AN ABSTRACT OF THE THESIS OF

Masami Takikawa for the degree of Doctor of Philosophy in Computer Science presented on October 15, 1998.

Title:

Representations and Algorithms for Efficient Inference in Bayesian Networks.

Abstract approved: Redacted for Privacy

Bruce D'Ambrosio

Bayesian networks are used for building intelligent agents that act under uncertainty. They are a compact representation of agents' probabilistic knowledge. A Bayesian network can be viewed as representing a factorization of a full joint probability distribution into the multiplication of a set of conditional probability distributions. Independence of causal influence enables one to further factorize the conditional probability distributions into a combination of even smaller factors. The efficiency of inference in Bayesian networks depends on how these factors are combined. Finding an optimal combination is NP-hard.

We propose a new method for efficient inference in large Bayesian networks, which is a combination of new representations and new combination algorithms. We present new, purely multiplicative representations of independence of causal influence models. They are easy to use because any standard inference algorithm can work with them. Also, they allow for exploiting independence of causal influence fully because they do not impose any constraints on combination ordering. We develop combination algorithms that work with heuristics. Heuristics are generated automatically by using machine learning techniques. Empirical studies, based on the CPCS network for medical diagnosis, show that this method is more efficient and allows for inference in larger networks than existing methods.

Representations and Algorithms for Efficient Inference in Bayesian Networks

by

Masami Takikawa

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed October 15, 1998
Commencement June 1999

Doctor of Philosophy thesis of Masami Takikawa presented on October 15, 1998

APPROVED:

Redacted for Privacy

Major Professor, representing Computer Science

Redacted for Privacy

Chair of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Masami Takikawa, Author

ACKNOWLEDGEMENT

I would like to express my deep appreciation to my major advisor Professor Bruce D'Ambrosio for all the help and support he provided me with during the course of this research.

I am grateful to the other members of my Ph.D. committee, Dr. Bella Bose, Dr. Timothy A. Budd, Dr. Prasad Tadepalli, and Dr. Terry M. Wood for their helpful comments and encouragement.

Special thanks are due to Dr. Tom Dietterich for insightful comments and discussions, to Dr. Jane Jorgensen for her continuous encouragement and thorough editorial comments on earlier drafts of this dissertation, and to Makoto Kimura for his support throughout my student life both in Japan and in the US.

This work was supported partially by the NSF under grant number IRI-9704232 through my major advisor.

Finally, I would like to thank my wife Midori and my children, Asumu, Ichigo, and Towaki, for their constant loving support and understanding.

TABLE OF CONTENTS

Page

1 Introduction 1

1.1 Probabilities 2

1.2 Bayesian Networks 4

1.3 Optimal Factoring Problem 5

1.4 Independence of Causal Influence 6

1.5 Reinforcement Learning 6

1.6 Thesis Objectives 7

1.7 Outline of the Dissertation 8

2 Related Work 9

2.1 Transformational Algorithms 9

2.2 Direct Computation Algorithms 10

2.3 Independence of Causal Influence 11

3 Representations of Independence of Causal Influence 13

3.1 Local Expression Languages 13

3.2 Independence of Causal Influence 14

3.3 Additive Factorization of ICI 15

3.4 Multiplicative Factorization of ICI 16

3.5 Optimization of Multiplicative Factorization 18

3.5.1 Multiplicative Factorization of Noisy-Or 18
3.5.2 Multiplicative Factorization of Noisy-Max 20
3.5.3 Multiplicative Factorization of Contract Renewal Model 23

3.6 Summary and Discussion of Multiplicative Representations 26

3.7 The Limitations of Multiplicative Factorization 29

TABLE OF CONTENTS (Continued)

Page

3.8 Experimental Comparison of Additive and Multiplicative Representations 33

3.8.1 Experiments using Noisy-Or models 33
3.8.2 Experiments using Noisy-Max models 35

3.9 Summary 36

4 Factoring Algorithms 38

4.1 Algorithm Template 38

4.2 Optimal Factoring Algorithm 40

4.3 SPI Factoring Algorithm 42

4.4 Variable Elimination 42

4.5 Greedy Pair-Combination Algorithms 44

5 Reinforcement Learning Approach to Factoring 46

5.1 Reinforcement Learning 46

5.2 Factoring Algorithms with Future Cost Estimators 47

5.2.1 Pair-Combination Algorithms with Estimators 48
5.2.2 Variable-Elimination Algorithms with Estimators 50

5.3 Future Cost Estimators 53

5.3.1 Function Approximators 54
Multilayer Perceptrons 54
Training Multilayer Perceptrons Using Temporal Difference 55
Radial-Basis Function Networks 55
Training RBFN 56
5.3.2 State Encoding 59
Feature Set 60
Matrix Representation 61
Feature Set v.s. Matrix Representation 62
Significant Subportion Representation 63
5.3.3 Cost Encoding 64

TABLE OF CONTENTS (Continued)

Page

5.4 Experimental Results 64

5.4.1 Training Estimators Using a Fixed Policy 65
5.4.2 On-line Training of Estimators 68
5.4.3 Generalizability 72
The Prediction Test Using The CPCS Network 72
The Prediction Test Using The QMR-DT Network 74

5.5 Summary 76

6 Conclusion 78

Bibliography 80

LIST OF FIGURES

Figure Page

1.1. A simple multiply connected Bayesian network. 5

3.1. $(L+M)(L+M) - LL = LM + MM + ML$. 21

3.2. Three dimensional table representing $\xi_1 * \xi_2 * \xi_3$. 24

3.3. Three example operators. 30

3.4. Six non-trivial operators that can be efficiently represented. 32

4.1. Factoring algorithm template. 39

4.2. The function combinePair. 40

4.3. Random factoring algorithm. 40

4.4. Optimal factoring algorithm. 41

4.5. SPI factoring algorithm. 42

4.6. Greedy-combination algorithm based on the number of multiplications. 44

4.7. Greedy-combination algorithm based on dimension. 45

5.1. Pair-combination algorithm with estimators for the number of multiplications. 48

5.2. Pair-combination algorithm with estimators for the maximum dimensionality. 49

5.3. Variable-elimination algorithm with estimators for the number of multiplications. 51

LIST OF FIGURES (Continued)

Figure Page

5.4. Variable-elimination algorithm with estimators for the maximum dimensionality. 52

5.5. Contribution-based center selection. 57

5.6. RBFN with the contribution-based center selection. 57

5.7. Clustering-based center selection. 58

5.8. RBFN with the clustering-based center selection. 59

5.9. The evaluation of various configuration of estimateDim. 65

5.10. The evaluation of various configuration of estimateMul. 66

LIST OF TABLES

Table Page

3.1. Summary of multiplicative representations. 27

3.2. The number of operators classified by the number of intermediate variables required. 30

3.3. The characteristics and results of QMR-DT BN2O test cases. 34

3.4. The characteristics and results of the CPCS marginal queries. 36

5.1. The results of on-line training of estimators. 71

5.2. The prediction test using four most difficult queries in the CPCS network. 73

5.3. The prediction test using Scientific American cases in the QMR-DT network. 75

Representations and Algorithms for Efficient Inference in Bayesian Networks

Chapter 1
Introduction

Most everyday reasoning and decision making is based on uncertain premises. Most of our actions are based on guesses, often requiring explicit weighing of conflicting evidence.

Bayesian networks are used for building intelligent agents that act under uncertainty. They are a compact representation of agents' probabilistic knowledge. Bayesian networks have been built for many applications including monitoring patients in intensive care [Beinlich *et al.*, 1989], evaluating car insurance risks [Binder *et al.*, 1997], and diagnosing lymph-node diseases [Heckerman *et al.*, 1992].

Inference in Bayesian networks is very time-consuming. In the general case, exact inference in Bayesian networks is known to be NP-hard [Wen, 1990, Cooper, 1990]. For very large networks, approximation using stochastic simulation is currently the method of choice. However, the problem of approximating within an arbitrary tolerance is itself NP-hard [Russell and Norvig, 1995].

For example, none of the existing exact inference systems are able to make inference with a Bayesian network called the CPCS network created by Pradhan *et al.* [1994] based on the Computer-based Patient Case Simulation system (CPCS-PM), developed by R. Parker and R. Miller [1987]. Development of the CPCS network was simplified by exploiting probabilistic structures and extending Bayesian networks to represent such structures, but existing inference systems cannot exploit them fully in order to make efficient inference. Developing such an efficient Bayesian inference system is the topic of this dissertation.

1.1 Probabilities

Uncertainty is inescapable in complex, dynamic, or inaccessible worlds. Therefore, a rational person or machine, which we call an *agent*, in such a world must act under uncertainty.

Probabilities have been used for centuries as the right way to handle uncertainties. Probability theory assigns a numerical degree of belief between 0 and 1 to propositional sentences. A probability of 0 for a given sentence corresponds to an unequivocal belief that the sentence is false, while a probability of 1 corresponds to an unequivocal belief that the sentence is true. Probabilities between 0 and 1 correspond to intermediate degrees of belief in the truth of the sentence.

The probability that is assigned to a sentence depends on the prior knowledge obtained to date. In discussing uncertain reasoning, we call this *evidence* (or the *background state of information*). As the agent receives new information, its probability assessments are updated to reflect the new evidence. Before the evidence is obtained, we talk about *prior*, *unconditional*, or *marginal* probability; after the evidence is obtained, we talk about *posterior* or *conditional* probability.

We will use the notation $P(X = x_i)$ for the prior probability that the *random variable* X takes a value x_i. For example, if we are concerned about the random variable *Weather*, we might have

$$P(Weather = Sunny) = 0.5$$
$$P(Weather = Rain) = 0.2$$
$$P(Weather = Cloudy) = 0.28$$
$$P(Weather = Snow) = 0.02.$$

Each random variable X has a *domain* of possible values $\langle x_1, \ldots, x_n \rangle$ that it can take on.

In the case that we want to talk about the probabilities of all possible values of a random variable, we will use an expression such as $P(Weather)$, which denotes a vector of values for the probabilities of each individual state of the weather. Given

the preceding values, for example, we would write

$$P(\mathit{Weather}) = \langle 0.5, 0.2, 0.28, 0.02 \rangle.$$

This statement defines a *probability distribution* for the random variable *Weather*. It is sometimes convenient to combine a variable, its domain, and numerical values all together to write a probability distribution as follows:

$$P(\mathit{Weather} : \langle \mathit{Sunny},\ \mathit{Rain},\ \mathit{Cloudy},\ \mathit{Snow} \rangle, \langle 0.5, 0.2, 0.28, 0.02 \rangle).$$

We sometimes want to write a density function instead of a numeric vector like this:

$$P(\mathit{Weather} : \langle \mathit{Sunny},\ \mathit{Rain},\ \mathit{Cloudy},\ \mathit{Snow} \rangle, \langle \mathit{func}(\mathit{Weather}) \rangle).$$

We will also use expressions $P(X, Y)$ to denote the *joint probability distribution*, that is, the probabilities of all combinations of the values of a set of random variables. Conditional or posterior probability distributions are written as $P(X|Y)$, which is a two-dimensional table giving the values of $P(X = x_i|Y = y_j)$ for each possible i, j. We call X a *conditioned variable* and Y a *conditioning variable*. Conditional probabilities can be defined in terms of unconditional probabilities. The equation

$$P(X|Y) = \frac{P(X, Y)}{P(Y)},$$

holds whenever $P(Y = y_i) > 0$. This equation can also be written as

$$P(X, Y) = P(X|Y) \times P(Y)$$

which is called the *product rule* or *chain rule*.

The *full joint probability distribution* (or "full joint" for short) is the joint probability distribution of all domain variables. It completely specifies an agent's probability assignments to all propositions in the problem domain. It is important to note that any probabilistic query can be answered based on the full joint. In general, however, it is not practical to define all the 2^n entries for the full joint probability distribution over n Boolean variables.

1.2 Bayesian Networks

A Bayesian network is a powerful tool for representing and reasoning with uncertain knowledge. It is based on the observation that conditional independence relationships among variables can both simplify the computation of query results and also greatly reduce the number of conditional probabilities that must be specified.

A Bayesian network [Pearl, 1988] is a directed acyclic graph containing a set of nodes, a set of arcs, and a set of numeric probability distributions. A node represents a domain variable with mutually exclusive and exhaustive values. Arcs and numeric probability distributions describe the probabilistic relationship among nodes.

Probabilistic inference in a Bayesian network is the task of computing a marginal or conditional probability distribution across some subset of the variables in the network, given evidence on some subset of the remaining variables. A Bayesian network is a concise representation of a full joint probability distribution over the n domain variables in the network. The full joint probability distribution can be calculated as follows [Pearl, 1988, Shachter, 1988]:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i|\pi_i)$$

where $x_1, \ldots, x_n$ are n variables in the network; π_i is the set of direct predecessors of x_i; $P(x_i|\pi_i)$ is the conditional probability for variable x_i if π_i is not the empty set, otherwise it is the marginal probability of x_i. The product of any two terms of the formula is called a *conformal product*, the number of variables appearing in a conformal product is called the *dimension* of the conformal product, and the maximum number of variables in any of the conformal products for a query is called the *maximum dimensionality* of the conformal products (or the query), or the *dimensionality* for short.

An important characteristic of a Bayesian network is that it uniquely defines the full joint probability distribution over the variables involved, and that this distribution can be recovered as described above. Any marginal, conditional, or conjunctive query can be answered simply by instantiating all observed variables in the formula,

evaluating it, and summing over variables not in the query. The time and space complexity of computing the full joint probability distribution of a Bayesian network is exponential in the number of nodes of the network.

1.3 Optimal Factoring Problem

The computational cost of inference in a Bayesian network can be reduced if variables can be summed early in the computation, rather than waiting to perform all marginalization after computing the full joint. The efficiency of probabilistic inference in a Bayesian network, then, depends on finding a factoring of the expression for the joint over the relevant set of variables which permits early marginalization of variables not in the query.

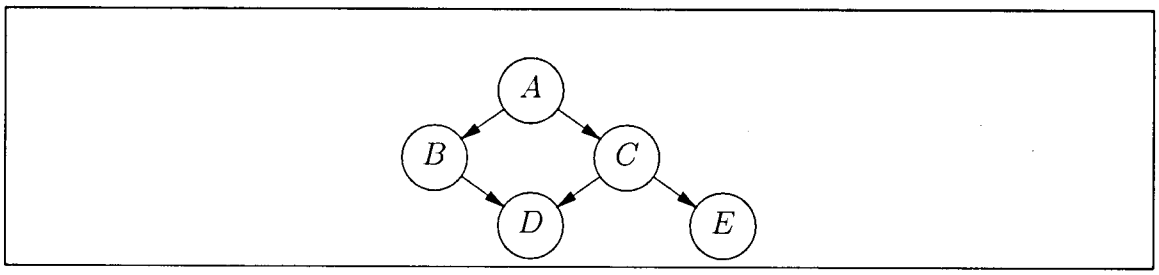


Figure 1.1. A simple multiply connected Bayesian network.

For example, given the simple Bayesian network in figure 1.1, the joint probability of nodes D and E, $P(D, E)$, can be computed in several ways. One factoring is given in the formula:

$$P(D,E) = [\sum_{A}[\sum_{B}[\sum_{C}[P(E|C) \times P(D|B,C)] \times P(C|A)] \times P(B|A)] \times P(A)]$$

which requires 72 multiplications.[1] The maximum dimensionality of the above factor-

[1]We assume all variables are binary.

ing is 5, so the maximum size of intermediate distributions required by the factoring is $2^5 = 32$.[2] The factoring below needs only 28 multiplications:

$$P(D,E) = [\sum_C [P(E|C) \times [\sum_B P(D|B,C) \times [\sum_A P(C|A) \times [P(B|A) \times P(A)]]]]].$$

The dimensionality is also reduced to 3, and the maximum distribution size to $2^3 = 8$.

From this example, we can see that different factorings yield significantly different computational costs.

Li and D'Ambrosio [1994] formally defined the *optimal factoring problem* — a combinatorial optimization problem to find a factoring of the minimum cost.

1.4 Independence of Causal Influence

Conditional independences allow for factoring the full joint probability distribution into smaller distributions which are easier to create and use. *Independence of causal influence*[3] (ICI) among local parent-child or cause-effect relationships allows for further factoring. The noisy-or interaction [Pearl, 1988, Srinivas, 1993] is one of the best studied and most widely used models of ICI.

ICI has been utilized to reduce the complexity of knowledge acquisition [Pearl, 1988, Henrion, 1987], as well as the complexity of inference [Kim and Pearl, 1983, Zhang and Poole, 1994]. The CPCS network has also benefited from ICI; the development of the network was simplified by exploiting ICI. As a result, the CPCS network contains abundant explicit causal independences.

1.5 Reinforcement Learning

Reinforcement learning [Russell and Norvig, 1995] is a technique for a software agent to improve its performance without requiring a teacher who provides a correct answer in each situation.

---

[2]The maximum size of intermediate distributions required can be reduced to $2^4 = 16$ by combining the conformal product and marginalization in one operation.

[3]Also known as *causal independence* and *intercausal independence*.

Reinforcement learning algorithms learn *policies* for state-space problem-solving tasks. For each state, the policy specifies what action should be performed. During learning, the learning system receives a reinforcement signal (called a "reward") after each action. The goal of the learning system is to find a policy that maximizes the expected reinforcement over future actions.

Reinforcement learning techniques have been applied to discover good domain-specific heuristics automatically. For example, Tesauro's work on TD-gammon [1992] showed that a reinforcement learning algorithm, called the temporal difference algorithm $TD(\lambda)$ [Sutton, 1988], can learn an excellent evaluation function for the game of backgammon. Zhang and Dietterich [1995] applied $TD(\lambda)$ to job-shop scheduling, which is an NP-complete combinatorial optimization problem, and also succeeded in obtaining good heuristics.

1.6 Thesis Objectives

The goal of this thesis is to develop a Bayesian network inference system that performs well with large Bayesian networks. There are two approaches we employ to tackle the problem. The first one relates to representations and the second one relates to algorithms.

First, we exploit independence of causal influence. In order to fully exploit ICI, a new representation is called for. That representation should reduce the complexity of inference without sacrificing the reduction in the complexity of knowledge acquisition. Designing such representation is the first objective of this thesis.

Finding optimal factoring is, in general, intractable, so efficient heuristics that provide good, but not necessarily optimal solutions are needed. However, the process of finding and engineering such heuristics by hand is expensive and time consuming. Developing good heuristics automatically using reinforcement learning techniques is the second objective of this thesis.

1.7 Outline of the Dissertation

- Chapter 2 reviews some related work.
- Chapter 3 presents a new representation of independence of causal influence. Experimental study of the new representation using an existing factoring algorithm is also given in this chapter.
- Chapter 4 studies factoring algorithms.
- Chapter 5 describes our approach to developing factoring heuristics using reinforcement learning techniques. We present empirical results comparing the performance of our and existing algorithms in this chapter.
- Chapter 6 summarizes the dissertation and presents some directions for further research.

Chapter 2
Related Work

This chapter reviews some related work. The first two sections review existing algorithms for exact inference in Bayesian networks. The last section discusses representation issues for independence of causal influence.

2.1 Transformational Algorithms

A number of algorithms have been developed to perform exact probabilistic inference in Bayesian networks in recent years.[Lauritzen and Spiegelhalter, 1988, Pearl, 1988, Jensen *et al.*, 1990, Shafer and Shenoy, 1990] Many of them first transform Bayesian networks into *singly-connected* networks, also known as *polytrees*. In such networks, there is at most one undirected path between any two nodes in the network. Efficient polynomial-time inference algorithms exist for singly-connected Bayesian networks. In the general case, exact inference in Bayesian networks is known to be NP-hard [Wen, 1990].

There are two classes of algorithms for transforming multiply-connected networks into polytrees:

Clustering methods transform the network into a probabilistically equivalent (but topologically different) polytree by merging offending nodes.

Conditioning methods do the transformation by instantiating a cutset of the variables to definite values, and then evaluating a polytree for each possible instantiation.

There are a number of ways to cluster or condition a given multiply-connected network. These algorithms heuristically search for the optimal way to cluster or condition that minimizes the total computational cost. In [Kjærulff, 1993], Kjærulff developed several new heuristics and compared them with existing ones.

In some sense, these algorithms can be considered as algorithms that try to create one factoring that works for all queries. In most cases, however, the optimal factoring varies for each query and evidence. For this reason, these algorithms are mainly used for answering multiple queries, such as the "all marginal" query, for which the clustering method is popular choice. The direct computation method described below could be used when only a small set of queries needs to be answered.

2.2 Direct Computation Algorithms

SPI[Shachter *et al.*, 1990] and ElimBel [Dechter, 1996] are based on direct application of the chain rule, rather than transforming networks prior to the computation.

SPI[Shachter *et al.*, 1990] answers a query by first creating an optimal factoring for the query. It tries to find an optimal factoring by using *set-factoring heuristics* [Li and D'Ambrosio, 1994]. Set factoring heuristics assign a value to each pair (or, in general, a set [D'Ambrosio, 1995]) of probability distributions. That value represents some goodness of the combination of those distributions. In each step, a combination that has the maximum heuristic value is chosen and combined.

The current best heuristic value function is created by *simulated annealing.* The value function is represented by a discrete linear combination of some *features* of the pair. The simulated annealing starts with a set of randomly generated heuristic functions. Then, it evaluates each function by using it for actual probabilistic inferences. Better functions are selected and mutated randomly according to a probability parameter called a *temperature* which decreases as search proceeds. The temperature is used to avoid local optima. This process repeats some predefined number of times, and the final best function is output.

Although the heuristic function currently employed by SPI works well in many

cases, it often fails to find optimal factorings because the locally best combination does not necessarily lead to a globally best factoring. The desirability of an action should depend not only on the pair of distributions to be combined by the action, but also on the remaining distributions that are combined later with the result of the action.

Also, the simulated annealing algorithm used for searching the heuristic function is very slow, both to compute and to converge. A more efficient learning approach is desired.

2.3 Independence of Causal Influence

Three different approaches have been proposed to represent ICI and to integrate the correspondence models into standard Bayesian network inference. Local expression languages [D'Ambrosio, 1995] provide a comprehensive approach for integration of many local structure models, including ICI, into standard Bayesian networks. An additive factorization of ICI using the local expression language is presented in [D'Ambrosio, 1995]. Heterogeneous factorization [Zhang and Poole, 1994, Zhang and Poole, 1996] and temporal belief networks [Heckerman and Breese, 1994] are two other major approaches which are capable of representing many forms of causal independences.

However, the results obtained by these approaches are not satisfactory. Borujerdi *et al.* [1998] analyze these three approaches and identify their weaknesses. One of weaknesses is that they impose unnecessary constraints on factoring, severely limiting their ability to find optimal factoring. Zhang [1995] reports experiments with heterogeneous factorization on the CPCS network for medical diagnosis, created by Pradhan *et al.* [1994], and shows that the algorithm is unable to answer two out of 422 possible zero-observation queries, for example. Zhang and Yan [1997] extend a message passing algorithm (clique tree propagation [Lauritzen and Spiegelhalter, 1988]) with heterogeneous factorization and show that the resulting algorithm is significantly more efficient than that of [Zhang and Poole, 1994, Zhang and Poole,

1996], but it is unable to deal with CPCS network because it runs out memory when initializing clique trees.

Chapter 3
Representations of Independence of Causal Influence

In this chapter, we study the representations of ICI models. We develop a purely multiplicative representation of ICI based on the SPI local expression language. We first present the most general form, which requires very large tables. We show how that form can be optimized to reduce table size. We then present examples of efficient optimized (specialized) forms such as noisy-or and noisy-max. Unfortunately, not all forms can be optimized satisfactorily as shown below. Finally, we compare the performance of the new multiplicative representation and an existing additive representation using SPI's set-factoring algorithm.

3.1 Local Expression Languages

Local expression languages [D'Ambrosio, 1995] provide a comprehensive approach for integration of many local structure models, including ICI, into standard Bayesian networks. The formal syntax is defined recursively as follows[4]:

$$\begin{aligned} exp \;\rightarrow\; & \mathit{distribution} \mid \\ & exp \times exp \mid \\ & exp + exp \mid \\ & exp - exp, \end{aligned}$$

where *distribution* is defined over some rectangular subspace of the Cartesian product of domains of its conditioned and conditioning variables. It is a generalization of the standard probability distribution defined in Section 1.1 because it does not have to

[4]Our definition is a simplification of the original of [D'Ambrosio, 1995]

be normalized and it can contain any numeric values such as negative numbers. Note that generalized distributions make subtraction expressions redundant because any subtraction can be converted into an addition by negating all numeric values of the second distribution.

The semantics of local expressions are quite simple to specify:

> An expression is equivalent to the distribution obtained by evaluating it using the standard rules of algebra for each possible combination of antecedent values, where a distribution contributes 0 when being evaluated for a parent case over which it is not defined.

3.2 Independence of Causal Influence

We now formalize independence of causal influence, following [Zhang and Poole, 1996, Zhang and Yan, 1997] with some adaptation. *Independence of causal influence* (ICI) refers to the situation where several causes (variables) $c_1, c_2, \ldots, c_m$ contribute independently to an effect (variable) e.

More precisely, $c_1, c_2, \ldots, c_m$ are said to *influence e independently* if there exist random variables $\xi_1, \xi_2, \ldots, \xi_m$ that have the same domain as e such that

1. for each i, ξ_i probabilistically depends on c_i and is conditionally independent of all other c_j's and all other ξ_j's given c_i; and
2. there exists a commutative and associative binary operator $*$ over the domain of e such that $e = \xi_1 * \xi_2 * \ldots * \xi_m$.

We will refer to ξ_i as the *contribution* of c_i to e.

In an ICI model, the conditional probability $P(e|c_1, c_2, \ldots, c_m)$ can be obtained from the conditional probabilities $P(\xi_i|c_i)$ as follows:

$$P(e = \alpha|c_1, c_2, \ldots, c_m) = \sum_{\alpha_1 * \alpha_2 * \ldots * \alpha_m = \alpha} \prod_{i=1}^{m} P(\xi_i = \alpha_i|c_i). \tag{3.1}$$

In order to get sound results, we require that each row of $P(\xi_i|c_i)$ adds up to 1, as stated in the following requirement.

Requirement 3.2.1 $\sum_{\alpha_i} P(\xi_i = \alpha_i | c_i = \beta) = 1$ *for each value* β *of any cause* c_i.

3.3 Additive Factorization of ICI

Let $f_i(e, c_i)$ be the function defined by

$$f_i(e = \alpha, c_i) = P(\xi_i = \alpha | c_i), \tag{3.2}$$

for each possible value α of e. Then, by using the notation, defined in Section 1.1, of a distribution with a density function, we can write the conditional probability distribution $P(e|c_1, c_2, \ldots, c_m)$ in local expression languages as follows:

$$P(e|c_1, c_2, \ldots, c_m) = \sum_{\alpha} \sum_{\alpha_1 * \alpha_2 * \ldots * \alpha_m = \alpha} \prod_{i=1}^{m} P(e = \alpha | c_i, \langle f_i(\alpha_i, c_i) \rangle). \tag{3.3}$$

Note that the notation $\langle f_i(\alpha_i, c_i) \rangle$ represents the numerical values of each distribution. The correctness of this formula can be shown by symbolic evaluation of this formula according to the semantics of local expression languages defined in the previous section.

Now, we have shown that local expression languages are capable of expressing any ICI model. Because such representations contain additions, we call them *additive factorizations* of ICI. The above formula can be seen as a generalization of additive factorization of the noisy-or models of [D'Ambrosio, 1995].

Additive factorization, however, has a difficult problem; the optimal factoring problem and associated inference algorithms are only defined on products of probability distributions, so they are not readily applicable to additive expressions. In order to handle additive expressions efficiently, the inference algorithm must be extended so as to find the best sequence of application of the distributivity, associativity, and commutativity axioms, interspersed with numeric combination operators. This task has been found to be extremely difficult [Borujerdi *et al.*, 1998]. SPI [D'Ambrosio, 1995] extended the inference algorithm so that it can handle local expressions, but it tends to combine expressions too early, rather than to wait for more applications

of the distributivity axiom. As a result, it has to carry an unnecessarily large intermediate distribution, which is often too large to fit in memory.

3.4 Multiplicative Factorization of ICI

Next we develop a new, multiplicatively factored, local expression language representation. Multiplicative factorization is desired because it allows for application of the optimal factoring problem and its associated algorithms, avoiding the difficult task of performing optimal application of the distributivity axiom. Our first development will be impractical, but serves as a basis for a more significant factorization presented in the following sections.

The key idea is to introduce an intermediate random variable to each product in Equation 3.3, and to eliminate additions by achieving the effects of additions through the standard marginalization of the intermediate variables.

First, we introduce intermediate variables $e_{\alpha_1\alpha_2\ldots\alpha_m}$ with domain $\langle V, I\rangle$ and parents $c_1, c_2, \ldots, c_m$ for all $\alpha_1 * \alpha_2 * \ldots * \alpha_m = \alpha$ for each value α of e, where α_i is a value of c_i. Then, define the conditional probability distribution for each intermediate variable as follows:

$$P(e_{\alpha_1\alpha_2\ldots\alpha_m}|c_1, c_2, \ldots, c_m) = \prod_{i=1}^{m} P(e_{\alpha_1\alpha_2\ldots\alpha_m}|c_i, \langle f_i'(e_{\alpha_1\alpha_2\ldots\alpha_m}, \alpha_i, c_i)\rangle), \tag{3.4}$$

where f_i' is a density function needed here to represent actual numerical values. It returns 1 if the value of the intermediate variable is I, returns f_i if it is V:

$$f_i'(e_{\alpha_1\alpha_2\ldots\alpha_m}, \alpha_i, c_i) = \begin{cases} 1 & \text{if } e_{\alpha_1\alpha_2\ldots\alpha_m} = I. \\ f_i(\alpha_i, c_i) & \text{if } e_{\alpha_1\alpha_2\ldots\alpha_m} = V. \end{cases} \tag{3.5}$$

All intermediate variables created for the variable e become the new parents of it. Let S be the set of the new parents. Then, the conditional probability distribution

for e is defined as follows:

$$P(e=\alpha|S)=\begin{cases}1 & \text{if } \alpha_1 * \alpha_2 * \ldots * \alpha_m = \alpha \\ & \text{and } e_{\alpha_1\alpha_2\ldots\alpha_m}(\in S) = V \\ & \text{and all other parents } (S \setminus \{e_{\alpha_1\alpha_2\ldots\alpha_m}\}) \text{ are } I \\ 0 & \text{otherwise.}\end{cases} \tag{3.6}$$

The correctness of this representation can be shown by the following theorem.

Theorem 3.4.1 *The probability distribution defined as additive factorization of Equation 3.3 is equivalent to the probability distribution defined as the conformal products of transformed distributions defined by Equation 3.4 and Equation 3.6, after marginalizing out all intermediate variables.*

Proof: The conformal products consist of the distribution of Equation 3.6 and distributions defined by Equation 3.4 for each intermediate variables:

$$P(e|S) \times \prod_{x \in S} P(x|c_1, c_2, \ldots, c_m).$$

For each value α of e, the first term becomes 1 if $\alpha_1 * \alpha_2 * \ldots * \alpha_m = \alpha$ and $e_{\alpha_1\alpha_2\ldots\alpha_m}(\in S) = V$ and all other parents $(S \setminus \{e_{\alpha_1\alpha_2\ldots\alpha_m}\})$ are I, and it becomes 0 otherwise. Thus, the above expression can be written as

$$\sum_{\alpha} \sum_{\alpha_1 * \alpha_2 * \ldots * \alpha_m = \alpha} P(e=\alpha, e_{\alpha_1\alpha_2\ldots\alpha_m} = V|c_1, c_2, \ldots, c_m) \times$$

$$\prod_{x \in S \setminus \{e_{\alpha_1\alpha_2\ldots\alpha_m}\}} P(e=\alpha, x=I|c_1, c_2, \ldots, c_m)$$

Using Equation 3.4 and Equation 3.5, the last term can be eliminated because all distributions in that term evaluate to 1. Also, the other term can be simplified using the same equations, yielding

$$\sum_{\alpha} \sum_{\alpha_1 * \alpha_2 * \ldots * \alpha_m = \alpha} \prod_{i=1}^{m} P(e=\alpha|c_i, \langle f_i(\alpha_i, c_i)\rangle).$$

This is exactly the same distribution defined by Equation 3.3. □

3.5 Optimization of Multiplicative Factorization

We have shown that any ICI model can be represented using purely multiplicative local expressions, but the resulting representations are impractically large; the number of intermediate variables introduced is exponential in the number of causes. Fortunately, however, this representation can be optimized to yield more efficient representations. In particular, this is the case for the popular ICI models, noisy-or, noisy-and, noisy-max, and noisy-min. In the following sections, we show how these models can be efficiently represented in a multiplicative form.

3.5.1 Multiplicative Factorization of Noisy-Or

In the noisy-or models [Good, 1961, Pearl, 1988], the effect variable is Boolean and the binary operator is logical or. Let the domain of the effect variable be $\langle F, T\rangle$. We first optimize the representation for two-cause cases as an illustration.

Using Equation 3.4, we need to create four intermediate variables, each of which corresponds to a combination of values, as follows:

$$\begin{aligned}
P(e_{FF}|c_1,c_2) &= P(e_{FF}|c_1,\langle f_1'(e_{FF},F,c_1)\rangle) \times P(e_{FF}|c_2,\langle f_2'(e_{FF},F,c_2)\rangle).\\
P(e_{FT}|c_1,c_2) &= P(e_{FT}|c_1,\langle f_1'(e_{FT},F,c_1)\rangle) \times P(e_{FT}|c_2,\langle f_2'(e_{FT},T,c_2)\rangle).\\
P(e_{TF}|c_1,c_2) &= P(e_{TF}|c_1,\langle f_1'(e_{TF},T,c_1)\rangle) \times P(e_{TF}|c_2,\langle f_2'(e_{TF},F,c_2)\rangle).\\
P(e_{TT}|c_1,c_2) &= P(e_{TT}|c_1,\langle f_1'(e_{TT},T,c_1)\rangle) \times P(e_{TT}|c_2,\langle f_2'(e_{TT},T,c_2)\rangle).
\end{aligned} \tag{3.7}$$

The distribution for e is derived from Equation 3.6:

$$P(e|e_{FF},e_{FT},e_{TF},e_{TT}) =$$

$$\begin{cases}
1 & \text{if } e=F\wedge e_{FF}=V\wedge e_{FT}=I\wedge e_{TF}=I\wedge e_{TT}=I;\text{ or}\\
 & \quad e=T\wedge e_{FF}=I\wedge e_{FT}=V\wedge e_{TF}=I\wedge e_{TT}=I;\text{ or}\\
 & \quad e=T\wedge e_{FF}=I\wedge e_{FT}=I\wedge e_{TF}=V\wedge e_{TT}=I;\text{ or}\\
 & \quad e=T\wedge e_{FF}=I\wedge e_{FT}=I\wedge e_{TF}=I\wedge e_{TT}=V\\
0 & \text{otherwise.}
\end{cases}$$

In order to optimize the above representation, the following proposition is necessary.

Proposition 3.5.1 *The contributions from e_{FF},e_{FT},e_{TF},e_{TT} add up to 1, that is, $P(e_{FF} = V|c_1 = \beta_1, c_2 = \beta_2) + P(e_{FT} = V|c_1 = \beta_1, c_2 = \beta_2) + P(e_{TF} = V|c_1 = \beta_1, c_2 = \beta_2) + P(e_{TT} = V|c_1 = \beta_1, c_2 = \beta_2) = 1$ for each value β_1 of c_1 and β_2 of c_2.*

Proof: Using Equation 3.7, we can factorize each distribution using the function f_i'. The function f_i' is defined in Equation 3.5 using f which is in turn defined in Equation 3.2. We can simplify the resulting factorization by applying the distributivity axiom. Thus, we have

$$
\begin{aligned}
& P(e_{FF} = V|c_1 = \beta_1, c_2 = \beta_2) + P(e_{FT} = V|c_1 = \beta_1, c_2 = \beta_2) + \\
& P(e_{TF} = V|c_1 = \beta_1, c_2 = \beta_2) + P(e_{TT} = V|c_1 = \beta_1, c_2 = \beta_2) \\
= \; & f_1'(V, F, \beta_1) \times f_2'(V, F, \beta_2) + f_1'(V, F, \beta_1) \times f_2'(V, T, \beta_2) + \\
& f_1'(V, T, \beta_1) \times f_2'(V, F, \beta_2) + f_1'(V, T, \beta_1) \times f_2'(V, T, \beta_2) \\
= \; & f_1(F, \beta_1) \times f_2(F, \beta_2) + f_1(F, \beta_1) \times f_2(T, \beta_2) + \\
& f_1(T, \beta_1) \times f_2(F, \beta_2) + f_1(T, \beta_1) \times f_2(T, \beta_2) \\
= \; & P(\xi_1 = F|c_1 = \beta_1) \times P(\xi_2 = F|c_2 = \beta_2) + \\
& P(\xi_1 = F|c_1 = \beta_1) \times P(\xi_2 = T|c_2 = \beta_2) + \\
& P(\xi_1 = T|c_1 = \beta_1) \times P(\xi_2 = F|c_2 = \beta_2) + \\
& P(\xi_1 = T|c_1 = \beta_1) \times P(\xi_2 = T|c_2 = \beta_2) \\
= \; & (P(\xi_1 = F|c_1 = \beta_1) + P(\xi_1 = T|c_1 = \beta_1)) \times \\
& (P(\xi_2 = F|c_2 = \beta_2) + P(\xi_2 = T|c_2 = \beta_2))
\end{aligned}
$$

Each sum of the final expression adds up to 1 as stated in Requirement 3.2.1. Thus, we finally get 1 as expected. □

Because contributions from e_{FF},e_{FT},e_{TF},e_{TT} add up to 1, the contributions from e_{FT},e_{TF},e_{TT} to $e = T$ can be computed as 1 minus the contribution from e_{FF}. Thus, we can rewrite the above distribution as follows:

$$
P(e|e_{FF}) = \begin{cases} 1 & \text{if } e = F \wedge e_{FF} = V \\ 1 & \text{if } e = T \wedge e_{FF} = I \\ -1 & \text{if } e = T \wedge e_{FF} = V \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}
$$

The first line corresponds to the contribution from e_{FF} to $e = F$, the second line corresponds to 1, and the third line subtracts the same contribution from 1. This optimized representation eliminates all intermediate variables but e_{FF}.

It is easy to generalize this representation to multiple-cause cases. Even if there are m causes, only one intermediate variable, e_{Fm}, needs to be defined as follows:

$$P(e_{Fm}|c_1, c_2, \ldots, c_m) = \prod_{i=1}^{m} P(e_{Fm}|c_i, \langle f_i'(e_{Fm}, F, c_i)\rangle).$$

The distribution $P(e|e_{Fm})$ is obtained just by substituting e_{FF} by e_{Fm} of $P(e|e_{FF})$ in Equation 3.8.

In this representation, the size of the conditional distribution table of e is always 4 no matter how many causes there are.

3.5.2 Multiplicative Factorization of Noisy-Max

Noisy-max is a generalization of noisy-or. It is used extensively in the CPCS network. This section shows how it can be encoded in the multiplicative form.

In the noisy-max models [Díez, 1993], the binary operator is max. For the sake of simplicity of presentation, assume that the size of domain of the effect variable is three and the number of causes is two. (We will generalize them later.) Let the domain be $\langle L, M, H\rangle$, in which values are ordered as $L < M < H$.

The unoptimized representation requires 3^2 intermediate variables, of which one variable (e_{LL}) contributes to $e = L$, and three variables (e_{LM},e_{MM},e_{ML}) contribute to $e = M$. One can reduce intermediate variables by merging some of them. For example, we can merge e_{LM} and e_{MM} and create a new intermediate variable, say, $e_{(L+M)M}$, as follows:

$$\begin{aligned} P(e_{(L+M)M}|c_1, c_2) = \; & P(e_{(L+M)M}|c_1, \langle f_1''(e_{(L+M)M}, \{L, M\}, c_1)\rangle)\times \\ & P(e_{(L+M)M}|c_2, \langle f_2'(e_{(L+M)M}, M, c_2)\rangle), \end{aligned}$$

where f_i'' returns 1 if the value of the intermediate variable is I, and returns the sum of contributions if it is V:

$$f_i''(e', D, c_i) = 1 \text{ if } e' = I.$$

$$f_i''(e', D, c_i) = \sum_{\alpha \in D} f_i(\alpha, c_i) \text{ if } e' = V.$$

Now only two variables ($e_{(L+M)M}$ and e_{ML}) contribute to $e = M$. We can further reduce variables by noticing that we can merge the above two variables as well as the variable required for $e = L$, which is e_{LL}, to obtain $e_{(L+M)(L+M)}$, and that the contribution to $e = M$ can be computed as the difference between the contribution from $e_{(L+M)(L+M)}$ and that from e_{LL}. Figure 3.1 illustrates this calculation.

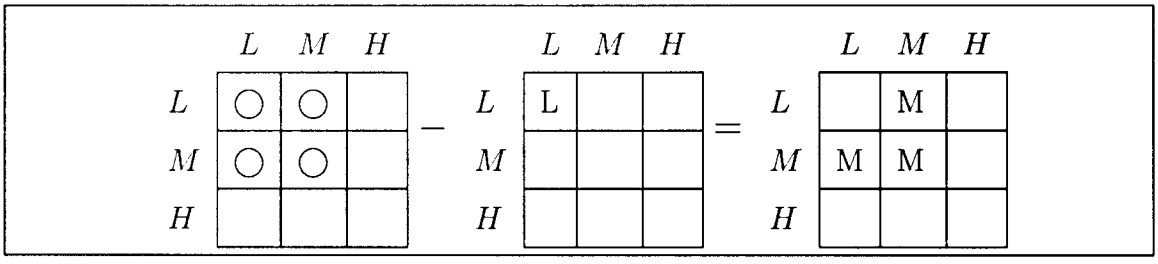


Figure 3.1. $(L + M)(L + M) - LL = LM + MM + ML$.

Similarly, the contribution to $e = H$ can be computed as the difference between the contribution from $e_{(L+M+H)(L+M+H)}$ and that from $e_{(L+M)(L+M)}$. By using the same line of reasoning as Proposition 3.5.1, the contribution from $e_{(L+M+H)(L+M+H)}$ can be shown to be equivalent to 1. Thus, we only need two intermediate variables for the noisy-max of three values, defined as follows:

$$P(e_{LL}|c_1, c_2) = \prod_{i=1}^{2} P(e_{LL}|c_i, \langle f_i'(e_{LL}, L, c_i)\rangle).$$
$$P(e_{(L+M)(L+M)}|c_1, c_2) = \prod_{i=1}^{2} P(e_{(L+M)(L+M)}|c_i, \langle f_i''(e_{(L+M)(L+M)}, \{L, M\}, c_i)\rangle). \quad (3.9)$$

The conditional distribution for e is defined as follows:

$$P(e|e_{LL}, e_{(L+M)(L+M)}) = \begin{cases} 1 & \text{if } e = L \wedge e_{LL} = V \wedge e_{(L+M)(L+M)} = I \\ 1 & \text{if } e = M \wedge e_{LL} = I \wedge e_{(L+M)(L+M)} = V \\ -1 & \text{if } e = M \wedge e_{LL} = V \wedge e_{(L+M)(L+M)} = I \\ 1 & \text{if } e = H \wedge e_{LL} = I \wedge e_{(L+M)(L+M)} = I \\ -1 & \text{if } e = H \wedge e_{LL} = I \wedge e_{(L+M)(L+M)} = V \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

The size of this table equals $3 \times 2 \times 2 = 12$.

We now develop a general noisy-max representation. Suppose there are m causes (c_1 to c_m) and n values (α_1 to α_n where $\alpha_i < \alpha_j$ if $i < j$). What we need to do is to introduce $n-1$ intermediate variables, each of which corresponds to a hypercube in Cartesian products of values. The ith variable, $e_{(\alpha_1+\alpha_2+\ldots+\alpha_i)^m}$, can be defined as follows:

$$\begin{aligned} &P(e_{(\alpha_1+\alpha_2+\ldots+\alpha_i)^m}|c_1, c_2, \ldots, c_m) = \\ &\prod_{j=1}^{m} P(\ e_{(\alpha_1+\alpha_2+\ldots+\alpha_i)^m}|c_j, \\ &\qquad \langle f''_j(e_{(\alpha_1+\alpha_2+\ldots+\alpha_i)^m}, \{\alpha_1, \alpha_2, \ldots, \alpha_i\}, c_j)\rangle). \end{aligned} \tag{3.11}$$

The conditional distribution for e is then defined using these $n-1$ intermediate variables as follows:

$$P(e|\{e_{(\alpha_1+\alpha_2+\ldots+\alpha_i)^m}|1 \le i \le n-1\}) =$$

$$\begin{cases} 1 & \text{if } e = \alpha_j \text{ and } e_{(\alpha_1+\alpha_2+\ldots+\alpha_j)^m} = V \\ & \text{and all other parents are } I \text{ for some } 1 \le j \le n-1; \\ -1 & \text{if } e = \alpha_{j+1} \text{ and } e_{(\alpha_1+\alpha_2+\ldots+\alpha_j)^m} = V \\ & \text{and all other parents are } I \text{ for some } 1 \le j \le n-1; \\ 1 & \text{if } e = \alpha_n \text{ and all parents are } I; \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

If we let $m = 2$ and $n = 2$, the above representation instantiates to that of the noisy-or defined by Equation 3.7 and 3.8. Also, if we let $m = 2$ and $n = 3$, it

instantiates to the representation of the simple noisy-max defined by Equation 3.9 and 3.10.

This representation of general noisy-max requires $n-1$ intermediate variables, hence the size of conditional distribution table for e is $n2^{n-1}$.

3.5.3 Multiplicative Factorization of Contract Renewal Model

In this section, we consider a contract renewal example taken from [Zhang and Poole, 1996]. This example is interesting because it is not an instance of any known causal independence models [Zhang and Poole, 1996], and is more complex than the noisy-or and the noisy-max models we studied above.

Suppose that faculty members at a university are evaluated in teaching, research, and service for the purpose of contract renewal. A faculty member's contract is not renewed, renewed without pay raise, renewed with a pay raise, or renewed with double pay raise depending on whether his performance is evaluated to be unacceptable in at least one of the three areas, acceptable in all areas, excellent in one area, or excellent in at least two areas, respectively.

Let c_1, c_2, and c_3 be the fractions of time a faculty member spends on teaching, research, and service respectively. Let ξ_i represent the evaluation he gets in the ith area. It can take values 0, 1, and 2 depending on whether the evaluation is unacceptable, acceptable, or excellent. The variable ξ_i depends probabilistically on c_i. It is reasonable to assume that ξ_i is conditionally independent of other c_j's and other ξ_j's given c_i.

Let e represent the contract renewal result. The variable can take values 0, 1, 2, and 3 depending on whether the contract is not renewed, renewed with no pay raise, renewed with a pay raise, or renewed with double pay raise. Then $e = \xi_1 * \xi_2 * \xi_3$, where the operator $*$ is given in the following table:

	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	3
3	0	3	3	3

Now we develop a multiplicative representation for this example. First we need to introduce intermediate variables that represent some rectangular subspaces and are combined to define areas for each value of e. Looking at the three-dimensional table, shown in Figure 3.2, may help decide what intermediate variables to define.

	0				1				2				3			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	2	3	0	2	3	3	0	3	3	3
2	0	0	0	0	0	2	3	3	0	3	3	3	0	3	3	3
3	0	0	0	0	0	3	3	3	0	3	3	3	0	3	3	3

Figure 3.2. Three dimensional table representing $\xi_1 * \xi_2 * \xi_3$.

There is only a single 1 in the table, at $\xi_1 = 1$, $\xi_2 = 1$, and $\xi_3 = 1$, so one intermediate variable, e_{111}, needs to be defined for it. There are three 2's in the table, but they do not seem to be combined to form a rectangle, so we introduce three intermediate variables, namely, e_{112}, e_{121}, and e_{211}, for those individual 2's. The area of 3's could be computed from a cube $e_{(1+2+3)^3}$ by subtracting the area of 1's and 2's from it. Finally, the area of 0's is computed by subtracting the above cube from the whole space. Thus, we introduce 5 intermediate variables whose distributions

are defined below:

$$
\begin{aligned}
P(e_{111}|c_1,c_2,c_3) &= \prod_{i=1}^{3} P(e_{111}|c_i,\langle f_i'(e_{111},1,c_i)\rangle).\\
P(e_{112}|c_1,c_2,c_3) &= P(e_{112}|c_1,\langle f_1'(e_{112},1,c_1)\rangle)\times\\
&\quad P(e_{112}|c_2,\langle f_2'(e_{112},1,c_2)\rangle)\times\\
&\quad P(e_{112}|c_3,\langle f_3'(e_{112},2,c_3)\rangle).\\
P(e_{121}|c_1,c_2,c_3) &= P(e_{121}|c_1,\langle f_1'(e_{121},1,c_1)\rangle)\times\\
&\quad P(e_{121}|c_2,\langle f_2'(e_{121},2,c_2)\rangle)\times\\
&\quad P(e_{121}|c_3,\langle f_3'(e_{121},1,c_3)\rangle).\\
P(e_{211}|c_1,c_2,c_3) &= P(e_{211}|c_1,\langle f_1'(e_{211},2,c_1)\rangle)\times\\
&\quad P(e_{211}|c_2,\langle f_2'(e_{211},1,c_2)\rangle)\times\\
&\quad P(e_{211}|c_3,\langle f_3'(e_{211},1,c_3)\rangle).\\
P(e_{(1+2+3)^3}|c_1,c_2,c_3) &=\\
&\quad \prod_{i=1}^{3} P(e_{(1+2+3)^3}|c_i,\langle f_i''(e_{(1+2+3)^3},\{1,2,3\},c_i)\rangle).
\end{aligned}
\tag{3.13}
$$

Using these intermediate variables, the conditional distribution for e is defined as follows:

$$
P(e|e_{111},e_{112},e_{121},e_{211},e_{(1+2+3)^3}) =
\begin{cases}
1 & \text{if } e=0 \text{ and all parents are } I\\
-1 & \text{if } e=0 \wedge e_{(1+2+3)^3}=V \text{ and all the other parents are } I\\
1 & \text{if } e=1 \wedge e_{111}=V \text{ and all the other parents are } I\\
1 & \text{if } e=2 \wedge e_{112}=V \text{ and all the other parents are } I\\
1 & \text{if } e=2 \wedge e_{121}=V \text{ and all the other parents are } I\\
1 & \text{if } e=2 \wedge e_{211}=V \text{ and all the other parents are } I\\
1 & \text{if } e=3 \wedge e_{(1+2+3)^3}=V \text{ and all the other parents are } I\\
-1 & \text{if } e=3 \wedge e_{111}=V \text{ and all the other parents are } I\\
-1 & \text{if } e=3 \wedge e_{112}=V \text{ and all the other parents are } I\\
-1 & \text{if } e=3 \wedge e_{121}=V \text{ and all the other parents are } I\\
-1 & \text{if } e=3 \wedge e_{211}=V \text{ and all the other parents are } I\\
0 & \text{otherwise.}
\end{cases}
\tag{3.14}
$$

The size of this table equals $4 \times 2^5 = 128$.

Suppose we want to adapt the above model to a company that makes the renewal decision in the same way, but has more evaluation areas, say m areas. We must generalize the above representation, using the same domain and the same operator, but with m causes instead of 3.

Let's imagine we have an m dimensional table this time and create intermediate variables in the same way as above. There is still only a single 1 in the table, so one intermediate variable, e_{1^m}, needs to be defined for it. There are now m 2's in the table, and they do not seem to be combined to form a rectangle, so we introduce m intermediate variables for those individual 2's. The area of 3's could be computed from a hypercube $e_{(1+2+3)^m}$ by subtracting the area of 1's and 2's from it. Finally, the area of 0's is computed by subtracting the above hypercube from the whole space. Thus, we introduce $m+2$ intermediate variables whose distributions could be defined similarly.

The conditional distribution for e is also similar to the above. Its size now becomes $4 \times 2^{m+2} = 2^{m+4}$ because e has 4 values and $m+2$ parents whose domain $\langle V, I \rangle$ is of size 2. If we did not exploit ICI, the size of conditional probability distribution for e would be $4^{m+1} = 2^{2m+2}$. Thus, the reduction we achieved is

$$\frac{2^{2m+2}}{2^{m+4}} = 2^{m-2},$$

which is exponential in the number of evaluation areas.

3.6 Summary and Discussion of Multiplicative Representations

The summary of representations developed so far is shown in Table 3.1. It shows, for each representation, the number of causes (#causes), the number of values (#values), the number of intermediate variables introduced (#i-vars), the size of conditional distribution table for the effect variable (table-size), and the size of conditional distribution table that would be needed when independence of causal influence was not exploited (orig-size).

	#causes	#values	#i-vars	table-size	orig-size
unoptimized	m	n	n^m	$n2^{n^m}$	n^{m+1}
2-cause noisy-or	2	2	1	4	8
general noisy-or	m	2	1	4	2^{m+1}
3-value noisy-max	2	3	2	12	27
general noisy-max	m	n	$n-1$	$n2^{n-1}$	n^{m+1}
3-area renewal	3	4	5	128	256
general renewal	m	4	$m+2$	2^{m+4}	2^{2m+2}

Table 3.1. Summary of multiplicative representations.

The difference between the sixth column (orig-size) and the fifth column (table-size) shows the gain we can obtain by exploiting ICI. In the case of the general noisy-or, the gain is exponential in the number of causes. In the case of the general noisy-max, whether or not there is gain depends upon m and n. However, the table size is always smaller than the original size when $m \geq 2$ and $n \geq 2$, which we suppose is always the case. If we consider n as a small constant, the gain is also exponential in the number of causes in the case of the general noisy-max just like the case of the general noisy-or.

As shown above, the table size can be quite large when there are many values. Fortunately, there are ways to handle those large tables as shown below.

First, because there are many zero's in the table, a sparse representation of distribution table could be developed. Because there are only $2(n-1)+1$ non-zero entries in the table in the case of the noisy-or and noisy-max models, the savings obtained by eliminating zero's could be exponential in the number of values.

Second, we could encode the table in local expression language using addition

and subtraction to eliminate zero's in the table. This representation is different from that of additive factorization, shown in Section 3.3. In this case, the factoring decision can be done as if the local expression is just a big distribution table without any loss of benefits of ICI. In the case of additive factorization, nothing is gained when the local expression is considered to be a big distribution, thus requiring difficult search of the optimal application of the distributivity axiom.

Third, we could use the technique of parent divorcing [Olesen *et al.*, 1989] to reduce the table size.[5] For example, suppose there are seven values. Then, six intermediate variables are required, say e_1 to e_6, and the table size is $7 \times 2^6 = 448$. We could create two new variables (say e' and e''), divide six parents into two, and assign each set to those newly created variables. Thus, we now have three conditional distribution tables, $P(e|e', e'')$, $P(e'|e_1, e_2, e_3)$, and $P(e''|e_4, e_5, e_6)$. Both e' and e'' need to have a domain of size 4, say $\langle V_1, V_2, V_3, I \rangle$, and their distribution tables should encode a deterministic function that relays the value from each parent to each output, such as e_1's V to V_1, e_2's V to V_2, e_3's V to V_3, so that those values reach e. The table size for e' is $4 \times 2^3 = 32$, that for e'' is the same, and that for e is now reduced to $7 \times 4 \times 4 = 112$. The total size is $32 + 32 + 112 = 176$ which is considerably less than the original table size of 448.

Computation using multiplicative factorization of noisy-or is easily shown to be worst-case exponential in the number of positive findings and linear in the number of negative findings and the number of parents. Similarly, computation using multiplicative factorization of noisy-max can be shown to be worst-case exponential in the number of non-minimum findings and linear in the number of minimum findings and the number of parents.

The complexity of knowledge acquisition using ICI can be measured by the number of conditional probabilities $P(\xi_i|c_i)$, which is linear in the number of causes (m).

---

[5]Suggested by Anders L. Madsen.

3.7 The Limitations of Multiplicative Factorization

We have shown that noisy-or and noisy-max can be represented efficiently using multiplicative factorization. An important question now is what other ICI models can be efficiently represented. Unfortunately, there is not much as shown below.

Here, we consider only a simple case. Suppose there are three values, $\langle 0, 1, 2 \rangle$, and two causes. In this case, the original distribution size is $3^{2+1} = 27$. The table size for multiplicative factorization is 3×2^p where p is the number of intermediate variables introduced. It is 6 for $p = 1$, 12 for $p = 2$, 24 for $p = 3$, and 48 for $p = 4$, thus we must not introduce more than three intermediate variables in order to reduce the table size from the original. If the number of intermediate variables is less than three, we could say the reduction is effective.

Each intermediate variable represents the contribution from a hyper-rectangular subspace of the Cartesian product of m domains. A binary operator divides the whole space into subspaces of equivalent classes, each of which must be represented by a linear combination of subspaces defined by intermediate variables plus the whole space defined by one (the case when all parents are I). If a binary operator can be represented using less than three intermediate variables, we can say that it is efficiently represented by multiplicative factorization at least for this simple case (two causes and three values).

In this case, there are 3^9 operators, of which 3^6 operators are commutative. Among those commutative operators, 63 operators are distributive. We have checked all 63 operators to see how many intermediate variables are necessary to represent them. Table 3.2 shows the results. The first column represents the number of intermediate variables, and the second column the number of operators that can be represented using the given number of intermediate variables.

All operators can be represented using 6 intermediate variables or less, which is less than the worst case of $3^2 = 9$ variables. Of all 63 operators, $3 + 15 + 21 = 39$ operators (62%) can be efficiently represented, and $2 + 0 + 3 = 5$ (8%) operators require larger tables than the original (non-ICI) ones.

p (#i-vars)	#operators	1 output	2 outputs	3 outputs
0	3	3		
1	15		15	
2	21		15	6
3	19			19
4	2			2
5	0			
6	3			3
total	63	3	30	30

Table 3.2. The number of operators classified by the number of intermediate variables required.

$f(x, y) = 1$

	0	1	2
0	1	1	1
1	1	1	1
2	1	1	1

(a)

if $xy = 0$ then 0 else 2

	0	1	2
0	0	0	0
1	0	2	2
2	0	2	2

(b)

$\max(x, y)$

	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

(c)

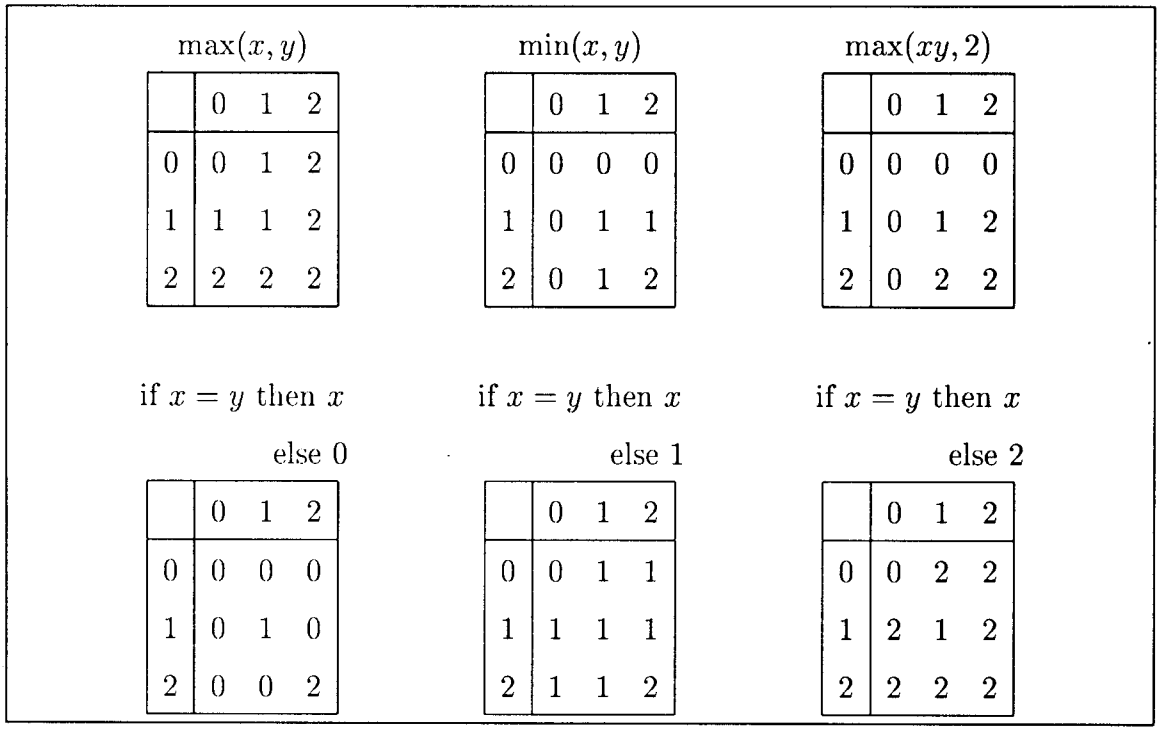
Figure 3.3. Three example operators.

Table 3.2 also shows the number of operators classified by the number of different outputs of the operators. For example, the operator $f(x, y) = 1$, which can be defined by a table in Figure 3.3 (a), has only one different output, which is 1, and can be represented using no intermediate variable, the operator of Figure 3.3 (b) has two different outputs and can be defined using one intermediate variable ($e_{(1+2)(1+2)}$), and the operator max defined by Figure 3.3 (c) has three different outputs and requires two intermediate variables, as shown in Section 3.5.1. The third, fourth, and fifth columns in Table 3.2 represent the number of operators with one, two, and three different outputs, respectively.

As shown in Table 3.2, all trivial operators (those who have only one or two different outputs) can be represented effectively, but only 6 out of 30 non-trivial operators can be efficiently represented using multiplicative factorization. Figure 3.4 shows all those six operators, which include max and min.

So far, we only considered a simple case with three values and two causes. It is difficult to know what will happen if we increase the number of values and/or causes, so it remains as an important future work. We conjecture that for three-value cases at most the above six operators are the only ones that can be defined effectively whatever the number of causes are, because the general cases of m causes are more complex than the above simple case and they contain the simple case as a special case.

For many-value cases, we anticipate that there are more operators that can be effectively represented in multiplicative factorization. The reason for this is that the number of values is the base of exponential with original tables, whereas it is just a linear factor with tables of multiplicative representations. For example, in the case of the contract renewal example in Section 3.5.3, both the original table and the table of multiplicative factorization are of a size exponential in a number that is linear in the number of causes. However, the original table is bigger exponentially due to the difference in the bases of the exponentials.



	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

	0	1	2
0	0	0	0
1	0	1	2
2	0	2	2

	0	1	2
0	0	0	0
1	0	1	0
2	0	0	2

	0	1	2
0	0	1	1
1	1	1	1
2	1	1	2

	0	1	2
0	0	2	2
1	2	1	2
2	2	2	2

Figure 3.4. Six non-trivial operators that can be efficiently represented.

3.8 Experimental Comparison of Additive and Multiplicative Representations

In this section, we experimentally compare two representations of ICI models, namely additive and multiplicative representations. We apply them to two large networks, that is, QMR-DT and CPCS, and use SPI's factoring algorithm to make actual queries to see how the multiplicative representations can improve the factoring of those queries.

To be fair, we specialized both additive and multiplicative representations to noisy-or and noisy-max. For the additive representation of noisy-or, we used the representation described in [D'Ambrosio, 1995] that can be considered as a specialization (optimization) of the general additive representation of ICI models presented in Section 3.3. For the additive representation of noisy-max, we generalized the representation of [D'Ambrosio, 1995], so that the resulting representation is also more efficient than the general additive representation.

3.8.1 Experiments using Noisy-Or models

We used the QMR-DT BN2O network for experiments on the noisy-or models. A BN2O network [Henrion and Druzdel, 1990] is a two-level network in which parent (disease) interactions at a child (symptom) are modeled using the noisy-or interaction model. The QMR-DT network [D'Ambrosio, 1994] is a very large network, with over 600 diseases, 4000 findings, and 40,000 disease-findings links. Some findings have as many as 150 parents, and a case can have as many as 50 positive findings. We used a set of Scientific American cases supplied by the Institute for Decision Systems Research.

The characteristics and results of 16 test cases are shown in Table 3.3. The first column represents the case number. There are five columns for each representation. The first and second of the five columns represent the number of variables and expressions (distributions or SPI local expressions) relevant to the query, respectively. The difference in the number of variables between the additive and multiplicative

	additive rep.					multiplicative rep.				
case	vars	exps	mults	dim	time	vars	exps	mults	dim	time
0	599	662	1.7e164	537	99	619	3088	1.5e8	20	139
1	580	1239	1.6e153	505	149	595	2511	1.0e6	14	338
2	609	2932	1.4e168	558	955	623	4655	2.5e6	14	1790
3	525	714	2.9e154	505	64	534	1845	9.0e4	10	107
4	602	2658	6.9e164	547	769	610	4032	1.5e78	258	1513
5	623	1330	1.5e171	560	214	639	3447	2.0e7	17	536
6	623	1745	1.1e164	537	386	642	4981	1.0e8	20	1220
7	589	967	5.6e160	533	172	598	2608	2.4e5	10	258
8	610	1240	2.4e153	507	165	626	3162	1.5e7	17	457
9	588	778	1.2e172	571	75	596	2498	1.3e5	8	162
10	576	777	1.4e155	512	79	590	2649	9.0e6	15	172
11	601	1690	1.4e154	507	283	611	3940	7.4e5	11	883
12	594	1747	2.8e153	506	321	604	3287	2.1e5	11	692
13	566	1103	3.4e154	505	130	573	2275	3.7e4	8	271
14	627	1534	1.3e179	595	316	653	5667	9.0e9	26	1229
15	599	1578	3.0e154	505	254	609	2656	5.2e4	10	499

Table 3.3. The characteristics and results of QMR-DT BN2O test cases.

representations comes from the introduction of intermediate variables in the multiplicative representation. The third column shows the number of multiplications. The notation "*nem*" represents $n \times 10^m$. The maximum dimensionality (the maximum number of variables involved in any conformal products) is shown in the fourth column. The maximum (intermediate) table size required for the query is exponential in the maximum dimensionality. Finally, the last column shows the CPU time in seconds measured on Pentium II 300MHz with 128MB of memory. This time only includes the time needed for factoring, excluding the time needed for the actual numeric computation of conformal products.

Apparently, in every test cases, the multiplicative factorization performed much better than the additive factorization. Although SPI's set-factoring heuristics worked well with the multiplicative factoring in most of the cases, there were cases in which SPI performed badly. Case 4 (1.5×10^{78} multiplications) and case 14 (9 billion multiplications) were particularly bad.

3.8.2 Experiments using Noisy-Max models

We used the CPCS network for experiments on the noisy-max models. The CPCS network [Pradhan *et al.*, 1994] is a multi-level, multi-valued network and is one of the largest network in use to date. It contains 422 nodes and 867 arcs. Most of the distributions are specified in the noisy-max interaction models. Some noisy-max nodes have as many as 17 parents and some nodes contain as many as 5 values. We made a marginal query for each individual variable. Table 3.4 shows some representative query results with the total figures.

Generally, the multiplicative factorization performed much better than the additive factorization. However, in some cases, SPI failed to find a good factoring with the multiplicative representation. The marginal query for "temperature" was particularly bad; it required 12 trillion multiplications for the multiplicative factorization compared to 1.6 trillion for the additive factorization.

	additive rep.					multiplicative rep.				
name	vars	exps	mults	dim	time	vars	exps	mults	dim	time
abdominal pain	77	77	2.9e12	28	0.31	128	394	1.9e8	20	2.1
appetite	79	79	1.1e16	29	0.43	129	387	2.0e8	20	2.2
diarrhea	62	62	2.3e7	16	0.65	96	301	4.4e7	15	1.5
temperature	87	87	1.6e12	29	0.57	147	516	1.2e13	29	3.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
total	422	422	1.1e16	2786	47	12K	37K	1.2e13	2572	135

Table 3.4. The characteristics and results of the CPCS marginal queries.

3.9 Summary

In this chapter, we have studied the representations of independence of causal influence (ICI) models. The important points are as follows:

- Any ICI model can be represented in the SPI local expression language using additions. Unfortunately, additions are difficult to handle.
- We have developed a purely multiplicative representation of ICI models using the SPI local expression language, and shown the correctness of the representation.
- The general multiplicative representation is impractically large. Fortunately, this representation can be specialized to yield a more compact representation. We have presented examples of efficient, specialized representations of ICI models, including noisy-or and noisy-max.
- We have shown that not all ICI models can be optimized effectively using the

purely multiplicative representation.

- We have experimentally compared the additive and the multiplicative representations of ICI models using the SPI factoring algorithm, and showed that the multiplicative representation performed much better. In some cases, the SPI factoring algorithm performed badly even with the multiplicative representation.

Chapter 4
Factoring Algorithms

The multiplicative factorization of Chapter 3 worked much better than the additive factorization with SPI factoring heuristics. However, there were cases when SPI could not exploit the benefits of the multiplicative factorization fully. In this chapter, we study factoring algorithms and prepare the framework for more efficient factoring algorithms developed in the next chapter.

4.1 Algorithm Template

This section presents a template that will be used as a basis for developing factoring algorithms later.

In our pseudocode, the following conventions are used. Indentation indicates block structure. The looping constructs **while** and the conditional constructs **if**, **then**, and **else** have the same interpretation as in Pascal. Variables (such as t and *terms*) are shown in italic and are local to the given procedure, unless otherwise noted. Functions (such as PROD and marginalize) are shown in sans serif.

Figure 4.1 shows the algorithm template. It returns a probability distribution resulting from the conformal product of a given set of probability distributions (*terms*), marginalizing out all variables not in the target (query) variables (*target*). The resulting distribution is a marginal joint probability distribution of a set of random variables that is the intersection of the target variables and all variables found in the given distributions.

Given a Bayesian network, suppose we want to compute $P(x, y)$. Then, what we need to do is to collect all probability distributions relevant to x and y, and give

```
function PROD(terms, target, choose, combine) returns a final distribution
    if terms = {t} then return marginalize(t, target)
    if terms = {t1, t2} then return combinePair({t1, t2}, target)
    while |terms| > 1
        chosen ← choose(terms, target)
        newTarget ← allVariables(terms\chosen) ∪ target
        newTerm ← combine(chosen, newTarget)
        terms ← terms \ chosen ∪ {newTerm}
    return t where terms = {t}
```

Figure 4.1. Factoring algorithm template.

that set to the above function with $\{x, y\}$ as *target*. A well-known technique, called *d-separation* [Geiger *et al.*, 1989], can be used to find variables (and their associated distributions) relevant to a query. If the multiplicative factorization of Chapter 3 is used, all products of the multiplicative representations should be expanded so that individual distributions are given to *terms*.

Let's look at the algorithm template more closely. The function PROD trivially computes the result when the number of input distributions is either one or two. When the number of given distributions is one, the algorithm uses a function called marginalize to marginalize out all variables that are not in *target*. Otherwise, it calls a function called combinePair, which is defined in Figure 4.2. Usually, combinePair is done in one operation, that is, the marginalization is done in part of the conformal product of a pair of distributions. If the number of given distributions is more than two, the algorithm enters a loop. In each step of the loop, it chooses a set of distributions and combines them, decreasing the number of distributions. If the number of distributions reaches one, the algorithm stops.

It is the function *choose* and *combine* that change the behavior of this template.

```
function combinePair(terms, target) returns a final distribution
    return marginalize(t1 × t2, target)
    where {t1, t2} = terms
```

Figure 4.2. The function combinePair.

As an illustration, Figure 4.3 shows a simple instantiation of this template, that is, a simple random factoring algorithm, called RAND. In this instantiation, *choose* (chooseRandomly) chooses a pair of distributions randomly, and *combine* computes their conformal product by calling combinePair.

```
function chooseRandomly(terms, target) returns a pair of distribution
    return a pair chosen randomly from terms
function RAND(terms, target) returns a final distribution
    return PROD(terms, target, chooseRandomly, combinePair)
```

Figure 4.3. Random factoring algorithm.

4.2 Optimal Factoring Algorithm

We can construct an optimal factoring algorithm as an instantiation of the template. Figure 4.4 shows that algorithm.

The function OPTChoose returns a pair that minimizes the total cost, which is defined as the sum of immediate cost of combining the pair and the future cost of combining all the rest as well as the immediate result. The future cost is found by recursively calling the function OPT with a smaller set of distributions.

function OPTChoose(*terms*, *target*) **returns** a pair of distribution

return *pair* $\subset$ *terms* that minimizes the total cost.

where

- the total cost is the sum of the immediate cost and the future cost;
- the immediate cost is the cost of computing *result* = combinePair(*pair*, *newTarget*);
- *newTarget* = allVariables(*terms**pair*) $\cup$ *target*; and
- the future cost is the cost of computing OPT(*terms* $\setminus$ *pair* $\cup$ {*result*}, *target*)

function OPT(*terms*, *target*) **returns** a final distribution

return PROD(*terms*, *target*, OPTChoose, combinePair)

Figure 4.4. Optimal factoring algorithm.

The function OPT has an obvious inefficiency, that is, when OPTChoose returns a pair, it has already computed the optimal factoring in order to find that pair, so it is redundant to call PROD with that pair. This redundancy can be easily avoided by modifying the algorithm, but even with that modification OPT is still impractical to use because the number of pairs to consider is huge, that is,

$$\prod_{i=2}^{n} \frac{n(n-1)}{2},$$

where n is the number of distributions.

Li and D'Ambrosio [1994] developed a more efficient optimal factoring algorithm using dynamic programming techniques. Even with dynamic programming, however, determining the exact optimal factoring is impractical.

Because seeking exact optimal factoring is impractical, we need to pursue an approximate solution. One way to make the above algorithm practical is to make it to compute the future cost using some kind of function approximator that estimates the

cost instead of computing the exact cost by calling OPT recursively. Developing such function approximator using machine learning techniques is the topic of Chapter 5.

4.3 SPI Factoring Algorithm

SPI tries to find an optimal factoring by employing *set-factoring heuristics* [Li and D'Ambrosio, 1994]. Set-factoring heuristics assign a value to each pair (later generalized to a set [D'Ambrosio, 1995]) of probability distributions. That value represents some goodness of the combination of those distributions. In each step, a combination that has the maximum heuristic value is chosen and combined. Figure 4.5 shows an instantiation of the template to SPI's factoring algorithm. SPI's factoring algorithm is greedy in nature.

```
function SPIChoose(terms, target) returns a pair of distribution
    return a pair chosen from terms that maximizes the heuristic value.
function SPI(terms, target) returns a final distribution
    return PROD(terms, target, SPIChoose, combinePair)
```

Figure 4.5. SPI factoring algorithm.

4.4 Variable Elimination

So far, all algorithms choose a pair and combine them at each step. We call these algorithms *pair-combination algorithms.* The optimal factoring can be specified by the ordering of pair combinations. In other words, the optimal factoring can be specified by a binary tree, where the leaves are the given distributions, the intermediate nodes represent the result of pair-wise combination of their two children, and the root is

the final result.

Among Bayesian inference systems, systems using pair-combination algorithms are exceptional — SPI is such a rare system. Most other systems, such as VE [Zhang and Poole, 1996] and ci-elim-bel [Rish and Dechter, 1998], use so called *variable-elimination algorithms*, which eliminate a non-target (non-query) variable at each step, by combining all the probability distributions that contain that variable.

Typically, variable-elimination algorithms define the cost either as the maximum dimensionality or as the maximum size of distributions created during the computation, and try to minimize such costs.

Apparently, if we define the cost as the number of multiplications, elimination ordering alone is not sufficient to specify an optimal sequence of computation. Multiple distributions must be combined to eliminate a variable and the cost depends upon how these distributions are combined, which is not specified by the variable-elimination ordering.

Suppose we combine a variable-elimination algorithm with an optimal combination algorithm, say OPT, within variable elimination. Then, the question is whether the optimal variable-elimination ordering, combined with such an optimal combination algorithm, is optimal. Unfortunately, it is not the case, as shown by the following theorem.

Theorem 4.4.1 *Variable-elimination algorithms are suboptimal in terms of the number of multiplications.*

Proof: This theorem is actually a corollary of a theorem (Lemma 2 in [Li and D'Ambrosio, 1994]) proven by Zhaoyu Li. Consider a net consisting of node H with parents A, B, C, and D, and a marginal query for H. Li showed that the optimal sequence is to combine $P(A)$ with $P(B)$, and $P(C)$ with $P(D)$, and then to combine either with $P(H|A,B,C,D)$. However, this sequence can not be generated from the variable-elimination perspective, since no matter which variable we choose to eliminate first, the algorithm would immediately combine the marginal distribution for that node with the conditional distribution for H. Thus, it fails to find the

optimal factoring. □

Furthermore, the optimal variable-elimination ordering in terms of the number of variables involved in a product does not necessarily lead to the optimal number of multiplications; it is often the case that the less optimal variable-elimination ordering requires fewer multiplications than the optimal variable-elimination ordering. This happens, for example, when the domain sizes of variables differ.

4.5 Greedy Pair-Combination Algorithms

A non-target variable can be eliminated by multiplying all distributions that contain that variable and marginalizing it out. The cost of this operation depends on how these distributions are combined.

We have found that simple greedy pair-combination algorithms work well for this purpose. Figure 4.6 and Figure 4.7 show such algorithms.

```
function GCMChoose(terms, target) returns a pair of distribution
    return pair ⊂ terms such that the number of multiplications needed for
        combinePair(pair, target) is minimal.
function GCM(terms, target) returns a final distribution
    return PROD(terms, target, GCMChoose, combinePair)
```

Figure 4.6. Greedy-combination algorithm based on the number of multiplications.

The difference between these algorithms is the cost they try to minimize. The function GCM uses the number of multiplications as a criterion in choosing a pair, whereas the function GCD uses dimension as a criterion.

These simple greedy pair-combination algorithms could be used to combine all distributions, not just to eliminate a variable, but their performance is poor for this

```
function GCDChoose(terms, target) returns a pair of distribution
    return pair ⊂ terms such that the dimension of pair is minimal.

function GCD(terms, target) returns a final distribution
    return PROD(terms, target, GCDChoose, combinePair)
```

Figure 4.7. Greedy-combination algorithm based on dimension.

purpose. They only work well with variable elimination, where the size of distributions grows monotonically[6] until the last combination.[7] Because of this monotonicity, it is important to minimize the size or dimension of earlier distributions.

Because of this property, that is, the monotonic increase of distribution size, these algorithms may seem optimal for eliminating a variable. Unfortunately, this is not the case. It is not hard to construct a counter example, in which choosing a smaller distribution does not lead to a minimum total cost.

---

[6]If a proper subset of distributions eliminates another variable, the monotonicity does not hold. However, such cases are not common.

[7]In the last combination, the distribution shrinks due to marginalization.

Chapter 5
Reinforcement Learning Approach to Factoring

In this chapter, we use machine learning techniques to generate a good factoring heuristic automatically.

5.1 Reinforcement Learning

Reinforcement learning methods [Russell and Norvig, 1995] learn a *policy* that tells, for each state, which actions should be taken in that state. After action a is chosen and applied in state s, the problem space shifts to state s' and the learning system receives reinforcement, $R(s, a, s')$.

To view the optimal factoring problem as a reinforcement learning problem, we must describe the problem space and the reinforcement function R. The state corresponds to a set of probability distributions and a set of target (query) variables. The starting state contains the initial set of probability distributions relevant to the given query. For each state, the action is to combine a subset of probability distributions chosen from the set of distributions in that state. The cost of the action, either the number of multiplications of individual probability values or the maximum dimensionality, is given as a negative reinforcement.

The characteristics of this formulation are given as follows:

- The environment is accessible; the learning agent perceives the exact state of the environment, namely, the set of probability distributions and the target variables.

- In the case that a pair-combination algorithm is used, the precise length of agent's life ($n - 1$, where n is the number of distributions given initially) is

known in advance. Even if a variable-elimination algorithm is used, the length of the agent's life is bounded (at most $n - 1$). Therefore, the *finite-horizon model* [Kaelbling *et al.*, 1996] can be used as a model of optimality. In finite-horizon models, the optimal policy is to maximize the expected reward for some finite number of steps.

- Whatever policy is followed, every action sequence corresponds to a valid factoring. This means that useful reward information is obtained fairly frequently.
- State transitions are deterministic. As a result, exploration might be necessary to visit all states in order to find an optimal policy. Also, this could imply that the initial value function is significant.
- State space is very large. Because there is no bound for the number of distributions, the size of the state space is, in fact, infinite. This implies that the policy cannot be represented by a simple table-lookup that maps states to optimal actions. Instead, the policy should be represented by some kind of function approximator that maps an implicit representation of states to optimal actions.
- The number of actions is large. In each state with n probability distributions, there are exactly $n(n-1)$ possible pairs (and 2^n subsets) to choose from.

While developing the actual learning system, we must keep the above characteristics in mind.

5.2 Factoring Algorithms with Future Cost Estimators

The optimal factoring algorithm, OPT, computes the immediate cost of combining the chosen pair and the future cost of combining the rest in order to choose an optimal pair. However, that algorithm is not practical because computing the exact future cost is very costly. If a good approximator exists for efficiently estimating the future costs, we may be able to develop a practical factoring algorithm that works better than greedy factoring algorithms such as SPI. In the next section, we try to

develop such approximators. In this section, we present factoring algorithms that work with various future cost estimators.

5.2.1 Pair-Combination Algorithms with Estimators

Figure 5.1 and Figure 5.2 show pair-combination algorithms that work with future cost estimators. The former defines the number of multiplications as the cost and the latter defines the maximum dimensionality as the cost.

function PAIR-EST-MULChoose(*terms*, *target*) **returns** a pair of distribution

return *pair* $\subset$ *terms* that minimizes the total number of multiplications.

where

- the total cost is the sum of the immediate cost and the future cost;
- the immediate cost is the cost of computing *result* = combinePair(*pair*, *newTarget*);
- *newTarget* = allVariables(*terms**pair*) $\cup$ *target*; and
- the future cost is the estimated optimal cost of combining the rest: estimateMul($terms \setminus pair \cup \{result\}, target$)

function PAIR-EST-MUL(*terms*, *target*) **returns** a final distribution

return PROD(*terms*, *target*, PAIR-EST-MULChoose, combinePair)

Figure 5.1. Pair-combination algorithm with estimators for the number of multiplications.

The difference between these algorithms lies in what estimators are used and how the immediate cost and the estimated future cost are combined. These functions are very similar to the optimal pair-combination algorithm, OPT, except that they

function PAIR-EST-DIMChoose(*terms*, *target*) **returns** a pair of distribution

return *pair* $\subset$ *terms* that minimizes the maximum dimensionality.

where

- the maximum dimensionality is the maximum of the immediate dimension and the future dimensionality;
- the immediate dimension is the number of variables involved in computing *result* = combinePair(*pair*, *newTarget*);
- *newTarget* = allVariables(*terms**pair*) $\cup$ *target*; and
- the future dimensionality is the estimated optimal dimensionality in combining the rest: estimateDim($terms \setminus pair \cup \{result\}, target$)

function PAIR-EST-DIM(*terms*, *target*) **returns** a final distribution

return PROD(*terms*, *target*, PAIR-EST-DIMChoose, combinePair)

Figure 5.2. Pair-combination algorithm with estimators for the maximum dimensionality.

estimate the optimal future cost instead of computing the exact cost.

These algorithms work well with small to modest sized Bayesian networks, but they are too costly for large networks such as the CPCS network. This is because the number of pairs checked at each iteration is too large, i.e., $N(N-1)/2$, where N is the number of distributions, which is as large as 516 for marginal queries in the CPCS network. In addition, the number of iterations, which is the number of distributions, is also large compared to that required by variable-elimination algorithms.

5.2.2 Variable-Elimination Algorithms with Estimators

Figure 5.3 and Figure 5.4 show variable-elimination algorithms that work with future cost estimators. The former defines the number of multiplications as the cost and the latter defines the maximum dimensionality as the cost.

The difference between these algorithms lies in what estimators are used and how the immediate cost and the estimated future cost are combined.

Combination algorithms also differ depending on the cost they try to minimize. VE-EST-MUL uses GCM to minimize the number of multiplications needed to eliminate a variable, and VE-EST-DIM uses GCD to minimize the dimensionality of combination.[8] Both GCM and GCD are greedy pair-combination algorithms shown in Section 4.5.

Note that the same estimators are used both for variable-elimination algorithms and pair-combination algorithms shown in the previous subsection.

Although variable-elimination algorithms are, in general, suboptimal as shown in Section 4.4, we have to use them because pair-combination algorithms shown in the previous section cannot handle large Bayesian networks. The variable-elimination algorithms shown here can handle large networks. The number of sets checked at each iteration equals the number of non-target variables, which is at most 147 for

[8]Using GCD does not affect the maximum dimensionality, which is the same as the final dimension in this case. However, it affects intermediate dimensions that matter for overall efficiency.

function VE-EST-MULChoose(*terms*, *target*) **returns** a set of distribution

return *set* $\subset$ *terms* that eliminates some variable(s) and minimizes the total number of multiplications.

where

- the total cost is the sum of the immediate cost and the future cost;
- the immediate cost is the cost of computing $result = \mathsf{GCM}(set, newTarget)$;
- $newTarget = \mathsf{allVariables}(terms \backslash set) \cup target$; and
- the future cost is the estimated optimal cost of combining the rest: $\mathsf{estimateMul}(terms \setminus set \cup \{result\}, target)$

function VE-EST-MUL(*terms*, *target*) **returns** a final distribution

return PROD(*terms*, *target*, VE-EST-MULChoose, GCM)

Figure 5.3. Variable-elimination algorithm with estimators for the number of multiplications.

function VE-EST-DIMChoose(*terms*, *target*) **returns** a set of distribution

return *set* $\subset$ *terms* that eliminates some variable(s) and minimizes the maximum dimensionality.

where

- the maximum dimensionality is the maximum of the immediate dimension and the future dimensionality;
- the immediate dimension is the number of variables involved in computing *result* = GCD(*set*, *newTarget*);
- *newTarget* = allVariables(*terms**set*) $\cup$ *target*; and
- the future dimensionality is the estimated optimal dimensionality in combining the rest: estimateDim($terms \setminus set \cup \{result\}, target$)

function VE-EST-DIM(*terms*, *target*) **returns** a final distribution

return PROD(*terms*, *target*, VE-EST-DIMChoose, GCD)

Figure 5.4. Variable-elimination algorithm with estimators for the maximum dimensionality.

marginal queries in the CPCS network and is much smaller than $516 \times (516 - 1)/2$ of pairs required by the pair-combination algorithms.

5.3 Future Cost Estimators

In the previous section, the functions estimateMul and estimateDim are used to map a state (i.e., a set of distributions and a set of target variables) to an estimated optimal cost (the number of multiplications or the maximum dimensionality). An optimal cost is defined to be the cost that could be obtained by following an optimal policy. Thus, given a good heuristic function, the factoring algorithms in the previous sections should find a globally optimal factoring. The success of those algorithms depends mostly upon the quality of the heuristic function they use. In this section, we try to obtain good heuristic functions by using machine learning techniques.

The future cost estimators we develop consist of a state encoder, which maps a state (i.e., a set of distributions and a set of target variables) to a vector of real numbers, and a function approximator, which learns a function from those vectors to a real number that approximates the optimal cost starting from the given state. Optionally, we use a cost encoder that maps from a cost to a real that is easier to handle than handling the cost directly.

We will measure the benefit of learning by comparing the performance of estimators with that of *zero-estimator*. Zero-estimator always returns zero regardless of its inputs. Factoring algorithms combined with zero-estimator implements simple greedy heuristics. In particular, VE-EST-DIM combined with zero-estimator is very similar to popular heuristics known as the *minimum size* heuristic and the *minimum weight* heuristics [Kjærulff, 1993]. The minimum size heuristic and the minimum weight heuristic choose the next variable to be eliminated as the one which produces the smallest clique, or the clique with the least weight (the base 2 logarithm of the clique size), respectively. Rose [1973] described the minimum size heuristic under the name *minimum degree* algorithm and referred to other papers [Sato and Tinney, 1963, Tinney and Walker, 1967] in which it is assumed that this algorithm produces near

optimal elimination orderings.

5.3.1 Function Approximators

We use artificial neural networks [Haykin, 1994] as a function approximator. Neural networks are a *universal approximator* in the sense that they can approximate any continuous multivariate function to any desired degree of accuracy, provided that sufficiently many hidden neurons are available [Haykin, 1994]. In practice, however, developing a good approximator is still a challenge. It is rather an art, not science. *Ad hoc* procedures have been used for developing application-specific networks.

In the following sections, we attempt to build a good approximator using two alternative neural networks: multilayer perceptrons (feed forward multilayer neural networks) and radial-basis function networks, and compare their performances.

Multilayer Perceptrons

A multilayer perceptron has layers of neurons. Each neuron receives inputs from all neurons in the previous layer and sends its output to all neurons in the next layer. The output of a neuron (a_i), given the set of output values (a_j's) from the previous layer, is computed as follows:

$$a_i \leftarrow g(\sum_j w_{ji} a_j),$$

where w_{ji} is the weight associated with the link between a_j and a_i. Different neuron models are obtained by using different mathematical functions for g. We use a monopolar sigmoidal function,

$$g(x) = \frac{1}{1 + e^{-x}},$$

for hidden layers (layers between the input layer and the output layer), and a linear function,

$$g(x) = x,$$

for the output layer. A linear function for the output layer allows neural networks to output numbers of any range.

Training Multilayer Perceptrons Using Temporal Difference

We need to train the neural network in order to create a good heuristic function. An important constraint on such function is the following equation:

$$\mathsf{estimateMul}(s_i) = \min_a(R(a) + \mathsf{estimateMul}(s_{i+1})).$$

That is, the estimated optimal cost of the state is the minimum sum of the immediate cost of action a and the estimated optimal cost of the next state.

Each time a state transition occurs, we compute the error of the neural network's output as follows:

$$error \leftarrow \min_a(R(a) + \mathsf{estimateMul}(s_{i+1})) - \mathsf{estimateMul}(s_i). \tag{5.15}$$

Because this statement uses the difference in estimates between successive states, it is often called the *temporal-difference* equation.

As a learning rule of multilayer perceptrons, we tried both back-propagation (BP) [Haykin, 1994] and TD(λ) [Sutton, 1988] which is a generalization of BP. Both the BP and TD(λ) propagate the above temporal error through the neural network and update the weights of links between neurons. Updating the weights using this error should cause the neural network to reach the desired equilibrium constraint given above.

The difference between BP and TD(λ) is that BP assigns the error only to the current decision, whereas TD(λ) also assigns the error to the previous decisions. The amount of the error assigned to the previous decisions decreases exponentially and is controlled by the parameter λ.

Radial-Basis Function Networks

A radial-basis function network (RBFN) has only one hidden layer and units in this layer represent *radial-basis functions* described below. In the standard RBFN, the

output from the network is computed as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i g_i(\|\mathbf{x} - \mathbf{x}_i\|),$$

where $\mathbf{x} \in \mathbf{R}^p$ is an input vector of length p, $g_i(\|\mathbf{x} - \mathbf{x}_i\|)$ is a set of N arbitrary (generally nonlinear) functions, known as *radial-basis* functions, and $\|\cdot\|$ denotes a *norm* that is usually taken to be Euclidean. The known data points $\mathbf{x}_i \in \mathbf{R}^p, i = 1, 2, \ldots, N$ are taken to be the *centers* of the radial-basis functions.

In order to cover the huge state space of our future cost estimators by using a practical number of centers, we use a variant of RBFN called the normalized radial-basis function network (NRBF) [Kretchmar and Anderson, 1997] which normalizes the activations by the sum of all radial-basis function activations, and we use an inverse square function for g. Thus, in our adapted RBFN, the output from the network is computed as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \frac{1/\|\mathbf{x} - \mathbf{x}_i\|^2}{\sum_{j=1}^{N} 1/\|\mathbf{x} - \mathbf{x}_j\|^2}. \tag{5.16}$$

Training RBFN

We developed two alternative algorithms for training RBFN. Because the state space is huge, choosing good centers is crucial to the success in this problem domain. Both algorithms choose centers based on the given data points, attempting to cover the most important space.

The first algorithm, called the *contribution-based center selection*, is shown in Figure 5.5 and Figure 5.6. The procedure ContribBasedCenterSelection learns the center locations, and the function RBFN-Cont computes the estimate based on the learned centers. ContribBasedCenterSelection remembers all data points as centers until the number of centers exceeds the predefined limit N. In that case, it removes one center with the least contribution to the computation of estimates. RBFN-Cont computes the estimate just as described in Equation 5.16 while calculating and storing contributions from each center.

parameter N : the maximum size of centers.

global-variable *centers* : a set of triples $\langle center, weight, contrib \rangle$, initialized to $\emptyset$.

procedure ContribBasedCenterSelection($dataPt$, $value$)

    if $|centers| = N$ **then**

        Remove a triple with minimum *contrib* from *centers*.

    Add a triple $\langle dataPt, value, 1 \rangle$ to *centers*.

Figure 5.5. Contribution-based center selection.

function RBFN-Cont($dataPt$) **returns** an estimate

    $totalContrib \leftarrow 0$

    forall $1 \leq i \leq |centers|$

        $contrib_i \leftarrow \frac{1}{\|dataPt - \text{center}(centers_i)\|^2}$

        $totalContrib \leftarrow totalContrib + contrib_i$

    $estimate \leftarrow 0$

    forall $1 \leq i \leq |centers|$

        $normalizedContrib_i \leftarrow contrib_i / totalContrib$

        $estimate \leftarrow estimate + \text{weight}(centers_i) \times normalizedContrib_i$

        $\text{contrib}(centers_i) \leftarrow$

            $\text{contrib}(centers_i) \times 0.99 + normalizedContrib_i$

    return *estimate*

Figure 5.6. RBFN with the contribution-based center selection.

The above algorithm places centers on heavily used areas, so it might happen that only a small areas are approximated. The second algorithm below tries to place centers in a more balanced manner so as to cover larger areas.

The second training algorithm, called the *clustering-based center selection*, is shown in Figure 5.7 and Figure 5.8. The procedure ClusteringBasedCenterSelection learns the center locations, and the function RBFN-Clus computes the estimate based on the learned centers. ClusteringBasedCenterSelection classifies data points and values into N clusters using an unsupervised learning technique, and regards means of each cluster as centers. RBFN-Clus computes the estimate just as described in Equation 5.16.

parameter N : the maximum size of centers.
parameter M : the maximum size of data points.
global-variable *centers* : an array of pairs $\langle center, weight \rangle$ of length N.
global-variable *dataPoints* : an array of pairs $\langle center, weight \rangle$ of length M.
global-variable *numDataPt* : the number of points in *dataPoints*, initialized to 0.
procedure ClusteringBasedCenterSelection(*dataPt*, *value*)
    $dataPoints[numDataPt \bmod M] \leftarrow \langle dataPt, value \rangle$
    $numDataPt \leftarrow numDataPt + 1$
    Classify *dataPoints* into at most N clusters and
        store means of them into *centers*.

Figure 5.7. Clustering-based center selection.

As the unsupervised learning method, we used the maximum-neuron-based [Takefuji *et al.*, 1992] self-organization classification algorithm [Oka *et al.*, 1996]. This algorithm converges faster than the more conventional Kohonen's self-organization

function RBFN-Clus($dataPt$) **returns** an estimate

$totalContrib \leftarrow 0$

forall $1 \leq i \leq |centers|$

$contrib_i \leftarrow \frac{1}{\|dataPt - \text{center}(centers_i)\|^2}$

$totalContrib \leftarrow totalContrib + contrib_i$

$estimate \leftarrow 0$

forall $1 \leq i \leq |centers|$

$normalizedContrib_i \leftarrow contrib_i / totalContrib$

$estimate \leftarrow estimate + \text{weight}(centers_i) \times normalizedContrib_i$

return $estimate$

Figure 5.8. RBFN with the clustering-based center selection.

map [Kohonen, 1993].

There are some omissions in these algorithms. First, if a data point given to RBFN-Cont or RBFN-Clus is found among the learned centers, the function simply returns the associated weight as the estimate. Second, in the actual implementation, ContribBasedCenterSelection and ClusteringBasedCenterSelection will not remember those data points that are estimated from the current centers with 99% of accuracy. This prevents adding large number of redundant centers.

5.3.2 State Encoding

The neural network can accept only a fixed vector of values describing each state (i.e., distributions and targets). Distributions and targets, on the other hand, are variable-length. Hence, it is necessary to define a mapping from variable-length objects to a fixed vector.

We developed three kinds of mappings. The first one extracts some fixed set of features from the original inputs. The second one uses a fixed, but enormous matrix

to represent the original inputs. The last one tries to capture the most important, fixed-length subportion of the inputs.

Feature Set

A *feature* extracts an important aspect of the original input that the neural network can use to predict the value of the state. We have tested a number of features including the following:

1. The number of variables.
2. The number of non-target variables.
3. The number of expressions.
4. The worst case cost of the state.
5. The maximum depth of the Bayesian network defined only by the remaining distributions of the state.
6. The total, mean, standard deviation, minimum, and maximum of distribution sizes.
7. The total, mean, standard deviation, minimum, and maximum of distribution dimensions.
8. The total, mean, standard deviation, minimum, and maximum of the number of variables needed to eliminate each non-target variable.

Of these features, we chose (1), (2), (3), (7), and (8) for estimateDim and (1), (3), (4), (6), and (7) for estimateMul. From our limited experiments, adding more features to those estimators did not improve their performances.

Zhang and Dietterich [1995] reported that normalization of feature values worked because reinforcement learning is easier if the reinforcement function is independent of the difficulty of the problem. In order to check to see if normalization works for

our problem domain, we used normalized feature sets in addition to unnormalized ones.

In order to normalize the cost, we compute *initialMaxCost*, which is the worst case cost of the given problem. It is computed as follows:

$$initialMaxCost \leftarrow (NumDists - 1) \times \prod_{v \in Vars} \text{domainSize}(v),$$

where *NumDists* is the number of distributions and *Vars* is the set of domain variables. The normalized cost is then given by $cost/initialMaxCost$.

Matrix Representation

We developed a matrix representation of the state, i.e., distributions and targets. Each row of a matrix represents a distribution, and each column represents a variable. An entry (i, j) of a matrix is 1 if the ith distribution contains jth variable and is 0 otherwise. In order to encode target variables, we use the first row and say that an entry $(0, j)$ of a matrix is 1 if the jth variable is a target.

In the case of estimateMul, we also need to encode the domain size of each variable. We use the row number -1 for this purpose so that an entry $(-1, j)$ contains the domain size of the variable j.

For example, the state that consists of the distributions

$$\{P(A|B,C), P(B|D), P(C|D), P(D)\}$$

and the targets $\{A\}$ can be represented by the matrix below.

	A	B	C	D
target	1	0	0	0
$P(A\|B,C)$	1	1	1	0
$P(B\|D)$	0	1	0	1
$P(C\|D)$	0	0	1	1
$P(D)$	0	0	0	1

Although this representation contains all necessary information for the prediction, there is a problem if we want to use the matrix representation as the input to the neural network. That is, the size of the matrix varies depending on state.

In order to handle matrices varying in size, we allocate a huge array and use a subportion of it for the matrix representation. The array must be large enough for the problems under consideration. For the marginal queries in the CPCS network, we allocated an array for 516 distributions and 147 variables. For the Scientific American queries in the QMR-DT BN2O network, an array for 5,667 distributions and 653 variables is necessary.

Feature Set v.s. Matrix Representation

The pros and cons of using the feature set versus the matrix representation can be summarized as follows.

- The feature set summarizes the state, throwing away some information, so it is very difficult to develop a good feature set that preserves necessary information for the prediction. On the other hand, the matrix representation preserves every piece of information, so all necessary information is kept.

- The feature set extracts information that the developer thinks is useful, which might help the learner. In contrast, the learner must extract useful information by itself when given the matrix representation.

- The feature set is of fixed length and is usually small, whereas the matrix representation can be very large when one wants to handle large Bayesian networks. Large inputs make the neural network learn slowly and consume huge memory.

- Computing a feature set may be expensive depending on what features are computed, whereas the cost of computing the matrix is minimum.

- The feature set can be designed so that the learned heuristic generalizes well

to Bayesian networks other than those used for training. On the other hand, it is difficult for the neural network to learn a generalizable heuristic using a matrix because different Bayesian networks use different portions of the input matrix and because the input matrix has many forms of symmetry.

Significant Subportion Representation

As described in the previous subsection, the feature set and the matrix representation are opposite extremes. In this section, we develop a compromised representation, called the *significant subportion representation*, that is intended to combine strong points from both extremes. This representation inherits generalizability from the feature set by representing each variable by its features, and preserves some topological information by extracting the most important subportion of the matrix.

In this representation, each variable is represented by the following features called the *variable features*:

- Whether or not the variable is among the target variables;
- The number of relevant variables;
- The number of expressions that contain this variable;

where relevant variables are those variables that share an expression. In the case of estimateMul, the following two features are also considered:

- The required table size that is the product of the number of values of all the relevant variables;
- The number of values in the domain of this variable.

In order to extract most significant subportion of a Bayesian network, we define a total ordering of significantness among variable features as follows:

1. A target variable is more significant than a non-target variable;
2. If a variable has greater number of relevant variables, it is more significant;

3. The variable with larger table size is more significant;

4. The variable with more values is more significant;

5. The variable that appears in more expressions is more significant.

In order to make a fixed vector, we choose N most significant variables and for each variable we pick M most significant relevant variables, where N and M are fixed parameters. Thus we give the total of $N(M+1)$ variable features to the neural network. In the following experiments, we let $N = 20$ and $M = 4$.

5.3.3 Cost Encoding

The output from the neural network is either the maximum dimensionality (in the case of estimateDim) or the number of numerical multiplications (in the case of estimateMul). Learning the maximum dimensionality might be easier than learning the number of multiplications because the maximum dimensionality is very much related to the number of variables of each distribution, which is given as an input. The number of multiplications is generally exponential in the number of variables of each distribution, requiring the neural network to learn such an exponential function. This exponential function could be avoided if we make the neural network learn the logarithm of the number of multiplications instead of the number of multiplications. We will test both cases in the experiments below.

5.4 Experimental Results

In this section, we present experimental results obtained using the factoring algorithms developed so far based on variable elimination combined with the multiplicative factorization of ICI. Subsection 5.4.1 evaluates various estimators by training each of them using a fixed policy. This test shows the basic ability of each estimator, that is, the ability to approximate a particular cost function. Subsection 5.4.2 compares various estimators by training them on-line. Finally, we will see the generalizability of the learned heuristic function in Subsection 5.4.3 by applying the

heuristic function to other kinds of Bayesian networks. We will also compare the results with those obtained by other methods.

5.4.1 Training Estimators Using a Fixed Policy

In this section, we see if each of various estimators can approximate a particular policy. First, we created data, that is, we collected all state-reward pairs, each of which corresponded to a decision the algorithm made, by running the above factoring algorithms with zero-estimator, which is considered as near-optimal. Then, we trained each estimator providing these pairs, and measured errors.

As the error, we computed the average square of *absolute error* normalized by the error given by zero-estimator. The absolute error is the difference between an estimate and a true cost, in contrast to the time difference error defined in Equation 5.15.

Figure 5.9 and Figure 5.10 show the results of various configurations of estimateDim and estimateMul, respectively.

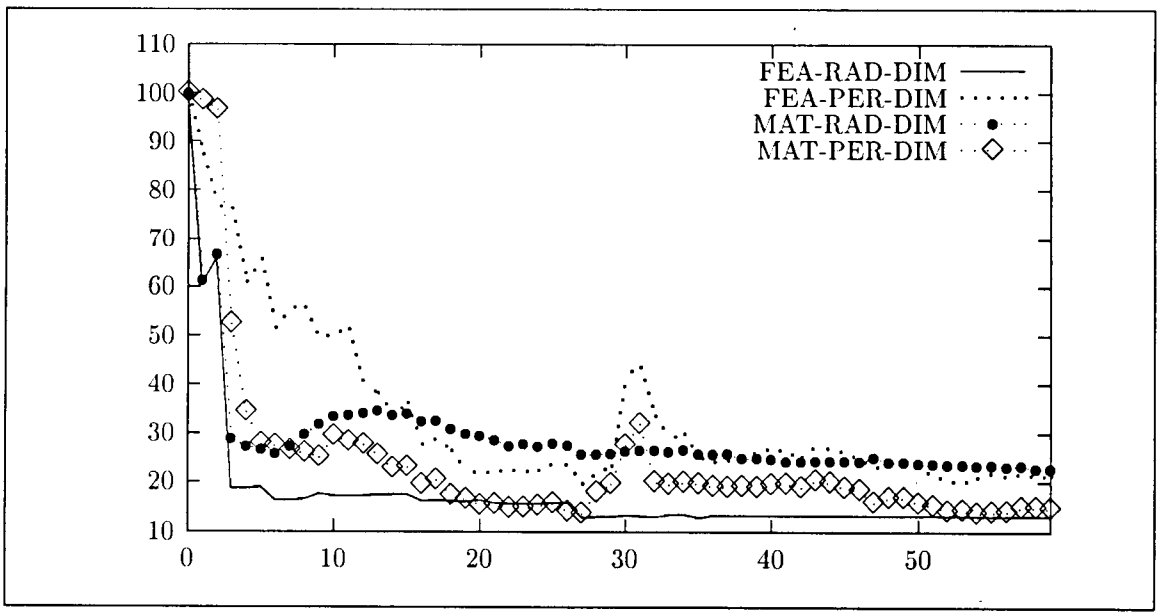


Figure 5.9. The evaluation of various configuration of estimateDim.

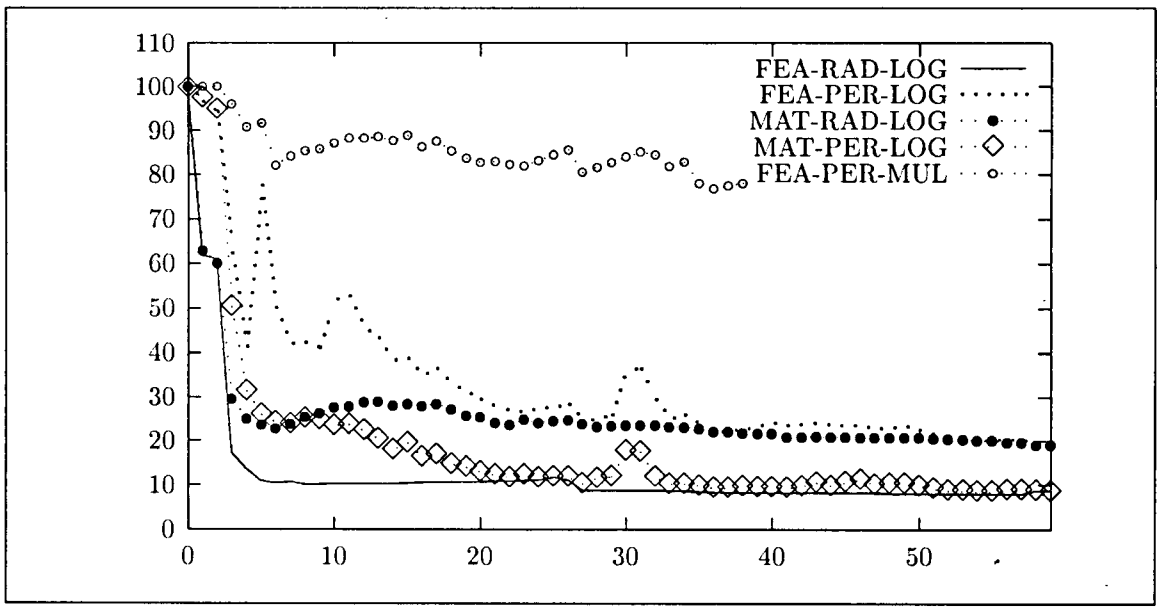


Figure 5.10. The evaluation of various configuration of estimateMul.

In these figures, the vertical axis shows the normalized absolute error shown in percent. The normalized error of 100% corresponds to the error that would be obtained without learning. The horizontal axis corresponds to iterations. In each iteration, the estimator was trained by an *epoch*, that is, a sequence of state-reward pairs obtained through a sequence of actions. In these experiments, one epoch corresponded to one marginal query randomly chosen from the CPCS network. Each iteration contained at most 147 decisions, the average being 40. We ran 1000 iterations, but only showed a small portion of them because the lines became flat (or the similar zigzag-curve) beyond that portion.

For estimateDim, each configuration consists of a choice of function approximator (the multilayer perceptron with 100 hidden units or the radial-basis functions with 250 centers), a choice of training rule (BP or TD(λ) for perceptron, and the contribution-based or clustering-based center selection for RBFN), and a choice of state encoding (the feature set, the matrix, or the significant subportion). Esti-

mateMul has more choices, that is, a choice of whether to normalize, and a choice of whether to encode cost using logarithm.

In order to simplify the presentation, these figures only show the representative configurations. Figure 5.9 shows four lines. The line named FEA-RAD-DIM corresponds to the result obtained by RBFN with the contribution-based training and the feature set. The results given by RBFNs with the feature set or the significant subportion encodings were all similar to this plot, no matter which training rule was used. The line named FEA-PER-DIM corresponds to the result obtained by perceptron with TD(λ) and the feature set. The perceptron with the feature set or the significant subportion encodings were all similar to this plot, no matter which training rule was used. The lines labeled MAT-RAD-DIM and MAT-PER-DIM were obtained with the matrix representation by RBFN with the contribution-based training and perceptron with TD(λ), respectively. Other training rules combined with the matrix representation gave similar plots to the respective lines.

Figure 5.10 shows five lines, of which the four lines labeled LOG were obtained using the logarithm to encode costs and look very similar to their counterpart lines in Figure 5.9. They also represent the same variants as their counterparts do.

The line labeled FEA-PER-MUL was obtained using perceptron with the feature set without using the logarithm to encode costs, and converged to about 80% of the normalized error which was very poor. Those configurations that used bare costs without using the logarithm encoder and those configurations that used the cost normalization were all similar to this line, meaning that they failed to learn.

There are four points to note for these results:

- RBFNs with the feature set or the significant subportion representation converged fast and estimated well.
- Perceptrons were able to make good estimates only with the matrix representation.
- The cost encoding using logarithm was a must for estimateMul.

- The normalization of costs using the worst case cost did not work.

5.4.2 On-line Training of Estimators

In this section, we train various estimators on-line and see if they can create a better policy than the policy defined by zero-estimator which is known to be near-optimal.

In order to avoid the danger of *overfitting*, that is, of memorizing the training data and failing to generalize well to new data points, we use the technique known as *early stopping* [Lang *et al.*, 1990]. Early stopping works as follows. Suppose we have a total of N data points available for training. We randomly set aside a subset of N_h of these points to form the *halting set*. The remaining $N \setminus N_h$ points are given to the learning algorithm. After each training, the current predictive accuracy is measured on the halting set. If the result is better than any previous results, the current state is saved. Training is conducted until some termination condition is met. The saved state is employed to make future predictions.

All of the learning algorithms described in this paper have several parameters (such as the number of centers, the choice of state encoders, and the way to train the neural network) that must be specified by the user. To set these parameters in a principled fashion, we employed the *cross-validation* methodology [Lang *et al.*, 1990]. This method further divides the training set ($N \setminus N_h$) into a subtraining set and a cross-validation set. Alternative configurations of learning algorithms are then tried while training on the subtraining set and testing on the cross-validation set. The best-performing configurations are then employed to train on the full training set. In this methodology, no information from the test set is used to determine any parameters during training.

In our experiments, the data sets consist of marginal queries in the CPCS network. In order to make data sets uniform with respect to their difficulties, we created them as follows. First, we classified 290 non-trivial marginal queries in the CPCS network into eight clusters using the number of relevant variables and the number of relevant expressions as their features. The classification was done by the maximum-

neuron-based [Takefuji *et al.*, 1992] self-organization classification algorithm [Oka *et al.*, 1996]. Then, we randomly picked eight queries each from a different cluster to make a data set. We created the total of four such data sets each of which was mutually exclusive. We used one of them as the subtraining set of eight queries, another as the cross-validation set of eight queries, and the remaining two as the halting set of 16 queries. The subtraining set and the cross-validation set constitute the (full) training set of 16 queries.

The training proceeded as follows.

1. Initialize the neural network. (In the case of perceptrons, the weights were initialized randomly. In the case of RBFNs, the centers were initially empty.)
2. Pick one query from the training (or subtraining) set.
3. Evaluate the query using the current network as the estimator.
4. Train the network by the epoch (the sequence of state-reward pairs) obtained through that evaluation.
5. Test the resulting network on the halting (or cross-validation) set by evaluating all queries in the set using this network as the estimator. If the result is better than any previous results, save the network.
6. If the test result is worse than any of the previous N results, stop, where N is 16 for the cross-validation and 32 for the full training.
7. Repeat from Step 2.

There are two omissions in the above training procedure. First, we used a technique called *experience replay* developed by Lin [1992, 1993] to ensure that the network does not forget good experiences from previous training episodes. This technique works as follows. For each query, we keep track of the epoch of the best factoring found so far. After every four iterations, we replay these "best" epochs to train the network.

The second omission is a random exploration we introduced in order to avoid a local optimum. Let β be a parameter between 0 and 1. During the evaluation of a query in Step 3 in the above procedure, we chose an action at random with probability β instead of using the one-step lookahead search defined in Figure 5.3 and Figure 5.4. Each epoch recorded which decision was made at random, and random decisions would not be used in training in Step 4 and experience replay. Non-random decisions were always used in training even if they belong to an epoch with a random decision.

We explored the following configurations:

- (Dim,Mul) The maximum dimensionality as costs or the logarithm of the number of multiplications as costs.
- (Fea,Mat,Sig) Three choices of state encoders, that is, the feature set, the matrix, or the significant subportion.
- (BP,TD,Cont,Clus) Four choices of training algorithms and network architectures, namely, BP with multilayer perceptron, TD(λ) with multilayer perceptron, the contribution-based center selection with RBFN, and the clustering-based center selection with RBFN.
- (0,1,5) The exploration parameter $\beta = 0, 0.01,$ or 0.05.
- (250,2500) The maximum number of centers in RBFNs be 250 or 2500.

Table 5.1 shows the results of the experiment. It lists all configurations that performed better on the halting set than estimateMul with zero-estimator does, sorted by the number of multiplications required for answering all queries in the halting set. This table also shows the performance of estimateMul with zero-estimator (Mul-Zero), of estimateDim with zero-estimator (Dim-Zero), and of SPI. The column titled dim shows the total dimensionalities required for answering all queries in the halting set.

There are six points to note for this table.

name	mults	dim
Mul-Sig-Clus-5-2500	417224	163
Mul-Fea-Clus-1-250	422596	152
Mul-Fea-Clus-5-250	432382	165
Mul-Sig-Cont-0-2500	438858	165
Mul-Sig-Cont-0-250	443450	166
Mul-Sig-Cont-1-2500	457112	166
Mul-Sig-Clus-0-250	460317	166
Mul-Sig-Clus-0-2500	460447	162
Mul-Sig-Clus-1-2500	465350	162
Mul-Sig-Clus-1-250	465494	170
Mul-Sig-Cont-1-250	477578	164
Mul-Sig-Clus-5-250	480130	142
Mul-Mat-Cont-1-250	499721	157
Mul-Fea-Clus-5-2500	502181	178
Mul-Mat-Cont-5-250	505393	158
Mul-Sig-Cont-5-250	508742	148
Mul-Sig-Cont-5-2500	508742	148
Dim-Zero	511659	144
Mul-Mat-Cont-5-2500	512686	173
Mul-Fea-Clus-0-250	519368	155
Dim-Sig-TD-1	534820	145
Mul-Mat-Cont-0-2500	542256	165
Mul-Mat-Cont-1-2500	551031	166
Mul-Fea-Clus-0-2500	654148	197
Mul-Zero	686766	155
SPI	2.3e8	181

Table 5.1. The results of on-line training of estimators.

- Some configurations of estimators actually succeeded in learning a policy that performed quite well. The best configuration, Mul-Sig-Clus-5-2500, needed 417224 multiplications on the halting set, which was 39% less than that of the greedy factoring algorithm labeled Mul-Zero, and was 99.8% less than that of SPI.

- No configurations with the dimensionality as costs (Dim) succeeded in learning a better policy than that of the greedy counterpart Dim-Zero with respect to either the dimensionality or the number of multiplications.

- All estimators based on multilayer perceptrons (marked BP or TD) failed to achieve improvement over the greedy heuristics.

- Exploration (marked 1 or 5) worked for the clustering-based center selection (Clus), but not for the contribution-based center selection (Cont).

- The clustering-based center selection worked with the feature set (Fea) or the significant proportion (Sig), but the contribution-based center selection did not work with Fea, whereas it worked well with Sig.

- No apparent improvements were achieved by increasing the number of centers from 250 to 2500.

5.4.3 Generalizability

In this section, we test the predictive accuracy of learned heuristics.

The Prediction Test Using The CPCS Network

The first test evaluates the generalizability of the five best heuristics learned in the previous subsection. The evaluation is done by making marginal queries for four variables that are among the most difficult ones found in the CPCS network. These variables were not used in the training set or in the test set, except "appetite" that was used in the test set.

configuration	abdominal pain	appetite	diarrhea	temperature	total	time
Mul-Sig-Cont-0-250	94(12)	141(12)	77(11)	507(13)	820(48)	767
Mul-Sig-Cont-0-2500	130(14)	141(12)	82(11)	507(13)	861(50)	766
Mul-Fea-Clus-5-250	126(14)	330(14)	268(14)	606(15)	1330(57)	156
Mul-Fea-Clus-1-250	126(14)	330(14)	268(14)	606(15)	1330(57)	158
Mul-Zero	126(14)	330(14)	268(14)	606(15)	1330(57)	80
Mul-Sig-Clus-5-2500	126(14)	330(14)	268(14)	606(15)	1330(57)	241

Table 5.2. The prediction test using four most difficult queries in the CPCS network.

Table 5.2 shows the results obtained by the five learned heuristics along with that obtained by the non-learning Mul-Zero heuristic. The first column shows the configuration from which the heuristics were obtained, followed by four columns each of which shows the number of multiplications in thousands and the maximum dimensionality required for the marginal query of the variable shown in the first row. The next column shows the total figures. Finally, the last column shows the CPU time in seconds measured on Pentium II 300MHz with 128MB of memory. This time only includes the time needed for factoring, excluding the time needed for the actual numeric computation of conformal products. The rows are sorted by the total number of multiplications.

Compared to the results shown in Table 3.4 in Chapter 3 obtained by SPI with the same marginal queries, the performance gain is tremendous. For example, the number of multiplications required for "temperature" by SPI using the multiplicative representation was 12 trillion, whereas Mul-Sig-Cont-0-250 requires only five hundred thousand multiplications.

The clustering-based heuristics (those marked with Clus) performed as well as the non-learning heuristic Mul-Zero. This is rather unsatisfactory. However, the contribution-based heuristics (those marked with Cont) performed quite well. The best heuristic Mul-Sig-Cont-0-250 achieved about 40% improvement over Mul-Zero.

The results shown in this subsection imply that we finally succeeded in computing marginal queries in the CPCS network.

The Prediction Test Using The QMR-DT Network

The QMR-DT BN2O network described in Subsection 3.8.1 is a very different and larger network compared to the CPCS network. In this subsection, we see how a heuristic trained by the CPCS network performed with the QMR-DT network.

It turned out that the QMR-DT network is too large to make queries by using the factoring algorithms described in this chapter. This is because at each state an algorithm must generate and evaluate all of the possible successor states. This becomes quite costly in search space with large branching factors, which is about 600 variables in the case of the QMR-DT network as shown in Table 3.3.

An alternative is a procedure called *random sample greedy search* (RSGS) [Zhang and Dietterich, submitted], which generates a random subset of the possible actions and evaluates their resulting states. The best of these actions is then chosen. The size of the random sample is determined incrementally. An initial sample of 10 operators is chosen.[9] Based on the resulting computed values, a permitted amount of error ε, and desired confidence $1 - \delta$, we can compute the probability that the value of the best sampled action is within ε of the best possible action. We continue sampling possible actions until this probability exceeds $1 - \delta$. We set $\varepsilon = 0.1$ and $\delta = 0.1$.

Table 5.3 shows the results of the Scientific American cases employing the heuristic learned by the configuration Mul-Fea-Clus-5-250 of Subsection 5.4.2. The first four columns show the case number, the number of multiplications in thousands, the

[9]Zhang and Dietterich [submitted] used an initial sample of 4 instead of 10. For us, 4 was not reliable, so we increased it to 10.

	Mul-Fea-Clus-5-250			Quickscore	SPI[D'Ambrosio, 1995]
case	mults (K)	dim	time (min)	mults (K)	mults (K)
0	175	14	120	(>2000)	(>2000)
1	18	8	47	(>2000)	148
2	91	12	97	(>2000)	321
3	11	7	43	289	11
4	20	6	74	167	17
5	115	13	113	(>2000)	(>2000)
6	3434	17	242	(>2000)	(>2000)
7	26	8	95	343	38
8	81	12	95	(>2000)	1252
9	25	6	73	191	36
10	50	11	65	–	–
11	84	10	133	–	–
12	48	10	66	–	–
13	13	7	48	–	–
14	53446	22	450	–	–
15	11	5	54	–	–

Table 5.3. The prediction test using Scientific American cases in the QMR-DT network.

dimensionality, and CPU-time in minutes.

Compared to the results shown in Table 3.3 of SPI with the multiplicative representation, our algorithm performed much better in every case. For example, the worst cases of SPI, that is, Case 4 with 1.5×10^{78} multiplications (258 dimensionality) and Case 14 with 9.0 billion multiplications (26 dimensionality) were reduced to 20160 multiplications (6 dimensionality) and 53 million multiplications (22 dimensionality), respectively.

Table 5.3 also shows the results taken from [D'Ambrosio, 1995]. In [D'Ambrosio, 1995], D'Ambrosio compared two methods: Quickscore [Heckerman, 1989] and SPI. This version of SPI used the additive representation of noisy-or, and its heuristic value function was different from the one used in the current SPI. The fifth and sixth columns in Table 5.3 show the number of multiplications in thousands required by Quickscore and SPI, respectively. Those entries marked with (>2000) indicate that the computation aborted because the number of multiplication exceeded a predefined limit of two million multiplications. D'Ambrosio used only the first 10 cases.[10] Note that he ignored all negative evidences in his experiments, but the effects of ignoring them was negligible. Comparison with these results clearly shows the superiority of our method.

5.5 Summary

In this chapter, we tried to develop a good factoring heuristic automatically using the reinforcement learning techniques. First, we viewed the optimal factoring problem as a reinforcement learning problem and developed factoring algorithms that worked with future cost estimators. We used two neural network architectures (multilayer perceptrons and radial-basis function networks) to implement future cost estimators. In order to use neural networks, we developed three state-encodings and a cost-encoding. For training radial-basis function networks, we developed two algorithms

[10] In [D'Ambrosio, 1995], those cases are named from 1 to 10.

for selecting centers. Empirical studies demonstrated that (1) some configurations of estimators could actually approximate a future cost function of this problem domain, (2) certain configurations of estimators could be trained on-line to acquire a policy that surpasses the existing heuristics, and (3) learned heuristics generalized well to unknown situations.

Chapter 6
Conclusion

In this chapter, we summarize the contribution of our research and list some of the anticipated future directions.

We began this thesis by studying the representations of independence of causal influence (ICI) models. We developed new multiplicative representations of ICI models. They are easy to use because any standard inference algorithm can work with them. Also, they allow for exploiting ICI fully because they do not impose any constraints on inference algorithms. We showed the correctness of the representations, and demonstrated how to specialize the general representation for specific ICI models in order to gain representational efficiency.

We then studied factoring algorithms. We developed a methodology for applying the reinforcement learning techniques to the optimal factoring problem, and developed factoring algorithms that can work with heuristics, automatically generated by the reinforcement learning techniques.

Finally, we empirically demonstrated that the combination of the new representation and the new factoring algorithm was more efficient and allowed for inference in larger Bayesian networks than existing methods.

The following lists some future directions for our research.

- In chapter 3, we demonstrated how to optimize multiplicative representations. Optimization is currently performed by humans, but this task could be automated. An automatic representation optimizer will not only help one to apply the multiplicative representation to a new ICI model, but could also facilitate better understanding of the multiplicative representations through its development.

- In chapter 5, we developed future cost estimators using two architectures of neural networks and four training methods. This is not at all a comprehensive list of function approximators that can implement the future cost estimators. We would like to experiment with other function approximators such as CMAC [Miller *et al.*, 1990] and other training methods such as the supervised center selection for radial-basic function networks [Wettschereck and Dietterich, 1992, Haykin, 1994].

- Comparing the results obtained by using SPI with those of our algorithms suggests that there is a trade-off between the time spent on factoring and the quality of factoring. Usually the quality of factoring matters more than the time spent on factoring because the quality of factoring is directly related to memory, and memory is the bottleneck. Our algorithms are based on this idea and spent as much time as needed on factoring. However, in order for the algorithms to be more practical, we need to reduce as much CPU time as possible. One way to reduce the time is to combine SPI and our algorithms in a way that use SPI as long as memory is sufficient and switch to our algorithm only if the quality of factoring SPI generated was too bad. Another interesting way to reduce the runtime CPU requirement is to precompute the variable-elimination ordering given a Bayesian network.

BIBLIOGRAPHY

[Beinlich *et al.*, 1989] I. Beinlich, G. Suermondt, R. Chavez, and G. Cooper. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on AI and Medicine.* Springer-Verlag, 1989.

[Binder *et al.*, 1997] J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29, 1997.

[Borujerdi *et al.*, 1998] M. Borujerdi, B. D'Ambrosio, and M. Takikawa. Representing intercausal independences in Bayesian networks. *Unpublished manuscript*, 1998.

[Cooper, 1990] G. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2–3):393–406, 1990.

[D'Ambrosio, 1994] B. D'Ambrosio. SPI in large BN2O networks. In Poole and Lopez de Mantaras, editors, *Tenth Annual Conference on Uncertainty on AI.* Morgan Kaufmann, July 1994.

[D'Ambrosio, 1995] B. D'Ambrosio. Local expression languages for probabilistic dependence. *International Journal of Approximate Reasoning*, 13:61–81, 1995.

[Dechter, 1996] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, 1996.

[Díez, 1993] F. J. Díez. Parameter adjustment in Bayes networks: the generalized noisy-or gate. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 99–105, 1993.

[Geiger *et al.*, 1989] D. Geiger, T. Verma, and J. Pearl. D-separation: From theorems to algorithms. In *Proceedings of the Fifth Workshop on Uncertainty in AI,*

pages 118–125, August 1989.

[Good, 1961] I. Good. A causal calculus (i). *British Journal of Philosophy of Science*, 11:305–318, 1961.

[Haykin, 1994] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Macmillan College Publishing Company, 1994.

[Heckerman and Breese, 1994] D. E. Heckerman and B. Breese. A new look at causal independence. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 286–292, Seattle, WA, 1994.

[Heckerman *et al.*, 1992] D. E. Heckerman, E. J. Horvitz, and B. N. Nathwani. Toward normative expert systems: Part I the Pathfinder Project. *Methods of Information in Medicine*, 31:90–105, 1992.

[Heckerman, 1989] D. Heckerman. A tractable inference algorithm for diagnosing multiple diseases. In *Proceedings of the Fifth Conference on Uncertainty in AI*, pages 174–181, August 1989.

[Henrion and Druzdel, 1990] M. Henrion and M. Druzdel. Qualitative propagation and scenario-based explanation of probabilistic reasoning. In *Proceedings of the Sixth Conference on Uncertainty in AI*, pages 10–20, August 1990.

[Henrion, 1987] M. Henrion. Some practical issues in constructing belief networks. In L. Kanal, T. Levitt, and J. Lemmer, editors, *Uncertainty in Artificial Intelligence, Vol 3*, pages 161–174. North-Holland, 1987.

[Jensen *et al.*, 1990] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[Kaelbling *et al.*, 1996] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Kim and Pearl, 1983] J. H. Kim and J. Pearl. A computational model for causal and diagnostic reasoning in infe rence engines. In *Proceedings of IJCAI-83.* Karlsruhe, FRG, 1983.

[Kjærulff, 1993] U. Kjærulff. Aspects of efficiency improvements in Bayesian networks. Thesis, Faculty of Technology and Science, Aalborg University, 1993.

[Kohonen, 1993] T. Kohonen. Physiological interpretation of the self-organization map algorithm. *Neural Networks*, 6:895–905, 1993.

[Kretchmar and Anderson, 1997] R. M. Kretchmar and C. W. Anderson. Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In *Proceedings of the 1997 International Conference on Neural Networks*, 1997.

[Lang *et al.*, 1990] K. J. Lang, G. E. Hinton, and A. Waibel. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3(1):23–43, 1990.

[Lauritzen and Spiegelhalter, 1988] S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, B 50, 1988.

[Li and D'Ambrosio, 1994] Z. Li and B. D'Ambrosio. Efficient inference in Bayes networks as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11:1–24, 1994.

[Lin, 1992] L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[Lin, 1993] L. Lin. Reinforcement learning for robots using neural networks. Phd thesis, Carnegie-Mellon University, 1993.

[Miller *et al.*, 1990] W. T. Miller, F. H. Glanz, and L. G. Kraft. CMAC: An associative neural network alternative to backpropagation. *Proceedings of the IEEE*, 78:1561–1567, 1990.

[Oka *et al.*, 1996] S. Oka, T. Ogawa, T. Oda, and Y. Takefuji. A new self-organization classification algorithm for remote-sensing images. In *Adaptive Distributed Parallel Computing Symposium*, August 1996.

[Olesen *et al.*, 1989] K. G. Olesen, U. Kjærulff, F. Jensen, B. Falck, S. Andreassen, and S. K. Andersen. A munin network for the median nerve — a case study on loops. *Applied Artificial Intelligence*, 3:384–403, 1989.

[Parker and Miller, 1987] R. C. Parker and R. A. Miller. Using causal knowledge to create simulated patient cases: the CPCS project as an extension of Internist-1. In *Proceedings of the Eleventh Annual Symposium on Computer Applications in Medical Care*, pages 473–480. IEEE Comp Soc Press, 1987.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, Palo Alto, 1988.

[Pradhan *et al.*, 1994] M. Pradhan, G. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 484–490, 1994.

[Rish and Dechter, 1998] I. Rish and R. Dechter. On the impact of causal independence. In *1998 Stanford Spring Symposium on Interactive and Mixed-Initiative Decision-Theoretic Systems*, March 1998.

[Rose, 1973] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In Ronald C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, 1973.

[Russell and Norvig, 1995] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Series in Artificial Intelligence. Prentice-Hall Inc., 1995.

[Sato and Tinney, 1963] N. Sato and W. F. Tinney. Techniques for exploiting the sparsity of the network admittance matrix. In *PAS*, pages 944–950. IEEE, 1963.

[Shachter *et al.*, 1990] R. Shachter, B. D'Ambrosio, and B. DelFavero. Symbolic probabilistic inference in belief networks. In *Proceedings Eighth National Conference on AI*, pages 126–131. AAAI, August 1990.

[Shachter, 1988] R. Shachter. Probabilistic inference and inference diagrams. *Operations Research*, 36(6):589–604, July–August 1988.

[Shafer and Shenoy, 1990] G. Shafer and P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

[Srinivas, 1993] S. Srinivas. A generalization of the noisy-or model. In *Ninth Annual Conference on Uncertainty on AI*, pages 208–218, July 1993.

[Sutton, 1988] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, August 1988.

[Takefuji *et al.*, 1992] Y. Takefuji, K. C. Lee, and H. Aiso. An artificial maximum neural network : a winner-take-all neuron model forcing the state of the system in a solution domain. *Biological Cybernetics*, 67:243–251, 1992.

[Tesauro, 1992] G. J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4):257–277, 1992.

[Tinney and Walker, 1967] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 55:1801–1809, 1967.

[Wen, 1990] W. X. Wen. Optimal decomposition of belief networks. In *Proceedings of the Sixth Workshop on Uncertainty in Artificial Intelligence*, pages 245–256, Cambridge, MA, 1990.

[Wettschereck and Dietterich, 1992] D. Wettschereck and T. G. Dietterich. Improving the performance of radial basis function networks by learning center locations. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann Publishers Inc., 1992.

[Zhang and Dietterich, 1995] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[Zhang and Dietterich, submitted] W. Zhang and T. G. Dietterich. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. submitted.

[Zhang and Poole, 1994] N. L. Zhang and D. Poole. Intercausal independence and heterogeneous factorization. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 606–614, Seattle, WA, 1994.

[Zhang and Poole, 1996] N. L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.

[Zhang and Yan, 1997] N. L. Zhang and L. Yan. Independence of causal influence

and clique tree propagation. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, 1997.

[Zhang, 1995] N. L. Zhang. Inference with causal independence in the CPSC network. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Montreal, Canada, August 1995.