	2	0.12	1.1×10^{-5}	1.3×10^{-12}
MinError	2	0.29	2.1×10^{-7}	1	0.047	1.3×10^{-6}	1.7×10^{-8}
MinError	3	4.4	8.6×10^{-7}	2	0.29	2.1×10^{-7}	2.9×10^{-13}

Algorithm	Refined BN	Abstracted BN	Ratios		
	ℓ	ℓ	$\frac{\text{Pr}(\hat{\mathbf{y}}_{\text{MPE}})}{\text{Pr}(\mathbf{x}_{\text{MPE}})}$	$\frac{T(\mathbf{y}_{\text{MPE}})}{T(\mathbf{x}_{\text{MPE}})}$	$\frac{\text{Pr}(\hat{\mathbf{y}}_{\text{MPE}})}{\text{Pr}(\mathbf{x}_{\text{MPE}})} \times \frac{T(\mathbf{y}_{\text{MPE}})}{T(\mathbf{x}_{\text{MPE}})}$
Random	2	1	9.1×10^{-5}	203.2	1.8×10^{-2}
Random	3	2	4.8×10^{-9}	11.0	5.3×10^{-8}
MinDistance	2	1	2.0×10^{-2}	7.5	1.5×10^{-1}
MinDistance	3	2	1.2×10^{-7}	25.0	3.0×10^{-6}
MinError	2	1	1.3×10^{-2}	6.2	8.1×10^{-2}
MinError	3	2	1.3×10^{-6}	15.2	2.0×10^{-5}

Table 7.5: Results from MPE experiments using the Barley BN using different abstraction algorithms Random, MinDistance, and MinError. Each row in the tables above relates a Refined BN and an Abstracted BN. In the top table, the inference times and explanation probabilities are presented. In the bottom table, we focus on the trade-off between quality of abstraction and speed of inference.

7.6.2 MPE Experiments

There are various ways of doing measurements on the quality of abstraction. For small BNs, one can perform brute-force enumeration. For larger BNs, where brute-force enumeration is infeasible, one can perform sampling and check to what extent the quality criteria are met. That is, sample a set of abstract explanations $\mathbf{x}_1, \dots, \mathbf{x}_s$, and compute the refined explanations for each \mathbf{x}_i , thus creating a refined set of explanations $\check{\mathbf{x}}_1, \dots, \check{\mathbf{x}}_r$. Now, by considering the quality criteria over these two sets of explanations, we can say something empirically about how our different abstraction algorithms perform. Finally, one can compute an MPE in a refined BN, abstract the MPE, and compare the resulting explanation to an MPE in the abstracted BN. This last approach is followed here.

Results for the Barley BN are presented in Table 7.5. Each row focuses on one algorithm and the relationship between two BNs, where a refined BN is presented to the left, an abstract BN is

presented to the right in the top table. In the abstract BN, \mathbf{x}_{MPE} is a MPE, while \mathbf{y}_{MPE} is an MPE in the refined BN. $\hat{\mathbf{y}}_{\text{MPE}}$ is an MPE estimate, and is constructed by starting with \mathbf{y}_{MPE} , and abstracting each state using the appropriate abstraction hierarchies such that a valid explanation for the abstracted BN results. The Level columns ℓ give the levels to which the BNs are abstracted. The level $\ell = 0$ is when there is one state per node, the root node in the abstraction hierarchy. Therefore, $\ell \geq 1$ means that the BN has two or more states.

In the bottom part of Table 7.5, we focus on maximizing MPE abstraction accuracy ρ_A and speed-up due to abstraction ρ_T . Here, we have the definitions

$$\rho_A = \frac{\Pr(\hat{\mathbf{y}}_{\text{MPE}})}{\Pr(\mathbf{x}_{\text{MPE}})} \leq 1$$

and

$$\rho_T = \frac{T(\mathbf{y}_{\text{MPE}})}{T(\mathbf{x}_{\text{MPE}})} \geq 1.$$

The accuracy ratio ρ_A is a measure of how much better the true MPE $\Pr(\mathbf{x}_{\text{MPE}})$ is compared to the estimated MPE $\Pr(\hat{\mathbf{y}}_{\text{MPE}})$, while the speed-up ratio ρ_T gives how much the abstraction speeds up inference. Clearly, we want to maximize both of ρ_A and ρ_T . In order to have an overall measure, we introduce $\rho_O = \rho_A \times \rho_T$. In the following, we use the Hugin system [Dawid, 1992] [Lauritzen and Spiegelhalter, 1988] to compute MPEs, and the times in the Time columns of Table 7.5 are for computing an MPE using Hugin.

Our interpretation of Table 7.5 is as follows. Note that the rows cannot be compared directly between the algorithms, since in general one cannot assume that they construct abstraction hierarchies with the same depth. However, the ρ_O measure compensates for this, and can be used for comparison between algorithms. The main result column are therefore for ρ_A and ρ_O . Considering the ρ_A column, we first compare the results for MinDistance and MinError versus the results for Random. Here, we see that MinDistance and MinError is much better. Second, we notice that there is no major difference between MinDistance and MinError. This is surprising, and we currently have no good explanation for this, since we found that MinError generally was better in terms of internal and external variance in Table 7.4. A third point is that we see that Random creates

abstraction hierarchies with greater depth, as pointed out earlier, and this is reflected in the Time columns. The times are much greater than those for MinDistance and MinError, so the BNs are much larger here.

Considering the ρ_O measure, we conclude that the quality drops quickly as the levels increase. Second, we see that under ρ_O , MinDistance is still the best abstraction approach at the highest abstraction level, as it was under ρ_A .

7.7 Conclusion and Future Work

By synthesizing research in abstraction and refinement and Bayesian networks, inspired by genetic algorithms, we have introduced novel ways of characterizing, constructing, and utilizing abstraction hierarchies. Quality measures for BN abstraction hierarchies were introduced and empirically investigated with respect to several different approaches to constructing abstraction hierarchies. The MinDistance and MinError abstraction hierarchy construction algorithms were shown to give superior performance over Random abstraction hierarchy generation.

Current and future research related to this includes the following. First, abstraction and refinement can be introduced into search algorithms used for BN inference, such as genetic algorithms, local search and simulated annealing. Second, there is a similarity between abstraction and refinement on the one hand and temperature in simulated annealing on the other, which can be exploited.

Chapter 8

Conclusion and Future Work

Bayesian networks are seeing increased use in artificial intelligence and other related sciences and disciplines. However, Bayesian networks are also inherently computationally hard to perform inference in for most interesting inference tasks. This is the case, for instance, for the problem investigated in this dissertation, namely computing a most probable explanation (MPE). The research reported in this dissertation has been concerned with overcoming these difficulties, and has made progress in the areas of niching genetic algorithms, stochastic local search, and abstraction. Second, we have developed a paradigm for systematic empirical evaluation of algorithms for MPE computation.

In the following, we provide conclusions and outline future work. Conclusions are presented in Section 8.1, while future work is presented in Section 8.2.

8.1 Conclusion

The results of dissertation fall in two categories: First, the algorithmic approaches; Second, the challenging problems. Since the approaches we have developed to construct challenging problems should have interest independently of our stochastic search algorithms, for instance as hard problems for machine learning, we give separate conclusions for these two categories in the following.

The probabilistic crowding genetic algorithm, which gives niching, is presented in Section 8.1.1. Deceptive Bayesian networks are discussed in Section 8.1.2. The stochastic greedy search (SGS) algorithm is presented in Section 8.1.3. An approach to construct challenging problems based on CNF formulas, with empirical results for SGS, is presented in Section 8.1.4. Finally, Section 8.1.5

presents results related to abstraction.

8.1.1 Genetic Algorithm Niching: Probabilistic Crowding

Our research on niching was motivated by an interest in using genetic algorithms for Bayesian network learning and inference, and in particular for computing the most probable explanation in a Bayesian network. In other words, we focused on genetic algorithm fitness functions represented as Bayesian networks.

Bayesian networks are typically multi-modal, something which make them challenging for local search and classical genetic algorithms. Motivated by this, we have investigated niching genetic algorithms and in particular developed a new niching algorithm, probabilistic crowding. The two main objectives of niching algorithms, generally, are to converge to multiple, highly fit, and significantly different solutions, and to slow down convergence in cases where only one solution is required. Different algorithms have been developed to fulfil these objectives, among them deterministic crowding. Probabilistic crowding is closely related to deterministic crowding, and has many of the same advantages: it is simple, fast, and requires no parameters beyond those of classical GAs. Probabilistic crowding is a tournament selection algorithm with similarity-based replacement, and it uses a probabilistic rather than a deterministic acceptance function. We have shown that using a probabilistic acceptance function gives stable, predictable convergence close to the niching rule, a gold standard for niching algorithms, and we showed this both analytically and experimentally.

Our research also identifies probabilistic crowding as a member of a family of algorithms, which we call integrated tournament algorithms. Integrated tournament algorithms also include deterministic crowding, restricted tournament selection, parallel recombinative simulated annealing, the Metropolis algorithm, and simulated annealing.

8.1.2 Problem Hardness: Deceptive Bayesian Networks

Using GAs for computing the most probable explanation raises the question of what the relationship is between a BN and the joint probability distribution defined by the BN, which is the GA's fitness function. The approach we took was to focus on characteristics of a Bayesian network, given a certain joint probability distribution, which was derived from a function of unitation. There

has been some research within the BN community on typical properties of the joint distribution defined by a BN, but little research on systematically varying the hardness of constructed networks, which is what we focused on in this research. To alleviate this, we mapped certain functions of unitation into Bayesian network instances. Deceptive and other functions of unitation have been considered in order to understand which fitness functions are hard and which are easy for genetic algorithms to optimize. There are several advantages to such artificial functions, notably their ease of specification and analysis. The main result of this research was that it showed that Bayesian networks can be deceptive.

8.1.3 Local Search: Stochastic Greedy Search

Stochastic local search has proven to be powerful algorithms for finding satisfying assignments of CNF formulas. The stochastic local greedy search algorithm (SGS) developed as part of this research is a generalization of the GSAT algorithm developed in the context of satisfiability. Two variations of SGS's utility function have been developed and experimented with. One version, GMPE, is a generalization of GSAT's utility measure to the probabilistic case, while the other measure Blanket is a probabilistic version of GSAT's utility measure. In experiments, we have found that both measures of gain are quite effective on hard instances, and in particular they perform several orders of magnitude better than the state-of-the-art exact inferencer Hugin on these instances.

Experiments with SGS on some application BNs have also been done. We have found that the algorithm is quite effective on these networks too, and performs comparably to Hugin. In the application BNs, an important component in the success of the algorithm are the stochastic initialization algorithms, and in particular the dynamic programming and forward simulation algorithms. The forward simulation algorithm is known from earlier research, while the forward and backward dynamic programming algorithms have been developed as part of this dissertation. The initialization algorithms suggest good starting points to the stochastic local search based on a complete solution of a randomly simplified network. In these simplified networks, the initialization algorithms perform in polynomial time.

8.1.4 Problem Hardness: Satisfiability Bayesian Networks

An experimental approach is needed to investigate the performance of inference algorithms for the MPE task. Unfortunately, application BNs have some limitations when used for this purpose. There are typically very few BNs per application, and BNs vary along too many dimensions between applications, making systematic experimentation difficult. Therefore, it is better for experimental purposes to construct random BNs according to some distribution in order to get more a more controlled setting and improved statistics. Careful selection of the distribution from which randomly constructed networks are generated is required, such that they are neither too easy nor too hard.

We have developed a paradigm for systematically generating increasingly hard instances for computing the MPE, as well as a carefully studied some of the factors that contribute to the hardness of inference. We have studied some structural and distributional parameters of Bayesian networks and showed how varying them (with constant network size) can change the hardness of a network as far as MPE computation is concerned. The main parameters studied are: the ratio of the number of root nodes to the number of non-root nodes in a BN, the irregularity of the underlying graph, and the distributions of conditional distribution tables. The hardness of the networks was investigated experimentally using two classes of networks. One of the classes extends research on generating hard instances for satisfiability problems [Mitchell et al., 1992]. We have exploited the relationship between computing an MPE and finding a satisfying assignment of a corresponding CNF formula to construct hard instances for the MPE problem. Two classes of BNs have been investigated: BNs directly constructed from SAT formulas, and a class of BNs introduced by Kask and Dechter [Kask and Dechter, 1999]. For the latter BN class we have showed the relationship to SAT-based BNs. For both classes, generating random networks can result in very easy instances. On the other hand, by carefully selecting parameters along certain dimensions, even while fixing the size of the network, one can construct BNs that existing algorithms cannot handle. Two algorithms have been investigated: First, Hugin, which is one of the best probabilistic inference algorithms available, and which uses clustering and propagation in join trees. Second, the stochastic search algorithm SGS, which we have presented in Section 8.1.3. The results of this research provide directions for how to construct hard and easy instances for the MPE problem, and also shows some interesting aspects of the Hugin and SGS algorithms, in particular how their

performance varies according to certain features of the problem instances being constructed.

8.1.5 Problem Approximation: Abstraction in Bayesian Networks

The size of node state spaces in BNs can have an impact of several orders of magnitude on the speed of inference. In certain circumstances, one might be interested in trading accuracy for speed, and perform inference over an approximated BN. Constructing abstracted BNs based on the use of abstraction hierarchies is one promising approach to BN approximation.

State space abstraction for BNs has been investigated earlier, however this is the first time abstraction has been used in the context of computing an MPE in a Bayesian network. In this research, we have developed criteria for measuring the quality of abstraction hierarchies constructed by different abstraction algorithms. We have regarded error introduced by abstraction as noise, and used variance as a measure of the quality of abstraction hierarchies (and by implication, abstraction algorithms). We have also developed abstraction algorithms, which are techniques for constructing abstraction hierarchies for state spaces of Bayesian networks nodes. Using the abstraction algorithms, we have performed abstraction experiments, both on a node basis and on an MPE basis. At the node level, the abstraction algorithms we have developed are empirically compared using the variance criteria. At the network level, we have evaluated the abstraction hierarchies with respect to their accuracy for computing the MPE of the BN. In both cases, we find that our abstraction algorithms perform significantly better than constructing abstraction hierarchies at random.

8.2 Future Work

Motivated by the MPE problem, we have developed algorithms as well as an approach to empirical investigations. Future work includes extending this work to other inference problems (Section 8.2.1); hybridization and integration of inference methods (Section 8.2.2); and additional research on approximation (Section 8.2.3).

8.2.1 Marginalization

Computing a most probable explanation (computing joint distribution over all nodes in a BN) and belief updating (marginal distribution over one node in a BN) are two extremes on a scale. In this dissertation, we have focused on computing a most probable explanation, and a natural next step is marginalizing over a fixed or variable number of nodes while computing a most probable explanation — or the k most probable explanations — over the remaining nodes. Probably the most obvious way to do marginalization is by simulation, where certain nodes in the BN are designated as fixed, others as variable. (This comes in addition to the clamping of evidence nodes.) Now, we optimize over the fixed nodes as before, but as part of fitness evaluation, for instance, we may simulate over the variable nodes. This gives a setting that is similar to, but not exactly the same as, previous research on genetic algorithms and noisy fitness functions. We have so far assumed that we have designated beforehand which nodes are fixed and which nodes are variable. Automatically detecting which nodes to marginalize out is a more involved extension which would be very interesting to solve.

8.2.2 Hybrid Reasoning and Learning

Several algorithms for computing an MPE have been investigated in this research. Given that as a base-line, there are two natural extensions: First, to combine the algorithms in ways not considered here. The three algorithmic approaches, stochastic local search, genetic algorithms, and abstraction, were investigated in conjunction because they are complimentary and also easy and worthwhile to integrate. Limited integration has been done in this dissertation, however, it seems to be a fruitful area for future work. Second, to learn when to use which inference algorithm. The second area of research focuses on the machine learning aspects of hybridization. One of the lessons from the present research is that different inference algorithms are applicable in different circumstances, and it is not obvious when to use which algorithm. So one wants to use a mixture of algorithms, and in order to do so one could learn a meta-level Bayesian network that estimates the different variables which influence the choice of algorithms. A Bayesian approach at the meta-level would also allow the use of priors for when to use which algorithm, and this seems very natural. For this type of learning for reasoning, there are also the issues related to on-line versus off-line learning.

8.2.3 Bayesian Network Approximation

Future work regarding approximation includes research in the following directions. First, there is the question of whether this research can be generalized to handle aggregation and decomposition, meaning varying the number of nodes in a BN, as well as abstraction and refinement. Note that a fully abstracted BN node has just one state, and one might as well drop it from the network. This gives a basis for aggregation and decomposition, the question is how to exploit this observation to improve speed of inference and for machine learning purposes. Second, there is an analogue between approximation (for example abstraction and refinement) on the one hand, and temperature as used in simulated annealing on the other. Simulated annealing was developed by adding the notion of temperature to the Metropolis algorithm, thus making it more suitable for optimization [Kirkpatrick et al., 1983]. Although this notion of temperature have proven very useful, the originators of simulated annealing remark that it is often hard to map temperature into concepts of the system in question [Kirkpatrick et al., 1983]: ‘[T]he concept of the temperature of a physical system has no obvious equivalent in the system being optimized.’ We suggest there is an equivalent to temperature in artificial intelligence systems, namely the notions of abstraction and refinement: Abstraction increases temperature, refinement lowers temperature. And high levels of abstraction are equivalent to high temperatures, low levels of abstraction are equivalent to low temperatures. The notion of annealing schedule in simulated annealing can be related to an abstraction and refinement schedule when abstraction hierarchies are used. Furthermore, in many areas of artificial intelligence and optimization it is very natural to introduce an abstraction hierarchy, or one might exist already. This contrasts with the fact that what the temperature is in many system being modeled has proven elusive. There is a difference between temperature and abstraction and refinement: The former is a macroscopic control parameter similar to cost (or energy, fitness), while the latter are microscopic control operators. This difference turns out to be fortunate, since it makes the two areas complementary. What temperature brings to the table is that from it, dynamic annealing schedules can be constructed. Given the equivalence between temperature and abstraction, this gives an approach to construct dynamic abstraction and refinement schedules.

Appendix A

Abstraction and Refinement Details

This appendix rederives and provides some more details related to Fung and Chang’s abstraction and refinement operators for Bayesian networks [Chang and Fung, 1991]. Their operators are important, but the original paper [Chang and Fung, 1991], being a conference paper, is lacking some details which we provide here. This material provides background for Chapter 7. Unlike in that chapter, we use the original notation below, for ease of cross-reference to the original paper.

A.1 Abstraction and Refinement

Fung and Chang introduced the two operations of refine and coarsen for discrete Bayesian networks (BNs) [Chang and Fung, 1991]. Coarsen eliminates states for a node (it is an abstraction operation), while refine introduces new states for a node (it is a refinement operation). Both operations take as input a target node and a desired refinement or coarsening, and they output a revised conditional probability distribution for the target node and for the target node’s children, and both operations are based on constraints on the Markov blanket of the node. Both external and internal refinement and external operations are defined. External operations change the BN topology by using Shachter’s operations [Shachter, 1988]. Internal operations, on the other hand, maintain the BN topology. In the following we focus on the internal operations.

These are the inputs to abstraction and refinement:

- node x whose state space Ω_x is to be refined or abstracted.
- new state space Ω'_x .
- a relation between Ω_x and Ω'_x , describing which states $\omega \in \Omega_x$ go with which states $\omega'_x \in \Omega'_x$.

- the Markov blanket of x

The outputs are the following:

- new conditional distribution for x , $\Pr'(x | P_x)$, where P_x are parents of x
- new conditional distribution for successors of x , $\Pr'(s_x | x, P_{s_x})$, where s_x is a successor of x , P_{s_x} is parents of successors of x (excluding x). Note the difference between S_x , which is all successors, and s_x , which is one successor.

The relation between Ω_x and Ω'_x , mentioned above, is specified by:

- R : maps a single value in Ω_x into multiple values in Ω'_x
- C : maps a single value in Ω'_x into multiple values in Ω_x

The topology of the simplest BN that exhibits all features mentioned above is shown in Figure 7.3.

A.2 Refinement

The central idea in the following is to keep the effect of abstraction and refinement localized, in particular by keeping the joint distribution of the Markov blanket intact. First we consider constraints that apply to the new conditional probability distribution when $\omega_x \in \Omega_x$ is refined to $\omega'_x \in \Omega'_x$.

For refinement, a constraint on the new probability distribution that obeys that the principle of leaving the Markov blanket intact is the following. This constraint applies to a node x and its parent P_x :

$$\Pr(\omega_x | P_x) = \sum_{\omega'_x \in R(\omega_x)} \Pr(\omega'_x | P_x). \quad (\text{A.1})$$

Recall that R is a refinement function that maps a single value $\omega_x \in \Omega_x$ into multiple values $\omega'_x \in \Omega'_x$.

For refinement, the constraint on the Markov blanket is then:

$$\begin{aligned} \Pr(S_x | P_x, P_{S_x}) &= \sum_x \Pr(s_x | x, P_{s_x}) \Pr(x | P_x) \\ &= \sum_{x'} \Pr(s_{x'} | x', P_{s_x}) \Pr(x' | P_{x'}) \end{aligned} \quad (\text{A.2})$$

In particular, for the state ω_x to be refined, the equation

$$\Pr(s_x | \omega_x, P_{s_x}) \Pr(\omega_x | P_x) = \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) \Pr(\omega'_x | P_x) \quad (\text{A.3})$$

needs to be satisfied. here, the old and abstract state is on the left hand side, the new and refined state is on the right hand side.

A.2.1 Fixed Children Method

A solution satisfying the constraint in Equation A.3 regardless of $\Pr(\omega'_x | P_x)$ is

$$\Pr(s_x | \omega'_x, P_{s_x}) = \Pr(s_x | \omega_x, P_{s_x}). \quad (\text{A.4})$$

This fixes the conditional distribution for the children, but allows full freedom for the parent conditional distributions.

We now turn to an example of Equation A.3 in use. The example uses the Bayesian network shown in Figure 7.4. here, M stand for military unit type, with $\Omega_M = \{A, B\}$. V stands for a vehicle in a particular place at a particular time, with $\Omega_V = \{Y, N\}$ (yes or no) or $\Omega_V = \{A, U, N\}$ (tank, truck, or no). T stands for terrain conditions, with $\Omega_T = \{G, B\}$ (good or bad). F is a feature.

Conditional probability tables (CPTs) related to this example are shown in Table A.1 and Table A.2. In this network, V 's state space is refined from $\Omega_V = \{Y, N\}$ to $\Omega_V = \{A, U, N\}$, so $R(\omega_x) = R(Y) = \{A, U\}$. We focus here on the case where the other nodes have values $F = A$, $T = G$, and $M = A$. First, consider the left hand side of Equation A.3, containing the state $\omega_x = Y$ which is refined:

Pr($F V, T$)				
V:	Y		N	
T:	G	B	G	B
A	0.45	<i>0.30</i>	0.10	0.20
B	0.45	<i>0.30</i>	0.10	0.20
O	0.10	<i>0.40</i>	0.80	0.60

Pr($F V, T$)						
V:	A		U		N	
T:	G	B	G	B	G	B
A	0.45	<i>0.30</i>	0.45	<i>0.30</i>	0.10	0.20
B	0.45	<i>0.30</i>	0.45	<i>0.30</i>	0.10	0.20
O	0.10	<i>0.40</i>	0.10	<i>0.40</i>	0.80	0.60

Table A.1: To the left F 's CPT is shown before refinement, to the right it is shown after refinement. Notice how these child distributions are kept the same.

$$\begin{aligned}
\Pr(s_x | \omega_x, P_{s_x}) \Pr(\omega_x | P_x) &= \Pr(F = A | T = G, V = Y) \Pr(V = Y | M = A) \\
&= 0.45 \times 0.8
\end{aligned}$$

Second, consider the states $R(\omega_x) = R(Y) = \{A, U\}$ in the right hand side of Equation A.3:

$$\begin{aligned}
\sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) \Pr(\omega'_x | P_x) &= \Pr(F = A | T = G, V = A) \Pr(V = A | M = A) \\
&\quad + \Pr(F = A | T = G, V = U) \Pr(V = U | M = A)
\end{aligned}$$

Now since $\Pr(F = A | T = G, V = A) = 0.45 = \Pr(F = A | T = G, V = U)$ (see Table A.1) while $\Pr(V = A | M = A) = 0.5$ and $\Pr(V = U | M = A) = 0.3$ (see Table A.2), we get:

$$\begin{aligned}
\sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) \Pr(\omega'_x | P_x) &= 0.45 \times 0.5 + 0.45 \times 0.3 \\
&= 0.45 \times 0.8
\end{aligned}$$

Pr(V M)		
M:	A	B
Y	0.8	0.4
N	0.2	0.6

Pr(V M)		
M:	A	B
A	0.3	0.1
U	0.5	0.3
N	0.2	0.6

Table A.2: To the left V 's CPT is shown before refinement, to the right it is shown after refinement. Notice how these distributions are changed.

Pr(V M)		
M:	A	B
Y	0.8	0.4
N	0.2	0.6

Pr(V M)		
M:	A	B
A	0.16	0.08
U	0.64	0.32
N	0.20	0.60

Table A.3: To the left, V 's CPT is shown before refinement, to the right it is shown after refinement.

A.2.2 Fixed Parents Method

Suppose that the proportions $\Pr(\omega'_x | P_x)$ assigned in Equation A.1 are the same for all predecessor values, namely if

$$\frac{\Pr(\omega'_x | P_x)}{\sum_{\omega'_x \in R(\omega_x)} \Pr(\omega'_x | P_x)} = K(\omega'_x) \tag{A.5}$$

where $K(\omega'_x)$ is a function depending only on ω'_x , then Equation A.3 can be reduced to a single constraint

$$\Pr(s_x | \omega_x, P_{s_x}) = \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) K(\omega'_x). \tag{A.6}$$

(See proof of this in the next section.) In this case, $\Pr(s_x | \omega'_x, P_{s_x})$ does not need to be the same as $\Pr(s_x | \omega_x, P_{s_x})$. This gives the opposite situation of the one described earlier: here the parent distribution is fixed, while the child distributions are changed.

Consider Table A.4. here, the column labeled ‘Y’ is refined into the columns labeled ‘A’ and ‘U’. Consider the parent state $M = A$:

Pr(F V, T)				
V:	Y		N	
T:	G	B	G	B
A	0.45	0.30	0.10	0.20
B	0.45	0.30	0.10	0.20
O	0.10	0.40	0.80	0.60

Pr(F V, T)						
V:	A		U		N	
T:	G	B	G	B	G	B
A	0.80	0.50	0.3625	0.25	0.10	0.20
B	0.10	0.10	0.5375	0.35	0.10	0.20
O	0.10	0.40	0.1000	0.40	0.80	0.60

Table A.4: To the left, F 's CPT is shown before refinement, to the right it is shown after refinement.

$$\begin{aligned} \frac{\Pr(\omega'_x | P_x)}{\sum_{\omega'_x \in R(\omega_x)} \Pr(\omega'_x | P_x)} &= \frac{\Pr(V = A | M = A)}{\Pr(V = A | M = A) + \Pr(V = U | M = A)} \\ &= \frac{0.16}{0.16 + 0.64} = 0.2 = K(A). \end{aligned}$$

Likewise for the parent state $M = B$:

$$\begin{aligned} \frac{\Pr(\omega'_x | P_x)}{\sum_{\omega'_x \in R(\omega_x)} \Pr(\omega'_x | P_x)} &= \frac{\Pr(V = A | M = B)}{\Pr(V = A | M = B) + \Pr(V = U | M = B)} \\ &= \frac{0.08}{0.08 + 0.32} = 0.2 = K(A). \end{aligned}$$

Along similar lines, we have $K(U) = 0.8$.

Now consider the left hand side of Equation A.6. The CPT value is given as (see Table A.4):

$$\Pr(s_x | \omega_x, P_{s_x}) = \Pr(F = A | T = G, V = Y) = 0.45$$

Next consider the right hand side of Equation A.6:

$$\begin{aligned} \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) K(\omega'_x) &= \Pr(F = A | T = G, V = A) \times K(A) \\ &\quad + \Pr(F = A | T = G, V = U) \times K(U) \end{aligned}$$

Now with $\Pr(F = A | T = G, V = A) = 0.8$, $K(A) = 0.2$, $\Pr(F = A | T = G, V = U) = 0.3625$, and $K(U) = 0.8$ we get

$$\sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) K(\omega'_x) = 0.8 \times 0.2 + 0.3625 \times 0.8 = 0.45.$$

Where do the numbers in Table A.4 come from? This is what we will focus on next. Consider

$$\Pr(F_i | V_y, T_j) = \frac{\Pr(F_i | V_a, T_j) \Pr(V_a | M_k) + \Pr(F_i | V_u, T_j) \Pr(V_u | M_k)}{\Pr(V_y | M_k)}, \quad (\text{A.7})$$

where, F_i is the i th value of the node F .

Now pick $\Pr(F_a | V_a, T_g)$ (or $\Pr(F_a | V_u, T_g)$); this determines $\Pr(F_a | V_u, T_g)$ (or $\Pr(F_a | V_a, T_g)$). Consider the case $F_i = F_a$ and $T_j = T_g$. The following is seen from Table A.4:

$$\Pr(F_a | V_y, T_g) = 0.45.$$

Using Equation A.7,

$$\begin{aligned} \Pr(F_a | V_y, T_g) &= \frac{\Pr(F_a | V_a, T_g) \Pr(V_a | M_k)}{\Pr(V_y | M_k)} + \frac{\Pr(F_a | V_u, T_g) \Pr(V_u | M_k)}{\Pr(V_y | M_k)} \\ &= \Pr(F_a | V_a, T_g) \times 0.2 + \Pr(F_a | V_u, T_g) \times 0.8. \end{aligned}$$

Taking the above together, we get:

$$\Pr(F_a | V_a, T_g) \times 0.2 + \Pr(F_a | V_u, T_g) \times 0.8 = 0.45.$$

Picking $\Pr(F_a | V_a, T_g) = 0.8$, this gives:

$$\Pr(F_a | V_u, T_g) = \frac{0.45 - \Pr(F_a | V_a, T_g) \times 0.2}{0.8} = \frac{0.45 - 0.8 \times 0.2}{0.8} = 0.3625,$$

as shown to the right in Table A.4.

Pr($F V, T$)						
$V:$	A		U		N	
$T:$	G	B	G	B	G	B
A	0.4	<i>0.1</i>	0.6	<i>0.4</i>	0.1	0.2
B	0.5	<i>0.5</i>	0.3	<i>0.2</i>	0.1	0.2
O	0.1	<i>0.4</i>	0.1	<i>0.4</i>	0.8	0.6

Pr($F V, T$)				
$V:$	Y		N	
$T:$	G	B	G	B
A	0.5375	<i>0.30625</i>	0.1	0.2
B	0.3625	<i>0.29375</i>	0.1	0.2
O	0.1000	<i>0.40000</i>	0.8	0.6

Table A.5: To the left, F 's CPT is shown before abstraction to the right it is shown after abstraction.

Pr($V M$)		
$M:$	A	B
A	0.3	<i>0.1</i>
U	0.5	<i>0.3</i>
N	0.2	<i>0.6</i>

Pr($V M$)		
$M:$	A	B
Y	0.8	<i>0.4</i>
N	0.2	<i>0.6</i>

Table A.6: To the left, V 's CPT is shown before abstraction, to the right it is shown after abstraction.

A.3 Abstraction (Coarsening)

Now let us turn to abstraction (what Chang and Fung call coarsening). The two constraints here are:

$$\Pr(\omega'_x | P_x) = \sum_{\omega_x \in C(\omega'_x)} \Pr(\omega_x | P_x)$$

and

$$\Pr(s_x | \omega'_x, P_{s_x}) \Pr(\omega'_x | P_x) = \sum_{\omega_x \in C(\omega'_x)} \Pr(s_x | \omega_x, P_{s_x}) \Pr(\omega_x | P_x).$$

The last equation is equivalent to

$$\Pr(s_x | \omega'_x, P_{s_x}) = \frac{1}{\Pr(\omega'_x | P_x)} \sum_{\omega_x \in C(\omega'_x)} \Pr(s_x | \omega_x, P_{s_x}) \Pr(\omega_x | P_x).$$

One might need to approximate when doing abstraction.

Let $\omega'_x = Y$, $C(\omega'_x) = C(Y) = \{A, U\}$. The coarsening operation is then :

$$\frac{1}{\Pr(\omega'_x | P_x)} \sum_{\omega_x \in C(\omega'_x)} \Pr(s_x | \omega_x, P_{s_x}) \Pr(\omega_x | P_x) =$$

$$\frac{1}{\Pr(V = Y | M = m)} \sum_{\omega_x \in C(V=Y)} \Pr(F = f | \omega_x, T = t) \Pr(\omega_x | M = m),$$

with some assignments $F = f$, $T = t$, and $M = m$. Now let $f = A$, $t = G$:

$$\sum_{\omega_x \in C(V=Y)} \Pr(F = A | \omega_x, T = G) \Pr(\omega_x | M = m).$$

Now let $M = A$; we can go to Table A.5 and Table A.6 to find values:

$$\frac{1}{\Pr(V = Y | M = A)} \sum_{\omega_x \in C(Y)} \Pr(F = A | \omega_x, T = G) \Pr(\omega_x | M = A) =$$

$$\frac{1}{0.8} (0.4 \times 0.3 + 0.6 \times 0.5) = 0.525.$$

More precisely, we used the fact that $\omega_x \in \{A, U\}$, so

$$\sum_{\omega_x \in C(Y)} \Pr(F = A | \omega_x, T = G) \Pr(\omega_x | M = A) = \Pr(F = A | V = A, T = G) \Pr(V = A | M = A)$$

$$+ \Pr(F = A | V = U, T = G) \Pr(V = U | M = A)$$

Similarly, we let $M = B$ and get

$$\frac{1}{\Pr(V = Y | M = B)} \sum_{\omega_x \in C(V=Y)} \Pr(F = A | \omega_x, T = G) \Pr(\omega_x | M = B) =$$

$$\frac{1}{0.4} (0.4 \times 0.1 + 0.6 \times 0.3) = 0.55.$$

Taking the above two results together and computing the average, one gets

$$\Pr(F = A | V = Y, T = G) = \frac{0.525 + 0.55}{2} = 0.5357,$$

which is the value shown in Table A.5.

A.4 Proofs

Equation A.6 is derived as follows. From Equation A.3 we have:

$$\Pr(s_x | \omega_x, P_{s_x}) \Pr(\omega_x | P_x) = \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) \Pr(\omega'_x | P_x)$$

which is equivalent to

$$\Pr(s_x | \omega_x, P_{s_x}) = \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) \frac{\Pr(\omega'_x | P_x)}{\Pr(\omega_x | P_x)}.$$

By the definition of $\Pr(\omega_x | P_x)$ in Equation A.1 we get

$$\Pr(s_x | \omega_x, P_{s_x}) = \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) \frac{\Pr(\omega'_x | P_x)}{\sum_{\omega'_x \in R(\omega_x)} \Pr(\omega'_x | P_x)}.$$

we can now introduce the definition of $K(\omega'_x)$ as shown in Equation A.5, and get the desired result:

$$\Pr(s_x | \omega_x, P_{s_x}) = \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) K(\omega'_x).$$

Next, we show the derivation of Equation A.7, reproduced here:

$$\Pr(F_i | V_y, T_j) = \frac{\Pr(F_i | V_a, T_j) \Pr(V_a | M_k) + \Pr(F_i | V_u, T_j) \Pr(V_u | M_k)}{\Pr(V_y | M_k)}.$$

Remember that F_i is the i th value of the node F . To derive the equation above, start with using Equation A.6

$$\Pr(F_i | V_y, T_j) = \sum_{V'_y \in R(V_y)} \Pr(F_i | V'_y, T_j) K(V'_y),$$

where $K(V'_y)$ is defined according to Equation A.5:

$$K(V'_y) = \frac{\Pr(V'_y | M_k)}{\sum_{V'_y \in R(V_y)} \Pr(V'_y | M_k)} = \frac{\Pr(V'_y | M_k)}{\Pr(V_y | M_k)}.$$

And since $R(V_y) = \{V_a, V_u\}$, we get two terms in the sum

$$\sum_{V'_y \in R(V_y)} \Pr(F_i | V'_y, T_j) \frac{\Pr(V'_y | M_k)}{\Pr(V_y | M_k)} = \frac{\Pr(F_i | V_a, T_j) \Pr(V_a | M_k) + \Pr(F_i | V_u, T_j) \Pr(V_u | M_k)}{\Pr(V_y | M_k)},$$

which is the desired result in Equation A.7.

The following is a derivation of Equation A.4.

$$\Pr(s_x | \omega_x, P_{s_x}) \Pr(\omega_x | P_x) = \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) \Pr(\omega'_x | P_x)$$

Now let $\Pr(s_x | \omega'_x, P_{s_x}) = \Pr(s_x | \omega_x, P_{s_x})$ This gives:

$$\begin{aligned} \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega'_x, P_{s_x}) \Pr(\omega'_x | P_x) &= \sum_{\omega'_x \in R(\omega_x)} \Pr(s_x | \omega_x, P_{s_x}) \Pr(\omega'_x | P_x) \\ &= \Pr(s_x | \omega_x, P_{s_x}) \sum_{\omega'_x \in R(\omega_x)} \Pr(\omega'_x | P_x) \\ &= \Pr(s_x | \omega_x, P_{s_x}) \Pr(\omega_x | P_x), \end{aligned}$$

where the last step follows from Equation A.1.

Bibliography

- [Ackley, 1987] Ackley, D. H. (1987). *An Empirical Study of Bit Vector Function Optimization*, chapter 13, pages 170–204. In [Davis, 1987].
- [Andersen et al., 1989] Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen, F. (1989). HUGIN—a shell for building Bayesian belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, volume 2, pages 1080–1085, Detroit, Michigan. Morgan Kaufmann.
- [Andreassen et al., 1987] Andreassen, S., Woldbye, M., Falck, B., and Andersen, S. (1987). MUNIN – A causal probabilistic network for interpretation of electromyographic findings. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 366–372, Milan, Italy.
- [Angluin and Valiant, 1979] Angluin, D. and Valiant, L. G. (1979). Fast probabilistic algorithms for Hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18(2):155–193.
- [Bäck, 1996] Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York.
- [Basye et al., 1992] Basye, K., Dean, T., Kirman, J., and Lejter, M. (1992). A decision-theoretic approach to planning, perception, and control. *IEEE Expert*, 7(4):58–65.
- [Beinlich et al., 1989] Beinlich, I. A., Suermondt, H. J., Chavez, R. M., and Cooper, G. F. (1989). The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings Second European Conference on Artificial Intelligence in Medicine*, pages 247–256. Springer-Verlag, New York.

- [Buchanan and Shortliffe, 1984] Buchanan, B. and Shortliffe, E. (1984). *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison Wesley, Reading, MA.
- [Castillo et al., 1997] Castillo, E., Gutierrez, J. M., and Hadi, A. S. (1997). *Expert systems and probabilistic network models*. Springer-Verlag, New York.
- [Chang and Fung, 1991] Chang, K. and Fung, R. (1991). Refinement and coarsening of Bayesian networks. In *Uncertainty in Artificial Intelligence 6*, pages 435–445. Elsevier, Amsterdam.
- [Chapman and Saitou, 1994] Chapman, C. D. and Saitou, K. (1994). Genetic algorithms as an approach to configuration and topology design. *Journal of Mechanical Design*, 116:1005–1012.
- [Charniak, 1991] Charniak, E. (1991). Bayesian networks without tears. *AI Magazine*, 12(4):50–63.
- [Cheeseman et al., 1991] Cheeseman, P., Kanefsky, B., and Taylor, W. M. (1991). Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, Sydney, Australia.
- [Cooper, 1990] Cooper, F. G. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405.
- [Culberson, 1992] Culberson, J. (1992). Genetic invariance: A new paradigm for genetic algorithm design. Technical Report 92-02, University of Alberta, Department of Computer Science.
- [Davis, 1987] Davis, L., editor (1987). *Genetic Algorithms and Simulated Annealing*. Pitman, London.
- [Dawid, 1992] Dawid, A. P. (1992). Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36.
- [Deb and Goldberg, 1993] Deb, K. and Goldberg, D. E. (1993). Analyzing deception in trap functions. In [Whitley, 1993].
- [Deb and Goldberg, 1994] Deb, K. and Goldberg, D. E. (1994). Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, 10:385–408.

- [DeJong, 1975] DeJong, K. A. (1975). *An Analysis of the Behavior a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan.
- [Druzdzel, 1994] Druzdzel, M. J. (1994). Some properties of joint probability distributions. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 187–194, Seattle, WA.
- [Duda et al., 1976] Duda, R., Hart, P., and Nilsson, N. (1976). Subjective Bayesian methods for rule-based inference systems. In *AFIPS Conference Proceedings of the 1976 National Computer Conference*, pages 1075–1082, vol 45.
- [Franco and Paull, 1983] Franco, J. and Paull, M. (1983). Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5:77–87.
- [Frey, 1998] Frey, B. J. (1998). *Graphical Models for Machine Learning and Digital Communication*. MIT Press, Cambridge, MA.
- [Fung and Chang, 1990] Fung, R. and Chang, K. (1990). Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *Uncertainty in Artificial Intelligence 5*, pages 209–219. Elsevier, Amsterdam.
- [Fung and Favero, 1994] Fung, R. and Favero, B. D. (1994). Backward simulation in Bayesian networks. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 227–234, Seattle, WA.
- [Gallager, 1962] Gallager, R. G. (1962). Low density parity check codes. *IRE Transactions on Information Theory*, 8:21–28.
- [Geiger et al., 1990] Geiger, D., Verma, T., and Pearl, J. (1990). d-separation: From theorems to algorithms. In *Uncertainty in Artificial Intelligence 5*, pages 139–148. Elsevier, Amsterdam.
- [Gelsema, 1995] Gelsema, E. S. (1995). Abductive reasoning in Bayesian belief networks using a genetic algorithm. *Pattern Recognition Letters*, 16:865–871.
- [Genesereth, 1984] Genesereth, M. (1984). The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436.

- [Goldberg, 1989a] Goldberg, D. (1989a). Genetic algorithms and Walsh functions: II. Deception and its analysis. *Complex Systems*, 3(2):153–171.
- [Goldberg, 1989b] Goldberg, D. (1989b). Genetic algorithms and Walsh functions: I. A gentle introduction. *Complex Systems*, 3(2):129–152.
- [Goldberg, 1990] Goldberg, D. (1990). A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 4(4):445–460.
- [Goldberg et al., 1992] Goldberg, D., Deb, K., and Clark, J. (1992). Genetic algorithms, noise and the sizing of populations. *Complex Systems*, 6(4):333–362.
- [Goldberg, 1987] Goldberg, D. E. (1987). *Simple Genetic Algorithms and the Minimal Deceptive Problem*, pages 74–88. In [Davis, 1987].
- [Goldberg, 1989c] Goldberg, D. E. (1989c). *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA.
- [Goldberg, 1992] Goldberg, D. E. (1992). Construction of high-order deceptive functions using low-order Walsh coefficients. *Annals of Mathematics and Artificial Intelligence*, 5:35–48.
- [Goldberg, 1994] Goldberg, D. E. (1994). Genetic and evolutionary algorithms come of age. *Communications of the ACM*, 37(3):113–119.
- [Goldberg and Richardson, 1987] Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, Hillsdale, NJ. Erlbaum.
- [Grefenstette, 1993] Grefenstette, J. J. (1993). Deception considered harmful. In [Whitley, 1993].
- [Grefenstette and Fitzpatrick, 1985] Grefenstette, J. J. and Fitzpatrick, J. M. (1985). Genetic search with approximate function evaluations. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 112–120, Pittsburg, PA. Erlbaum.
- [Harik, 1995] Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 24–31, San Francisco, CA. Morgan Kaufmann.

- [Henrion, 1988] Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Uncertainty in Artificial Intelligence 2*. Elsevier, Amsterdam.
- [Henrion, 1991] Henrion, M. (1991). Search-based methods to bound diagnostic probabilities in very large belief networks. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence (UAI-91)*, pages 142–150.
- [Henrion et al., 1991] Henrion, M., Breese, J. S., and Horvitz, E. J. (1991). Decision analysis and expert systems. *AI Magazine*, 12(4):64–9.
- [Holland, 1975] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.
- [Horvitz et al., 1988] Horvitz, E. J., Breese, J. S., and Henrion, M. (1988). Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2:247–302.
- [Horvitz et al., 1989] Horvitz, E. J., Suermondt, H. J., and Cooper, G. F. (1989). Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, pages 182–193, Windsor, Ontario. Morgan Kaufmann.
- [Huang and Darwiche, 1996] Huang, C. and Darwiche, A. (1996). Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15:225–263.
- [Jensen, 1996] Jensen, F. V. (1996). *An Introduction to Bayesian Networks*. Springer-Verlag, New York.
- [Jensen et al., 1990a] Jensen, F. V., Lauritzen, S. L., and Olesen, K. G. (1990a). Bayesian updating in causal probabilistic networks by local computations. *SIAM Journal on Computing*, 4:269–282.
- [Jensen et al., 1990b] Jensen, F. V., Olesen, K. G., and Andersen, S. K. (1990b). An algebra of Bayesian belief universes for knowledge-based systems. *Networks*, 20(5):637–659.
- [Kanazawa et al., 1995] Kanazawa, K., Koller, D., and Russell, S. (1995). Stochastic simulation algorithm for dynamic probabilistic networks. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 346–351, Montreal, Canada.

- [Kask and Dechter, 1999] Kask, K. and Dechter, R. (1999). Stochastic local search for Bayesian networks. In *Proceedings Seventh International Workshop on Artificial Intelligence and Statistics*, Fort Lauderdale, FL. Morgan Kaufmann.
- [Kim and Pearl, 1983] Kim, J. H. and Pearl, J. (1983). A computational model for combined causal and diagnostic reasoning in inference systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 190–193, Karlsruhe, Germany. Morgan Kaufmann.
- [Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt Jr., C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.
- [Koller, 1996] Koller, D. (1996). Approximate probabilistic inference in dynamic processes. In *Working Notes of the 1996 AAAI Spring Symposium on Learning Dynamical Systems*.
- [Koza, 1989] Koza, J. R. (1989). Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 768–774, Detroit, MI. Morgan Kaufmann.
- [Kristensen and Rasmussen, 1997a] Kristensen, K. and Rasmussen, I. A. (1997a). A decision support system for mechanical weed control in malting barley. In *Proceedings of First European Conference for Information Technology in Agriculture*, pages 447–452, The Royal Veterinary and Agricultural University, Copenhagen, Denmark.
- [Kristensen and Rasmussen, 1997b] Kristensen, K. and Rasmussen, I. A. (1997b). Using a bayesian network for setting up a decision support for effect of fungal diseases in malting barley. In *XXVII International Biometrical Colloquium, Abstracts*, pages 24–25, The Polish Biometric Society.
- [Laarhoven and Aarts, 1987] Laarhoven, P. J. M. v. and Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications*. Mathematics and Its Applications. D. Reidel, Dordrecht, Holland.
- [Larranaga et al., 1996a] Larranaga, P., Kuijpers, C. M. H., Murga, R. H., and Yurramendi, Y. (1996a). Learning Bayesian network structure by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(4):487–493.

- [Larranaga et al., 1996b] Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Yurramendi, Y., Grana, M., Lozano, J. A., Albizuri, X., D’Anjou, A., and Torrealdea, F. J. (1996b). *Computational Learning and Probabilistic Reasoning*, chapter Genetic Algorithms Applied to Bayesian Networks, pages 211–234. Wiley, Chichester, UK.
- [Larranaga et al., 1996c] Larranaga, P., Murga, R. H., Poza, M., and Kuijpers, C. M. H. (1996c). *Learning from Data: AI and Statistics V*, chapter Structure Learning of Bayesian Networks by Hybrid Genetic Algorithms, pages 165–174. Springer-Verlag, New York.
- [Lauritzen and Spiegelhalter, 1988] Lauritzen, S. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society series B*, 50(2):157–224.
- [Li and D’Ambrosio, 1994] Li, Z. and D’Ambrosio, B. (1994). Efficient inference in Bayes nets as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11(1):55–81.
- [Lin et al., 1990] Lin, R., Galper, A., and Shachter, R. (1990). Abductive inference using probabilistic networks: Randomized search techniques. Technical Report KSL-90-73, Knowledge Systems Laboratory, Stanford.
- [Liu and Wellman, 1996] Liu, C. and Wellman, M. P. (1996). On state-space abstraction for anytime evaluation of Bayesian networks. *SIGART Bulletin*, pages 50–57.
- [Luby et al., 1998] Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., and Spielman, D. A. (1998). Improved low-density parity-check codes using irregular graphs and belief propagation. In *ISIT 1998*, Cambridge, MA.
- [Mahfoud, 1995] Mahfoud, S. W. (1995). *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA. IlliGAL Report 95001.
- [Mahfoud and Goldberg, 1995] Mahfoud, S. W. and Goldberg, D. E. (1995). Parallel recombinative simulated annealing: A genetic algorithm. *Parallel Computing*, 21:1–28.

- [Mengshoel, 1997] Mengshoel, O. J. (1997). Belief network inference in dynamic environments. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, page 813, Madison, WI.
- [Mengshoel and Wilkins, 1998a] Mengshoel, O. J. and Wilkins, D. C. (1998a). Abstraction for belief revision: Using a genetic algorithm to compute the most probable explanation. In *Satisficing Models: Papers from the 1998 AAAI Spring Symposium*, pages 46–53, Menlo Park, CA. AAAI Press. Technical Report SS-98-05.
- [Mengshoel and Wilkins, 1998b] Mengshoel, O. J. and Wilkins, D. C. (1998b). Genetic algorithms for belief network inference: The role of scaling and niching. In *Proceedings Seventh Annual Conference on Evolutionary Programming*, pages 547–556, San Diego, CA.
- [Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092.
- [Michalewicz, 1992] Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs*. Artificial Intelligence. Springer-Verlag, New York.
- [Miller, 1997] Miller, B. (1997). *Noise, Sampling, and Efficient Genetic Algorithms*. PhD thesis, University of Illinois, Urbana-Champaign. IlliGAL Report 97001.
- [Miller et al., 1993] Miller, J. A., Potter, W. D., Gandham, R. V., and Lapena, C. N. (1993). An evaluation of local improvement operators for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(5):1340–1351.
- [Mitchell et al., 1992] Mitchell, D., Selman, B., and Levesque, H. J. (1992). Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 459–465, San Jose, CA.
- [Mitchell, 1996] Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, MA.

- [Neal, 1993] Neal, R. M. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto.
- [Neapolitan, 1990] Neapolitan, R. E. (1990). *Probabilistic Reasoning in Expert Systems*. John Wiley & Sons, New York, New York.
- [Olesen, 1993] Olesen, K. G. (1993). Causal probabilistic networks with both discrete and continuous variables. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):275–279.
- [Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA.
- [Pearl, 1990] Pearl, J. (1989–1990). Reasoning under uncertainty. In *Annual Review of Computer Science*, volume 4, pages 37–72. Annual Reviews Inc., Palo Alto, CA.
- [Peng and Reggia, 1987a] Peng, Y. and Reggia, J. A. (1987a). A probabilistic causal model for diagnostic problem solving—part I: Integrating symbolic causal inference with numeric probabilistic inference. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(2):146–162.
- [Peng and Reggia, 1987b] Peng, Y. and Reggia, J. A. (1987b). A probabilistic causal model for diagnostic problem solving—part II: Diagnostic strategy. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(3):395–406.
- [Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286.
- [Rojas-Guzman, 1995] Rojas-Guzman, C. (1995). *An Evolutionary Programming Approach to Probabilistic Model-Based Fault Diagnosis of Chemical Processes*. PhD thesis, MIT.
- [Rojas-Guzman and Kramer, 1993] Rojas-Guzman, C. and Kramer, M. A. (1993). GALGO: A Genetic ALGORITHM decision support tool for complex uncertain systems modeled with Bayesian belief networks. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence (UAI-93)*, pages 368–375, Washington, DC.

- [Rojas-Guzman and Kramer, 1996] Rojas-Guzman, C. and Kramer, M. A. (1996). An evolutionary computing approach to probabilistic reasoning on Bayesian networks. *Evolutionary Computation*, 4(1):57–85.
- [Roth, 1996] Roth, D. (1996). On the hardness of approximate reasoning. *Artificial Intelligence*, 82:273–302.
- [Russell and Norvig, 1995] Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ.
- [Santos et al., 1997] Santos, E., J., Shimony, S. E., and Williams, E. (1997). Hybrid algorithms for approximate belief updating in Bayes nets. *International Journal of Approximate Reasoning*, 17:217–237.
- [Santos et al., 1996] Santos, Eugene, J., Shimony, S. E., and Williams, E. (1996). Sample-and-accumulate algorithms for belief updating in Bayes networks. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 477–484, Portland, OR.
- [Scheines et al., 1994] Scheines, R., Spirtes, P., Glymour, C., and Meek, C. (1994). *TETRAD II: Tools for Causal Modeling*. Erlbaum, Hillsdale, NJ.
- [Selman et al., 1994] Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 337–343, Seattle, Washington.
- [Selman et al., 1992] Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, CA. AAAI Press.
- [Shachter, 1988] Shachter, R. (1988). Probabilistic inference and influence diagrams. *Operations Research*, 36:589–604.
- [Shachter and Peot, 1990] Shachter, R. and Peot, M. (1990). Simulation approaches to general

- probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence 5*, pages 221–231, Amsterdam. Elsevier.
- [Shimony, 1994] Shimony, E. (1994). Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68:399–410.
- [Spiegelhalter et al., 1993] Spiegelhalter, D. J., Dawid, A. P., Lauritzen, S. L., and Cowell, R. G. (1993). Bayesian analysis in expert systems (with discussion). *Statistical Science*, 8:219–283.
- [Srinivas, 1994] Srinivas, S. (1994). A probabilistic approach to hierarchical model-based diagnosis. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 538–545, Seattle, WA.
- [Srinivas and Horvitz, 1995] Srinivas, S. and Horvitz, E. (1995). Exploiting system hierarchy to compute repair plans in probabilistic model-based diagnosis. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 523–531, Montreal, Quebec, Canada.
- [Syswerda, 1991] Syswerda, G. (1991). A study of reproduction in generational and steady-state genetic algorithms. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms*, pages 94–101, San Mateo, CA. Morgan Kaufmann.
- [Thierens and Goldberg, 1994] Thierens, D. and Goldberg, D. E. (1994). Elitist recombination: An integrated selection recombination GA. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 152–159, Orlando, FL.
- [Welch, 1996] Welch, R. L. (1996). Real time estimation of Bayesian networks. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 533–544, Portland, Oregon.
- [Welch, 1997] Welch, R. L. (1997). Personal communication.
- [Wellman and Liu, 1994] Wellman, M. P. and Liu, C.-L. (1994). State-space abstraction for any-time evaluation of probabilistic networks. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 567–574, Seattle, WA.

[Whitley, 1993] Whitley, D., editor (1993). *Foundations of Genetic Algorithms II*. Morgan Kaufmann, San Mateo, CA.

Vita

Personal Data

Full name: Ole Jakob Mengshoel

Year and place of birth: 1963, Hamar, Norway

Nationality: Norwegian

Education

Ph.D., Computer Science, University of Illinois, Urbana-Champaign (UIUC), Illinois (Aug 1993 – May 1999).

- Passed the Preliminary Defense in July 1997, tentative Ph.D. dissertation title ‘Efficient Bayesian Network Inference: Genetic Algorithms, Stochastic Local Search, and Abstraction.’ Advisor Professor David C. Wilkins, co-advisor Professor David E. Goldberg. Anticipated graduation date is May 1999.
- Research interests: artificial intelligence and knowledge-based systems, in particular uncertainty reasoning using Bayesian networks; evolutionary computation; machine learning and knowledge acquisition; intelligent tutoring systems and expert critiquing systems.

B.S., Computer Science, Norwegian Institute of Technology (NTH), Trondheim, Norway (Aug 1984 – Apr 1989).

- Specialization in computer networks and artificial intelligence.
- Undergraduate thesis on natural language interfaces to databases.

Experience

Research Assistant, University of Illinois, Urbana-Champaign, Illinois (Jan 1995 – present).

- Member of the Knowledge-Based Systems Group, headed by Professor David C. Wilkins. The KBS Group focuses on the interplay between expert system shell architectures, machine learning, probabilistic reasoning, and immersive expert interfaces.
- Research and develop approximate inference in hard Bayesian networks using genetic algorithms and stochastic local search techniques as well as abstraction; compare performance to exact algorithms.
- Head KBS Bayesian network group consisting of undergraduate and graduate students developing the RAVEN tool for Bayesian network representation, inference, and visualization. The RAVEN tool is implemented in MS Visual C++ (back-end) and Java (front-end), and provides a generic environment that is currently part of CoRAVEN, a set of tools geared towards enhancing situation awareness for military intelligence analysts.
- Research and development of techniques and tools for expert critiquing, plan recognition, and intelligent tutoring, in particular for Naval ship damage control. This research is part of developing a multi-media system, with an expert critiquing component, for crisis management training for the Navy.
- Participate and coordinate writing of proposals for grants, holding reviews, demonstrating software prototypes; present results of research at conferences, symposia, and workshops.

Summer Intern, First Quadrant, Pasadena, California (May 1998 – Aug 1998).

- Worked in the group headed by Dr. David J. Leinweber at First Quadrant. First Quadrant is a financial asset manager which manages portfolios for large corporate investors.
- Designed and developed multi-objective and niching genetic algorithm (MNGA) in C++ under Unix, interfacing to First Quadrant's model engineering system including regression and user interface software.

- Received 100% bonus, the highest bonus ever received by a Summer Intern at First Quadrant, based on the success of the software.

Research Assistant, University of Illinois, Urbana-Champaign, Illinois (Aug 1994 – Jan 1996).

- Solid Modeling group, headed by Professor Yong Se Kim.
- Research and development in intelligent tutoring and expert critiquing systems for teaching spatial reasoning skills.
- Led team in design and development of a spatial reasoning tutor Visual Teacher, implemented in C and the rule-based language Clips. The Visual Teacher, which is based on ‘missing view’ problems, is used in the introductory engineering course GE103 at the University of Illinois to enhance spatial visualization abilities of students.

Research Scientist, SINTEF DELAB, Trondheim, Norway (Apr 1989 – Aug 1993).

- Knowledge Engineering and Image Processing Group, headed by Ingeborg Sølvsberg.
- Research and development of techniques and tools for knowledge engineering, in particular a knowledge validation tool (KVAT) and a tool for interchange between different knowledge acquisition tools (KRF). Both KVAT and KRF are implemented in an object-oriented package on top of Common Lisp, and both included graphical users interfaces.
- Participated in the ESPRIT III project UniTe. The purpose of UniTe was to integrate uncertain, incomplete and temporal reasoning. This was done by performing theoretical work on integration of these reasoning types, developing a methodology and a software platform, and, finally, building two applications. The applications concerned planning in the context of prematurely born infants and temporal and uncertainty reasoning during satellite testing. Participated in UniTe methodology development and development of prototypes for fact interchange. Specified, designed, and implemented the Knowledge Reformulation Tool, a tool for translation and interchange of knowledge during knowledge acquisition. KRF is a part of the knowledge engineering workbench KEW (see below).

- Researched and developed knowledge-based systems including data-, knowledge- and enterprise models. Developed an enterprise model for the Norwegian Directorate of Nature Management. The purpose of the model is to support the directorate in different aspects of their activities, especially to evaluate actions initiated by the directorate itself. Interviewed experts at the Directorate. The enterprise model was partly based on the interviews, partly on written knowledge sources.
- Participated in the development of the software system the Knowledge Engineering Workbench (KEW). KEW was developed in the ACKnowledge project (ESPRIT II). The project emphasized integration of tools for knowledge acquisition and machine learning. In addition to these tools, the workbench contains tools for knowledge integration, validation, transformation, and a set of editors. KEW was implemented in Common Lisp using the object-oriented user interface package PCE, and runs under Unix on Sun workstations.

Specified, designed, and developed the knowledge validation tool KVAT, and participation in work on conceptualizing KEW, knowledge transformation, knowledge representation, KEW functional specification, and KEW evaluation. Participated in SINTEF DELAB's work in developing a help desk assistant for the Norwegian insurance company UNI Storebrand A/S, including knowledge elicitation, acquisition, and prototyping using KEW.

- Participated in feasibility study, requirement specification, knowledge acquisition and early design of knowledge-based system (KBS) in the GALENO-2000 health-check station (EUREKA-project EU26). GALENO-2000 is based on non-invasive measurements of heart- and lung-activity during exercise testing, and the objective is to integrate in one system function measurement, patient administration, and knowledge-based decision support.
- Supervised NTH students towards the degree of Sivilingeniør (B.S.) in Computer Science. Examination of Sivilingeniør students at NTH.

Teaching Assistant, Norwegian Institute of Technology, Trondheim, Norway (Aug 1985 – Apr 1989).

- TA in the Department of Computer Science, in the following courses: Introductory course in computer science, Data bases, and Symbolic computation.

Programmer, Basic Communication Group, Norsk Data A/S, Oslo, Norway (Summers 1987 and 1988).

- *Summer 1988*: Designed, developed and tested local area network software facilitating communication between PCs. The software supported a client-server model of computing, and was implemented using threads in Microsoft C and Microsoft Lan Manager under OS/2.
- *Summer 1987*: Designed and implemented prototype user interface for a network monitor. The program was implemented on a PC-AT using Microsoft C under Microsoft Windows.

Programmer, RegnskapsRevisjon A/S, Hamar, Norway (Summer 1986).

- Designed and developed time and cost accounting software for the accounting firm RegnskapsRevisjon A/S. The software was implemented using dBase-III + on a PC-AT. The systems computes how much to bill each client based on data on hours spent and other costs per employee in RegnskapsRevisjon A/S.

Publications

Journal Publications

- Hubbard, C., O. J. Mengshoel, C. Moon, and Y. S. Kim. ‘Visual Reasoning Instructional Software System.’ In *Computers and Education*, 28(4), 1997.
- Mengshoel, O. J., S. Chauhan, and Y. S. Kim, ‘Intelligent Critiquing and Tutoring of Spatial Reasoning Skills.’ In *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 10(3):235–249, June 1996.
- Mengshoel, O. J., ‘A Reformulation Technique and Tool for Knowledge Interchange during Knowledge Acquisition.’ In *International Journal of Human-Computer Studies*, 43(2):177–212, 1995.
- Mengshoel, O. J., ‘Knowledge Validation — Principles and Practice.’ In *IEEE Expert*, 8(3):62–68, June 1993.

Strongly Refereed Conference and Workshop Publications

- Mengshoel, O. J. and D. E. Goldberg, ‘Probabilistic Crowding: Deterministic Crowding with Probabilistic Replacement.’ Accepted to *Genetic and Evolutionary Computation Conference (GECCO-99)*, July 13–17, Orlando, FL.
- Wilkins, D. C., O. J. Mengshoel, O. Chernyshenko, P. M. Jones, C. C. Hayes, and R. Bargar. ‘Collaborative Decision Making and Intelligent Reasoning in Judge Advisor Systems.’ In *Proceedings Hawaii International Conference on System Sciences (HICSS-32)*, January 1999.
- Jones, P. M., C. C. Hayes, D. C. Wilkins, R. Bargar, J. Sniezek, P. Asaro, O. J. Mengshoel, D. Kessler, M. Lucenti, I. Choi, N. Tu, and J. Schlabach, ‘CoRAVEN: Modeling and Design of a Multimedia Intelligent Infrastructure for Collaborative Intelligence Analysis.’ In *1998 IEEE International Conference on Systems, Man, and Cybernetics*, October 11-14, 1998, San Diego, CA.
- Mengshoel, O. J., D. E. Goldberg, and D. C. Wilkins, ‘Deceptive and Other Functions of Unitation as Bayesian Networks.’ In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 559–566, July 22-25, 1998, University of Wisconsin, Madison, WI.
- Mengshoel, O. J. and D. C. Wilkins. ‘Genetic Algorithms for Belief Network Inference: The Role of Scaling and Niching’. In *Proc. of the Seventh Annual Conference on Evolutionary Programming*, pp. 547–556, San Diego, CA, March 25–27, 1998.
- Mengshoel, O. J. and D. C. Wilkins, ‘Abstraction for Belief Revision: Using a Genetic Algorithm to Compute the Most Probable Explanation.’ In *Satisficing Models: Papers from the 1998 AAAI Spring Symposium*, pp. 46- 53, Technical Report SS-98-05, AAAI Press, March 1998, Stanford University, CA.
- Mengshoel, O. J. and D. C. Wilkins, ‘Abstraction and Aggregation in Belief Networks.’ In *Abstractions, Decisions, and Uncertainty: Collected Papers from the 1997 Workshop*. pp. 53–58, Technical Report WS-97-08, AAAI Press, Menlo Park, CA, July 1997.
- Mengshoel, O. J. and D. C. Wilkins, ‘Recognition and Critiquing of Erroneous Agent Actions.’

In *Agent Modeling: Papers from the 1996 AAAI Workshop*, M. Tambe and P. Gmytrasiewicz (eds.), pp. 61–68. Technical Report WS-96-02, AAAI Press, Menlo Park, CA, August 1996.

- Mengshoel, O. J. and D. C. Wilkins, ‘Towards an Approach to Exploiting Domain Structure for Planning.’ In *Proc. AAAI Workshop on Structural Issues in Planning and Temporal Learning*, Portland, OR, August 1996.
- Hubbard, C., O. J. Mengshoel, C. Moon, and Y. S. Kim, ‘Multimedia Instructional Software for Visual Reasoning: Visual Reasoning Tutor (VRT).’ In *Proc. IEEE International Conference on Multimedia Computing and Systems*, Hiroshima, Japan, June 1996.
- Mengshoel, O. J., ‘Definite Clause Grammars for Knowledge Interchange during Knowledge Acquisition.’ In *Proc. Knowledge Acquisition Workshop*, B. R. Gaines and M. Musen (eds.), pp. 11.1–11.20, Banff, Canada, January 1994.
- Mengshoel, O. J. and I. Sølvberg, ‘Acquisition and Modelling of Uncertain, Incomplete and Time-Varying Knowledge.’ In *Proceedings of 7th European Knowledge Acquisition for Knowledge-Based Systems Workshop*, Toulouse, France, September 1993. Also printed in *Lecture Notes in Artificial Intelligence*, N. Aussenac et al. (ed.), pp. 300–319. Springer-Verlag, Heidelberg, 1993.
- Nordbø, I., O. J. Mengshoel, and I. Sølvberg, ‘Development of Complex Applications: Integration Support for Time, Uncertainty, and Incompleteness.’ In *Proc. Second World Congress on Expert Systems*, Lisboa, Portugal, January 1993.
- Mengshoel, O. J., I. Nordbø, and I. Sølvberg, ‘The Knowledge Engineering Workbench: Experiences from Building the Knowledge Base of a Helpdesk Application.’ In *Proc. Twelfth International Conference on Artificial Intelligence, Expert Systems and Natural Language*, pp. 53–65, Avignon, France, June 1992.
- Mengshoel, O. J., ‘KVAT: A Tool for Incremental Knowledge Validation in a Knowledge Engineering Workbench,’ In *Proceedings of 1st European Workshop on the Verification and Validation of Knowledge-Based Systems*, pp. 133–146. Logica Cambridge, Cambridge, England, July 1991.

Less Strongly Refereed Conference and Workshop Publications

- Mengshoel, O. J., D. C. Wilkins, and S. Uckun, ‘Filtering and Visualizing Uncertain Battlefield Data Using Bayesian Networks.’ In *Proc. ARL Advanced Displays and Interactive Displays Federated Laboratory Second Annual Symposium*, College Park, MD, February 1998.
- Mengshoel, O. J. and D. C. Wilkins, ‘Visualizing Uncertainty in Battlefield Reasoning Using Belief Networks.’ In *Proc. ARL Advanced Displays and Interactive Displays Federated Laboratory First Annual Symposium*, pp. P15–P22, Adelphi, MD, ARL Adelphi Laboratory, January 1997.
- Kim, Y. S., C. Moon, S. Chauhan, C. Hubbard, O. J. Mengshoel, and H. Zhao, ‘Visual Reasoning Tutor (VRT): Instructional Software System for Missing View Problem.’ In *Proc. ASEE Engineering Design Graphics Conf.*, Ames, November 1995.

Abstracts and Position Statements

- Mengshoel, O. J. ‘Evolutionary Computation in Bayesian Networks.’ In *Late Breaking Papers of the Third Annual Genetic Programming Conference*, p. 159, July 1998, Madison, WI.
- Mengshoel, O. J. ‘Belief Network Inference in Dynamic Environments.’ In *Proc. AAAI 97*, p. 813, Providence, RI, July 1997.