A COMPUTATIONAL THEORY OF DECISION NETWORKS

By

Nevin Lianwen Zhang

B. Sc. (Mathematics) China University of Elect. Sci. & Tech.

M. Sc., Ph. D. (Mathematics) Beijing Normal University

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

COMPUTER SCIENCE

We accept this thesis as conforming

to the required standard

. . . .

. . . .

. . . .

. . . .

. . . .

THE UNIVERSITY OF BRITISH COLUMBIA

December 1993

Computer Science
The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1Z1

Date:

Abstract

This thesis is about how to represent and solve decision problems in Bayesian decision theory (e.g. Fishburn 1988). A general representation named decision networks is proposed based on influence diagrams (Howard and Matheson 1984). This new representation incorporates the idea, from Markov decision processes (e.g. Puterman 1990, Denardo 1982), that a decision may be conditionally independent of certain pieces of available information. It also allows multiple cooperative agents and facilitates the exploitation of separability in the utility function. Decision networks inherit the advantages of both influence diagrams and Markov decision processes.

Both influence diagrams and finite stage Markov decision processes are stepwise-solvable, in the sense that they can be evaluated by considering one decision at a time. However, the evaluation of a decision network requires, in general, simultaneous consideration of all the decisions. The theme of this thesis is to seek the weakest graph-theoretic condition under which decision networks are guaranteed to be stepwise-solvable, and to seek the best algorithms for evaluating stepwise-solvable decision networks.

A concept of decomposability is introduced for decision networks and it is shown that when a decision network is decomposable, a divide and conquer strategy can be utilized to aid its evaluation. In particular, when a decision network is stepwise-decomposable it can be evaluated not only by considering one decision at a time, but also by considering one portion of the network at a time. It is also shown that stepwise-decomposability is the weakest graphical condition that guarantees stepwise-solvability.

Given a decision network, there are often nodes and arcs that can harmlessly removed. An algorithm is presented that is able to find and prune all graphically identifiable

removable nodes and arcs.

Finally, the relationship between stepwise-decomposable decision networks (SDDN's) and Markov decision process is investigated, which results in a two-stage approach for evaluating SDDN's. This approach enables one to make use of the asymmetric nature of decision problems, facilitates parallel computation, and gives rises to an incremental way of computing the value of perfect information.

Table of Contents

List of Figures

List of symbols

α, β, γ: value of a variable or of a set of variables (nodes)

B, B_1, B_2: sets of nodes (variables)

C: the set of random nodes in a decision network

d, d_i: a decision node (variable)

δ: a policy for a decision network, i.e a vector of decision functions

Δ: policy space, the set of all policies

Δ_i, Δ_d: decision function space of d_i, d

δ_i, δ_d: a decision function for a decision node d_i, d

δ^o: optimal policy of a decision network

δ_i^o, δ_d^o: optimal decision function for a decision node d_i, d

D: the set of decision nodes in a decision network

$e(d, \pi_d)$: the evaluation functional of a simple semi-decision network

$E[\mathcal{N}]$: the optimal expected value of a decision network $\mathcal{N}$

$E[\mathcal{N}|S]$: the optimal conditional expected value of a decision network $\mathcal{N}$ given S

$E_\delta[\mathcal{N}]$: the expected value of a decision network $\mathcal{N}$ under policy δ

$E_\delta[\mathcal{N}|S]$: the conditional expected value under policy δ

$\mathcal{K}$: a decision network skeleton.

$\mathcal{K}_I(d, \mathcal{K})$, $\mathcal{K}_I$: body of a decision network skeleton.

$\mathcal{K}_{II}(d, \mathcal{K})$, $\mathcal{K}_{II}$: tail of a decision network skeleton.

μ: utility or value function

$\mathcal{N}$: a Bayesian network or a decision network.

$\mathcal{N}_I(d, \mathcal{N})$, $\mathcal{N}_I$: body of a decision network.

$\mathcal{N}_{II}(d, \mathcal{N})$, $\mathcal{N}_{II}$: tail of a decision network.

$\mathcal{N}^c$: condensation of a decision network.

$\mathcal{N}(d_i, d_{i+1})$: section of a decision network.

$\mathcal{N}_\delta$: the Bayesian network induced from a decision network $\mathcal{N}$ by a policy δ.

Ω: the frame of a variable or the Cartesian product of the frames of variables

$\mathcal{P}$: a set of probabilities

$P(c|\pi_c)$: the conditional probability of random variable c given its parents π_c

P_δ, $P_\delta(X)$: the joint probability induced by a policy δ

P_0: the multiplication of all the conditional probabilities in a simple semi-decision network

π_x: the set of parents of node x in a network

S: a set of variables, usually related to separator

V: the set of value nodes in a decision network

X: the set of random and decision nodes in a decision network

X_I: the set of random and decision nodes in upstream set

X_{II}: the set of random and decision nodes in downstream set

Y: the set of all the nodes in a decision network

Y_I, $Y_I(d, \mathcal{N})$, $Y_I(d, \mathcal{K})$: upstream set

Y_{II}, $Y_{II}(d, \mathcal{N})$, $Y_{II}(d, \mathcal{K})$: downstream set

Acknowledgement

First, I would like to thank my thesis supervisor David Poole for his guidance, encouragement, support, and friendship. David, it has been great fun being your student.

I am grateful to Runping Qi for stimulating discussions, his friendship, and his poem.

Members of the UBC reasoning group have been always the first to hear about and to criticize my ideas. Besides David and Runping, other members of the group include Brent Boerlage, Craig Boutillier, Mike Horsch, Keiji Kanazawa, Ying Zhang. Folks, thank you all for bearing with me and for all your valuable comments and suggestions.

Thanks also go to Bruce D'Ambrosio, David Kirkpatrick, David Lowe, Alan Mackworth, Raymond Ng, Nicholos Pippenger, Martin Puterman, and Jim Zidek for serving on my supervisory committee and/or my examination committee.

I thank the department staffs: Deborah, Everlyn, Gale, Grace, Jean, Joyce, Monica, and Sunnie for their help and friendliness.

I thank my friends, office mates, lab-mates for sharing wonderful times, and for essential support during some not-so-wonderful times.

Finally, I would like to thank all my teachers from the past: Xinfu Zhang, Duangcai Wang, Quanglin Hou, Peizhuang Wang, Sijian Yan, Glenn Shafer, Prakash Shenoy, and David Poole, to name a few. Together, they have made an academice career possible for a shabby boy from a remote and poor Sichuan village.

Chapter 1

Introduction

This thesis is about how to represent and solve decision problems in Bayesian decision theory (e.g. Fishburn 1988). A general representation named decision networks is proposed based on influence diagrams (Howard and Matheson 1984). This new representation incorporates the idea, from Markov decision processes (e.g. Denardo 1982), that a decision may be conditionally independent of certain pieces of available information. It also allows multiple cooperative agents and facilitates the exploitation of separability in the utility function.

Influence diagrams are stepwise-solvable, that is they can be evaluated by considering one decision at a time (Shachter 1986). However, the evaluation of a decision network requires, in general, simultaneous consideration of all the decisions. The theme of this thesis is to seek the weakest condition under which decision networks are stepwise-solvable, and to search for the best algorithms for evaluating stepwise-solvable decision networks.

This introductory chapter provides a synopsis of our theory, and describes how and why it differs from its two mother theories: the theory of influence diagrams and the theory of Markov decision processes.

The synopsis in Section 1.1 below describes salient features of the following chapters. Section 1.2 reviews Bayesian decision theory, and Section 1.3 reviews two methodologies for decision analysis, namely decision trees and influence diagrams.

An influence diagram is a representation of a single agent's view of the world as relevant to a decision problem; it spells out information availability for each decision.

Several constraints follow from its semantics (Section 1.4). A decision network, on the other hand, is a representation of a group of cooperative agents' view of the world; it indicates both information availability and dependency for each decision node. Some constraints of influence diagrams do not apply to decision networks.

Syntactically decision networks are arrived at by lifting some of those constraints (Section 1.4). Reasons for lifting the constraints originate from the demand of a more general representation framework in decision analysis (Section 1.5), and from the effort to provide Markov decision processes with a graph-theoretic language (Sections 1.6). More importantly, the lifting of constraints also allows us to apply more techniques in solving a problem and hence leads to better and more efficient algorithms (Section 1.7).

Section 1.9 echoes the synopsis by providing a description of all the subclasses of decision networks we will encounter later. Finally, Section 1.10 suggests who might be interested in this thesis, and why.

1.1 Synopsis

Our goal is to enable computers to help decision makers solve complex decision problems. The first step in achieving this goal is to design a language or some kind of representation framework so that decision makers and computers can communicate about decision problems. Two frameworks exist previously, namely *influence diagrams* (Howard and Matheson 1984, Shachter 1986) and *Markov decision processes (MDP)* (see, for example, Denardo 1982).

MDP's are a model of sequential decision making for the sake of controlling dynamic systems; it is special-purpose. Influence diagrams are a general framework for decision analysis; however they are always required to satisfy the so-called no-forgetting constraint, which requires a decision node and its parents be parents to all subsequent decision nodes.

There are at least three reasons that a decision maker would be interested in lifting the no-forgetting constraint (details coming up in later in this chapter):

1. The decision maker may be able to qualitatively determine that a decision does not depend on certain pieces of available information. As a matter of fact, one result of MDP theory is that given the current state of the dynamic system, the optimal current decision is independent of previous states and decisions, even though they are known. Such conditional independence for decisions can-not be represented in influence diagrams. The no-forgetting constraint is supposed to capture the rationale that information available earlier should also be available later. In the mean time, unfortunately, it also excludes the possibility of earlier information being irrelevant later.

2. There may be several cooperative decision makers, each responsible for a subset of decisions. When communication is not feasible or is too expensive, information available earlier to one decision maker may not be available later to a different decision maker. Furthermore, there may not be a predetermined ordering among the decisions. This defeats not only the no-forgetting constraint, but also another constraint — the so-called regularity constraint, which requires a total ordering among the decisions.

3. It has been noticed that given an influence diagram, a decision node may turn out to be independent of some of its parents. In such a case, the arcs from those parents to the decision node can be harmlessly removed. It is a good idea to remove such arcs at a preprocessing stage, since it yields a simpler diagram. However, removing arcs from an influence diagram leads to the violation of the no-forgetting constraint.

In this thesis, we lift the no-forgetting constraint, together with two other constraints previously imposed on influence diagrams, and make due semantic modifications to arrive at *decision networks (DN)*, a framework for representing decision problems that is more general then both influence diagrams and finite stage MDP's.

In both influence diagrams and finite stage MDP's, a decision problem can be solved by considering one decision at a time, while solving a decision problem in the framework of general DN's requires simultaneous consideration of all the decision nodes, even when the structure of the problem is simple (Chapter 3). One of the themes of this thesis is to investigate when a decision network can be solved by considering one decision node at a time. We give a graph-theoretic criterion called stepwise-decomposability (Chapter 5); and we prove that this criterion is the weakest graph-theoretic criterion that guarantees stepwise-solvability (Chapter 8).

Another theme of this thesis is to develop algorithms for evaluating stepwise-decomposable decision networks (SDDN's). As a first step, we find a way to prune all the removable arcs that are graph-theoretically identifiable (Chapters 7, 8). It is shown that pruning removable arcs from SDDN's does not destroy the stepwise-decomposability (Section 7.3).

A divide-and-conquer procedure named EVALUATE1 for evaluating SDDN's is developed (Chapters 4, 5, and 6). This procedure has several other advantages in addition to embracing the divide and conquer strategy. It clearly identifies all the Bayesian network (BN) inferences involved in evaluating a SDDN. Consequently, it can be readily implemented on top of any BN inference systems. The procedure does not require arc reversals and induces little numerical divisions. Finally, the procedure explicitly exploits independencies allowed by multiple value nodes (Section 6.5).

A two stage procedure named EVALUATE2 is also developed on the basis of EVALUATE1 (Chapter 9) as a result of our investigation on the relationship between MDP's and SDDN's. EVALUATE2 inherits all the advantages of EVALUATE1. Furthermore,

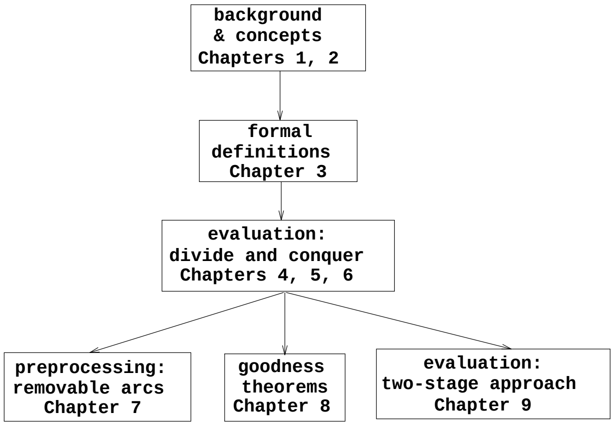

Figure 1.1: Dependencies among the chapters.

it enables one to exploit the asymmetric nature of decision problems[1] and opens up the possibility of parallel processing (Section 9.2.1). It also leads to an incremental way of computing the value of information (Zhang *et al* 1993b).

There have been a number of previous algorithms for evaluating influence diagrams. Since influence diagrams are special SDDN's, there can also be evaluated by the algorithms developed in this thesis for evaluating SDDN's. Our algorithms are shown to be advantageous over the previous algorithms in a number of aspects (Sections 6.5, 9.6.1).

The dependency relationships among the nine chapters of this thesis are shown in Figure 1.1.

The remainder of this chapter relates the background of this thesis, and gives a more detailed account to the points put forward by the story. Let us begin with Bayesian decision theory.

[1]This was first pointed out by Qi (1993).

1.2 Bayesian decision theory

We make numerous decisions every day. Many of our decisions are made in the presence of uncertainty. A simple example is to decide whether or not to bring the umbrella in light of the weather forecast. If the weather man were an oracle such that his prediction is always correct, then the decision would be easy. Bring the umbrella if the weather man predicts rain and leave the umbrella home if he predicts no rain. However real life forecasts are not as predictive as we wish. Instead of saying that it will rain, the weather man says, for instance, that there is a sixty percent chance of precipitation.

We would be happy if we bring the umbrella and it rains, or if we leave the umbrella at home and it does not rain. But we would regret carrying the umbrella around if it does not rain, and we would regret even more not having the umbrella with us when it rains. We have all made this decision many times in our lives, and did not find it hard because we thought this particular decision is not significant. However, there are decisions, such as buying a house or making a major investment in the stock market, that are of significance to us. In such cases, we want to make rational decisions.

Understanding how to make rational decisions is also important for building intelligent systems.

Bayesian decision theory provides a framework for rational decision making in the face of uncertainty. One setup for Bayesian theory consists of a set S of possible states of the world, a set O of possible observations, and a set Ω_d of decision alternatives. There is a conditional probability distribution $P(o|s)$ describing how likely it is to observe o when the world is in state s, and there is a prior probability distribution $P(s)$ describing how likely the world is to be in state s. There is also a utility function $\mu(d, s)$, which represents the reward to the decision maker if he chooses the decision alternative $d \in \Omega_d$ and the world is in state $s \in S$. The problem is to decide on a *policy* , i.e a mapping from O to Ω_d, which dictates the action to take for each observation.

In our example, the possible states of worlds are `rain` and `no-rain`. The observations are all the possible forecasts, that is the set $\{$"`there is an` x `percent chance of precipitation`"$\mid x \in \{0, \ldots, 100\}\}$. There are two possible decision alternatives: `take-umbrella` or `not-take-umbrella`. The conditional probability of the forecast that "`there is an` x `percent chance of precipitation`" given `rain` and the prior probability of `rain` are to be assessed from our experience. Our utilities could be as shown in the following table:

	`rain`	`no-rain`
`take-umbrella`	0	-10
`not-take-umbrella`	-100	0

The problem is to decide whether or not to bring the umbrella in light of the weather forecast.

The *expected utility* E_δ *induced by the policy* $\delta : O \rightarrow \Omega_d$ is defined by

$$E_\delta = \sum_{s \in S, o \in O} P(s)P(o|s)\mu(\delta(o), s). \tag{1.1}$$

The *principle of maximizing the expected utility* (von Neumann and Morgenstein 1944, Savage 1954) states that a rational decision maker choses the policy δ^o that satisfies

$$E_{\delta^o} = max_\delta E_\delta, \tag{1.2}$$

where the maximization is over all possible policies. The quantity $max_\delta E_\delta$ is called the *optimal expected value* of the decision problem.

1.3 Decision analysis

In the setup of Bayesian decision theory given in the previous section, there is only one decision to make. Applications usually involve more than one decision (e.g. Hosseini

1968). This thesis is about how to apply Bayesian decision theory to problems that involve multiple decisions and multiple variables.

There exist two methodologies that deal with multiple decisions, namely decision analysis (e.g. Smith 1988) and Markov decision processes (e.g. Denardo 1982). Between them, decision analysis is more general-purpose. It emphasizes exploring the structures of complex problems. In a sense, it has a representation advantage. On the other hand, finite stage Markov decision processes deal with decisions for controlling a dynamic system (e.g. Bertsekas 1976). This class of multiple-decision problems have relatively simple structures. Finite stage Markov decision processes emphasize problem solving by using the technique of dynamic programming. In a sense, they have a computational advantage.

One goal of this thesis is to combine the representational advantage of decision analysis and the computational advantage of finite stage Markov decision processes.

This section gives a brief account of decision analysis. A latter section will touch on finite stage Markov decision processes.

1.3.1 Decision trees

Within decision analysis, there are two frameworks for representing the structures of decision problems, namely decision trees (North 1968, Raiffa 1968) and influence diagrams (Howard and Matheson 1984). Decision trees represent the structure of a decision problem all at one level, while influence diagrams distinguish three levels of specification for a decision problem.

Consider the following oil wildcatter problem taken from (Raiffa 1968). The oil wildcatter must decide either to drill or not to drill. He is uncertain whether the hole is dry, wet or soaking. The prior probabilities (obtained from experts) are as follows.

dry	wet	soaking
.500	.300	.200

His utilities are given in the following table.

	dry	wet	soaking
drill	-$70,000	$50,000	$200,000
not-drill	0	0	0

At a cost of $10,000, our wildcatter could conduct a test on the seismic structure, which will disclose whether the terrain below has no structure, closed structure, or open structure. The conditional probabilities of the test result given the states of the hole are given in the following table.

	dry	wet	soaking
no structure	.600	.300	.100
open structure	.300	.400	.400
closed structure	.100	.300	.500

The problem is whether or not the wildcatter should conduct the test? And whether or not he should drill?

The decision tree for this problem is shown in Figure 1.2, where rectangles stand for decision variables and ellipses stand for random variables. The values of the variables and the corresponding probabilities appear on the edges. The tree is to be read as follows.

If our wildcatter decides not to test, he must make the drill decision based on no information. If he decides not to drill, that is the end of the story. He does not make nor lose any money. If he decides to drill, there is a 50 percent chance that the hole is dry, in which case he loses $70,000; there is a 30 percent chance that the hole is wet, in

Figure 1.2: A decision tree for the oil wildcatter problem.

which case he makes \$50,000; and there is a 20 percent chance that the hole is soaking, in which case he makes \$200,000.

If he decides to test, there is a 41 percent chance that there turns out to be no seismic structure. The probability .41 is calculated by using Bayes' rule from the prior and conditional probabilities given. If he still decides to drill, there is a 73 percent chance that the hole is dry, in which case he loses \$80,000, for now the test has cost him \$10,000 already. Again the probability .73 is calculated by using Bayes' rule from the prior and conditional probabilities given. There will be a 22 percent chance that the hole is wet, in which case he makes \$40,000; and there will be only a 5 percent chance that the hole is soaking, in which case he makes \$190,000. And so on and so forth.

An optimal policy and the optimal expected value of a decision tree can be found by the so-called folding backing strategy (Raiffa 1968, Smith 1987).

Figure 1.3: An influence diagram for the oil wildcatter problem.

1.3.2 Influence diagrams

Decision trees came into being during the 1930's and 1940's (Shafer 1990). They were the major framework for representing the structure of a decision problem until late seventies and early eighties, when researchers began to notice the shortcomings of decision trees. For one thing, decision trees are usually very complicated. According to Smith (1988), the first thing to do in decision analysis is to find a large piece of paper. A more important drawback of decision trees include that they are unable to represent independencies.

Influence diagrams were introduced by Howard and Matheson (1984) (see also Miller *et al* 1976) to overcome the shortcomings of decision trees. They specify a decision problem in three levels: relation, function, and number. The level of relation indicates that one variable depends in a general way on others; for example `test-result` probabilistically depends on `test` and `seismic-structure`; and `utility` deterministically depends on `test`, `drill` and `oil-underground`. At the level of number, we specify numerical probabilities for each conditional and unconditional event; and the numerical value of a variable given the values of the variables it deterministically depends upon. The level of function describes the form of dependencies, which is useful in arriving at the level of number. Two examples: profit equals revenue minus cost; if a man is in his thirties, then the probability distribution of his income is a normal distribution with mean $45,000 and standard deviation 1000.

Figure 1.3 shows the level of relation of the influence diagram for our oil wildcatter problem. The diagram clearly shows that the `test` decision is to be made based on no information, and the `drill` decision is to be made based on the decision to `test` and the `test-result`. The random variable `test-result` directly depends on the decision to `test` and the `seismic-structure`, and it is independent of `oil-underground` given `test` and `seismic-structure`. The random variable `seismic-structure` directly depends on `oil-underground`. Finally, the `utility` deterministically depends on `test`, `drill`, and `oil-underground`.

At the level of number, we need to specify the prior probability of `oil-underground`, the conditional probability of `seismic-structure` given `oil-underground`, and the conditional probability of `test-result` given `test` and `seismic-structure`. We need also to specify the value of `utility` for each array of values of `test`, `drill` and `oil-underground`.

In Howard and Matheson (1984), an influence diagram is transformed into a decision tree in order to be evaluated to find an optimal policy and the optimal expected value. Shachter (1986) shows that influence diagrams can be directly evaluated.

Before moving on, let us note that variables will be also called nodes when they are viewed as members of an influence diagram. With that in mind, we can now say that influence diagrams consists of three types of nodes: *decision nodes* , *random node* and a single *value node* , where the value node represent utilities.

1.3.3 Representing independencies for random nodes

A quick comparison of the influence diagram in Figure 1.3 with the decision tree in Figure 1.2 should convince the reader that influence diagrams are intuitive, as well as more compact. They make numerical assessments easier (Howard and Matheson 1984). Furthermore, they serve better than decision trees to address the issue of value of information (Matheson 1990).

Figure 1.4: An influence diagram for the extended oil wildcatter problem.

The most important advantage of influence diagrams over decision trees, however, lies their ability to represent independencies for random nodes at the level of relation.

This point could be illustrated by using the oil wildcatter problem. For later convenience, consider extending the oil wildcatter problem by considering one more decision — the decision of determining a `oil-sale-policy` based on oil quality and `market-information`. The influence diagram for this extended oil wildcatter problem is shown in Figure 1.4. By using the so-called d-separation criterion (Pearl 1988), one can read from the network that `market-information` is marginally independent of `test`, `test-result`, `seismic-structure`, `oil-underground`, `drill`, and `oil-produced`. Also, as mentioned in section 1.3.2, `test-result` is independent of `oil-underground` given `test` and `seismic-structure`. Those marginal and conditional independencies can not be represented in decision trees.

1.4 Constraints on influence diagrams

There are five constraints that one can impose on influence diagrams: namely the acyclicity constraint, the regularity constraint, the no-forgetting constraint, the single value node constraint, and the no-children-to-value-node constraint. Before this thesis, only influence diagrams that satisfy all those constraints have been studied [2]. In this sense,

[2] With the exception of Tatman and Shacter (1990), who deal with one super value node. A super value node may consist of many value nodes. See sections 1.5.2 and 1.6.2 for details.

we say that the five constraints have always been imposed on influence diagrams. From now on, we always mean an influence diagram that satisfies all those five constraints by the term "influence diagram".

The *acyclicity constraint* requires that an influence diagram does not contain any directed cycles. The *regularity constraint* requires that there exists a directed path that contains all the decision nodes. The *no-forgetting constraint* requires that each decision node and its parents be parents to all subsequent decision nodes. The *single value node constraint* requires that there be only one value node, and the *no-children-to-value-node constraint* requires that the value node have no children.

The regularity constraint is due to the fact that an influence diagram is a representation of a single agent's view of the world as relevant to a decision problem. The no-forgetting constraint is due to the fact that in an influence diagram, arcs into decision nodes are interpreted as indications solely of information availability. The constraint follows if the agent does not forget information (Howard and Matheson 1984).

This thesis is about *decision networks* , a representation framework for multi-decision problems that is more general than influence diagrams. Syntactically, decision networks are arrived at by lifting the regularity, no-forgetting, and single value node constraints from influence diagrams. Semantically, a decision network is a representation of the view of the world of a group of cooperative agents with a common utility; and in decision networks, arcs into a decision node indicate both information availability and dependency.

The idea of a representation framework for decision problems free of the regularity and no-forgetting constraints is not new. Howard and Matheson (1984) have suggested the possibility of such a framework. The next three sections conduct a close examination on the reasons for lifting the regularity, no-forgetting, and single value node constraints from influence diagrams. The reasons arise from decision analysis, from Markov decision processes.

1.5 Lifting constraints: Reasons pertaining to decision analysis

1.5.1 Lifting the no-forgetting constraint

As mentioned in the synopsis, there are three major reasons for lifting the no-forgetting constraints. The first reason is explained in detail in this subsection. The second and third reasons will be addressed in the next two subsections.

Semantics for arcs into decision nodes and independencies for decision nodes

The no-forgetting constraint originates from the interpretation of arcs into decision nodes as indications of only information availability (Howard and Matheson 1984). More specifically, there is an arc from a random node r to a decision node d if and only if the value of r is observed at the time the decision d is to be made. The no-forgetting constraint is to capture the rationale that people do not destroy information on purpose; thus information available earlier should also be available later (Howard and Matheson 1984, Shachter 1986).

The primary reason for lifting the no-forgetting constraint is that it does not allow the representation of conditional independencies for decision nodes. However, there do exist cases where the decision maker, from her/his knowledge about the decision problem, is able to tell that a certain decision does not depend on certain pieces of available information. In our extended oil wildcatter problem, for instance, it is reasonable to assume that the decision `oil-sale-policy` is independent of `test`, `test-result`, and `drill` given `oil-produced`.

Sometimes independence assumptions for decision nodes are made for the sake of computational efficiency or even feasibility. In the domain of medical diagnosis and treatment, for instance, one usually needs to consider a number, say ten, of time points. To compute the diagnosis and treatment for the last time slice, one needs to consider all the previous nine time points. In the acute abdomen pain example studied by Provan and Clarke (1993), there are, for each decision node, 6 parent nodes that lie in the same time slice as the decision node. This means that the decision node at the last time slice has a total of 69 parents. In the simplest case of all variables being binary, we need to compute a decision table of 2^{69} entries; an impossible task. The same difficulty exists for planning under uncertainty (Dean and Wellman 1992). One way to overcome this difficulty is to approximate the decision problem by assuming that the decision in a time slice depends only on the previous, say one time slice, and is conditionally independent of all earlier time points. In this case, the decision table sizes are limited to $2^{13} = 8192$; still large but manageable.

Independence for decision nodes cannot be represented in influence diagrams. Going back to our extended oil wildcatter problem, even though we have made the assumption that `oil-sale-policy` is independent of `test`, `test-result`, and `drill` given `oil-produced`. But in Figure 1.4 there are still arcs from `test`, `test-result`, and `drill` to `oil-sale-policy`.

Following Smith (1988), this thesis reinterprets arcs into decision nodes as indication of both information availability and (potential) dependency. This new interpretation enables us to explicitly represent conditional independencies for decision nodes. To be more specific, the judgement that d is conditionally independent of r can be represented by simply not drawing an arc from r to d, even when the value of a random node r is observed at the time the decision d is to be made.

In our example, if we explicitly represent the assumption that `oil-sale-policy` is independent of `test`, `test-result`, and `drill` given `oil-produced`, then the decision network for the extend oil wildcatter problem becomes the one shown in Figure 1.5. We notice that there are no arcs from `test`, `test-result`, and `drill` to `oil-sale-policy`; the network is simpler than the one in Figure 1.4.

Figure 1.5: A decision network for the extended oil wildcatter problem, with independencies for the decision node `oil-sale-policy` explicitly represented.

Note that a user may be wrong in assuming that a decision is independent of a certain piece of information. To prevent such a case from happening, one can run the algorithm in Chapter 7 to graph-theoretically verify the user's independence judgements. If the algorithm is not able to verify, the user should be informed, and the user should abandon the independence assumption by adding an arc.

Another advantage of the new interpretation of arcs into decision nodes is that it provides uniform semantics to both arcs into decision nodes and arcs into random nodes; namely they both indicate dependence. This was first mentioned by Smith (1988).

It is evident that the no-forgetting constraint is not compatible with the new interpretation of arcs into decision nodes. It needs to be lifted.

Limited memory

Another reason for lifting the no-forgetting constraint is that the agent, say a robot, that executes decisions (actions) may have limited memory. There may be cases where the agent has only a few bits of memory. Even in the case when the agent has a fair amount of memory, it can not remember things forever. Because if so, the memory will run out sooner or later. Even if the agent has unlimited memory, remembering too much information would lead to inefficiency. We human being seem to remember only

Figure 1.6: A decision network for the extended oil wildcatter problem with multiple value nodes. The total utility is the sum of all the four value nodes.

important things.

1.5.2 Lifting the single value node constraint

As pointed out by Tatman and Shachter (1990) and by Shenoy (1992), the total utility of a decision problem can sometimes be decomposed into several components. In our extended oil wildcatter problem, for instance, `utility` can decomposed into the sum of four components, namely `test-cost`, `drill-cost`, `sale-cost`, and `oil-sales`. In such a case, we assign one value node for each component of the total utility, with the understanding that the total utility is the sum of all the value nodes. Figure 1.6 shows the resulting decision network after splitting the value node `utility` in Figure 1.4.

A major advantage of multiple value nodes over a single value node is that multiple value nodes may reveal independencies for decision nodes that are otherwise hidden. As the reader will see later in the thesis, there is a way for one to graph-theoretically tell that in Figure 1.6 `oil-sale-policy` is independent of `test`, `test-result`, and `drill` given `oil-produced`. The same can not be done for the network in Figure 1.4.

In the last subsection, we said that from her/his knowledge about the extended oil wildcatter problem, the decision maker may be able to say that `oil-sale-policy` is independent of `test`, `test-result`, and `drill` given `oil-produced`. Here we see that when multiple value nodes are introduced, those independencies can actually be read from the network itself, even if the decision maker fails to explicitly recognize them.

Independence for decision nodes and removable arcs

The next two paragraphs briefly revisit the third reason for lifting the no-forgetting constraint as listed in the synopsis. In Section 7.1, we shall formally define the concept of a decision node being independent of a certain parent and prove that when it is the case, the arc from that parent to the decision node is removable, in the sense that its removal does not affect the optimal expected value of the decision problem. It is a good idea to remove such arcs at a preprocessing stage, since it yields simpler diagrams. However, removing arcs from an influence diagram leads to the violation of the no-forgetting constraint.

Consider the no-forgetting decision network in Figure 1.6. Since from the network itself it can be determined that `oil-sale-policy` is independent of `test`, `test-result`, and `drill` given `oil-produced`, the arcs from `test`, `test-result`, and `drill` to `oil-sale-policy` are removable. Removing those arcs results in the network in Figure 1.7, which is no longer no-forgetting. This shows that in order to prune removable arcs from influence diagrams, we need to consider decision networks that do not satisfy the no-forgetting constraint.

1.5.3 Lifting the regularity constraint

The regularity constraint requires that there be a total ordering among the decision nodes. It is also called the single decision maker condition (Howard and Matheson 1984). When there are more than one decision maker who cooperate to achieve a common goal, the regularity constraint is no longer appropriate.

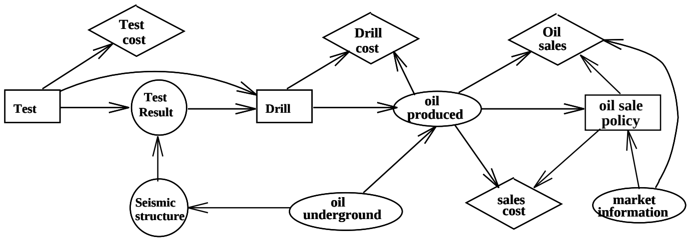

Figure 1.7: The decision network obtained from the one in Figure 1.6 by deleting some removable arcs. This network is no longer no-forgetting.

Consider further extending our oil wildcatter problem so that that there is not only oil but also natural gas. In this case, a `gas-sale-policy` also needs to be set. Suppose the company headquarter makes the `test` and `drill` decisions, while the oil department sets the `oil-sale-policy` and the gas department sets the `gas-sale-policy`. Then it is inappropriate to impose an order between `oil-sale-policy` and `gas-sale-policy`, since there is no reason why the gas department (or the oil department) should reach its decision earlier than the other department. A decision network for the further extended oil wildcatter problem is shown in Figure 1.8. We notice that there is no ordering between `oil-sale-policy` and `gas-sale-policy`.

Even in the case of one decision maker, the regularity constraint may be over-restrictive. From her/his knowledge and experience, the decision maker may be able to conclude that the ordering between two decision nodes is irrelevant; one has the same optimal expected value either way. In our further extended oil wildcatter problem, it may be reasonable to assume that it makes no difference whether `gas-sale-policy` or `oil-sale-policy` is set first.

Even when the ordering between two decision matters, the decision maker may not know the ordering beforehand. Suppose our oil wildcatter determine, on the first day a every month, the `gas-sale-policy` and `oil-sale-policy` for the coming month, based

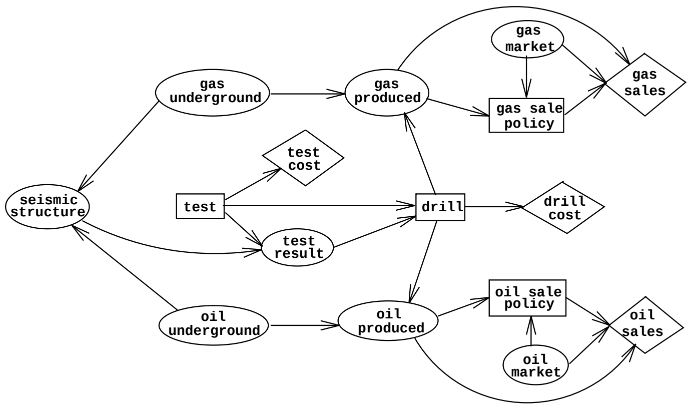

Figure 1.8: A decision network for the further extended oil wildcatter problem. It is not regular, "forgetting" and has more than one value node.

on the policies for the last month and market information. In this case, we are uncertain as to which one of those two decisions should be made first.

Let us now briefly revisit the second reason for lifting the no-forgetting constraint as listed in the synopsis. Together with the regularity constraint, the no-forgetting constraint says that information available when making an earlier decision should also be available when making a later decision. In the further extended oil wildcatter problem, we do not know before hand whether `oil-sale-policy` comes first or `gas-sale-policy` comes first. In such a case, the no-forgetting constraint can not be enforced. This is why we said in the synopsis that the existence of unordered decisions not only defeats the regularity constraint, but also the no-forgetting constraint.

1.6 Lifting constraints: Reasons pertaining to MDP's

Like decision analysis, finite stage Markov decision processes (MDP) are also a model for applying Bayesian decision theory to solve multiple-decision problems. Recent research has shown application promise for a combination of MDP's and influence diagrams in the form of temporal influence diagrams in planning under uncertainty (Dean and Wellman 1991) and in diagnosis and treatment/repair (Provan and Clarke 1993). One goal of this thesis is to provide a common framework for both of finite stage MDP's and influence diagrams. Doing so necessitates the lifting of the no-forgetting and the single value node constraint.

1.6.1 Finite stage MDP's

This subsection briefly reviews finite stage MDP's; and the next subsection will explain why it is necessary to lift the two constraints.

Finite stage MDP's are a model for sequential decision making (Puterman 1990, Denardo 1982, Bertsekas 1976). The model has to do with controlling a dynamic system over a finite number of time periods. There is a finite set T of time points. At time t, if the decision maker observes the system in state $s_t \in S_t$, s/he must choose an action, d_t, from a set of allowable actions at time t, Ω_{d_t} [3]. This choice may also depend all the previous states of the system. There are two consequences of choosing the action d_t when the system is in state s_t; the decision maker receives an immediate reward $v_t(s_t, d_t)$ and the probability distribution $P(s_{t+1}|s_t, d_t)$ for the state of the system at the next stage is determined. The collection $(T, S_t, d_t, \{P(s_{t+1}|s_t, d_t)\}, v_t(s_t, d_t))$ is called a *finite stage Markov decision process* (Puterman 1990). The problem is how to make the choice d_t at each time point t so as to maximize the decision maker's total expected reward. The function which makes this choice is called *decision rule* and a sequence of decision rules is called a *policy*.

A classic example of finite stage MDP is the problem of inventory control. Consider a ski retailer (Denardo 1982). From September to February, he makes an order from the wholesaler at the first day of the month. The amount of the order depends on his

[3] In general, Ω_{d_t} can vary according to s_t. Here we assume it does not.

current stock. His stock at the beginning of next month depends probabilistically on his current stock and how large the order is. This conditional (transition) probability can be estimated since the number of customers who arrive at a service facility during a period has, typically, a Poisson distribution. The profits our retailer makes during each month is computed from the number of pairs of skis sold and the difference between the wholesale and retail prices.

The standard way to find optimal decisions in a finite stage Markov decision process is by means of dynamic programming. In this approach, one begins with the last period and works backward to the first period. An optimal policy for the last period is found by maximizing the reward for that period. Then the whole last period is replaced by one value node, which is counted as reward in the next last period. This results in a finite stage MDP with one less period. One keeps repeating the procedure on the new process, till all the periods have been accounted for. This is very similar to the folding-back strategy for evaluating decision trees.

For the above model, one can show that an optimal decision rule depends only on the state s_t of the dynamic system at time t and is independent of the previous states and decisions.

1.6.2 Representing finite stage MDP's

This thesis achieves a common framework for decision analysis and finite stage MDP's by representing the MDP's as decision networks.

Since we have reinterpreted arcs into decision nodes as indications of both information availability and potential dependency, finite stage MDP's can be naturally represented as decision networks. Figure 1.9 (1) depicts a three stage MDP in the graph-theoretical language of decision networks. We notice that there are no arcs from s_1 and d_1 to d_2 even though s_1 and d_1 will be observed at the time the decision d_2 is to be made. The

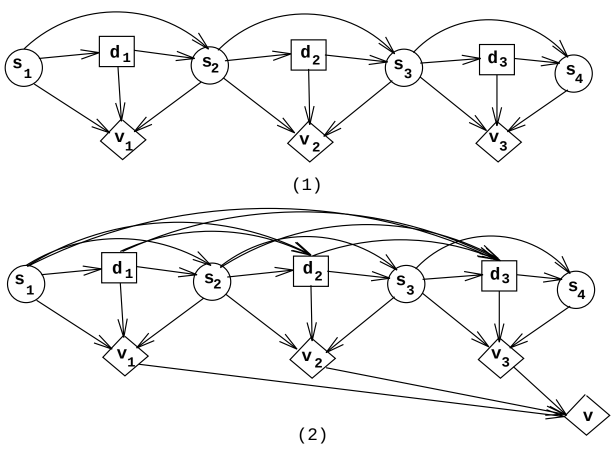

Figure 1.9: A three period finite stage MDP.

reason is that the optimal decision rule for d_2 is independent of s_1 and d_1 given s_2.

However if we insist, as in influence diagrams, on interpreting arcs into decision nodes as indications of only information availability, then it is cumbersome to represent finite stage MDP's. Figure 1.9 (2) depicts the influence diagram that represents the three stage MDP (Tatman and Shachter 1990). One can see that there is a number of extra no-forgetting arcs, namely arcs from s_1 and d_1 to d_2 and d_3, and from s_2 and d_2 to d_3. The presence of those arcs not only complicates the network, but also fails to reflect one important conclusion of MDP, namely that the current decision is independent of previous states and decisions given the current state.

Tatman and Shachter's algorithm is able to detect that d_2 does not depend on s_1 and d_1, and that d_3 does not depend on s_1, d_1, s_2, and d_2. So, the extra no-forgetting arcs makes no difference to the decision problem after all. They were introduced only because there was no concept of a decision network that does not satisfy the no-forgetting constraint.

In a finite stage MDP, there is a reward in each period. This can be naturally represented by assigning one value node for each period, as shown in Figure 1.9 (1). Note that s_3 separates the last period from all the previous periods. If we insist, as in influence diagrams, on the single value node constraint, then we need to connect v_1, v_2, and v_3 into a "super node" (Tatman and Shachter 1990), as shown in Figure 1.9 (2). One notices that no longer s_3 separates the last period from all the previous periods. This is another reason for lifting the single value node constraint.

1.7 Computational merits

The lifting of the no-forgetting, regularity, and single value node constraints allows us to discover stepwise-decomposable decision networks (SDDN). SDDN's are more general than both influence diagrams and finite stage MDP's. Moreover when evaluating SDDN's we can prune removable arcs, while the same cannot be done when evaluating influence diagrams since pruning arcs leads to the violation of the no-forgetting constraint. To put it more abstractly, SDDN's relax constraints imposed by influence diagrams and thus allow us to apply more techniques in solving a problem, and hence to solve the problem more efficiently. See Sections 6.5 and 9.6.1.

1.8 Why not lifted earlier

Howard and Matheson (1984) have hinted that in the case of multiple decision makers, the regularity and no-forgetting constraints may be violated. Smith (1987) has also mentioned that it is possible that a decision maker may choose or be compelled to "forget". Yet, no one before has studied decision networks that are not regular and/or are "forgetting". Why?

Howard and Matheson (1984) deal only with regular and no-forgetting decision networks (influence diagrams), because for evaluation, decision networks are first transformed into decision trees, and the transformation is possible only for regular no-forgetting decision networks. Even though new algorithms for evaluating influence diagrams have been developed after Howard and Matheson (1984) (see, for example, Shachter 1986), the correctness of all those algorithms relies on the regularity and no-forgetting constraints. This is probably why those constraints have always been imposed on influence diagrams. In this thesis, we shall show that one can evaluate a decision network, even if it is not regular and no-forgetting. This opens up the possibility of working with general decision networks.

1.9 Subclasses of decision networks

The lifting of the no-forgetting, the regularity, and the single value node constraints from influence diagrams leaves us only with the acyclicity and no-children-to-value-node constraints. In Chapter 2, we shall argue that those two constraints are fundamental and can not be lifted.

The acyclicity and no-children-to-value-node constraints define the concept of decision network. This section previews subclasses of decision networks we will encounter in this thesis.

Influence diagrams and finite stage MDP's are two existing subclasses of decision networks, which have been studied for many years. It is known that both of those subclasses of decision networks are stepwise-solvable, i.e they can be evaluated by considering one decision node at a time.

The most important subclass of decision networks introduced in this thesis is stepwise-decomposable decision networks (SDDN). They include both influence diagrams and

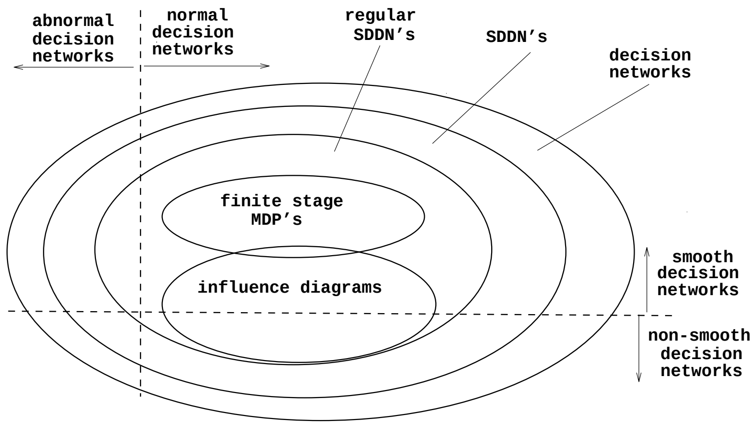

Figure 1.10: Subclasses of decision networks.

finite stage MDP's as special cases. See Figure 1.10. SDDN's are also stepwise-solvable. As a matter of fact, regular SDDN's[4] are the subclass of decision networks that can be evaluated by conventional dynamic programming (Denardo 1982, Chapter 9), and SDDN's in general constitute the subclass of decision networks that can be evaluated by non-serial dynamic programming (Bertelè and Brioshi 1972, Chapter 9).

The decision networks that are not stepwise-decomposable can be of various degrees of decomposability. To evaluate them, one needs to simultaneously consider two or more decision nodes. The number of decisions one need to consider simultaneously is determined by the degree by which the network is decomposable. The divide and conquer strategy spelled out in Chapter 4 can be utilized to explore the decomposability of a given decision network.

Smooth decision networks are introduced for technical convenience. They are conceptually simple and thus easy to manage. They are used extensively in this thesis to introduce new concepts and to prove theorems. Non-smooth decision networks can be

[4]To be more precise, the term decision network should be replace by the term decision network skeleton in this section.

be transformed into equivalent smooth decision networks when necessary.

Finally normal decision networks are introduced so that the equivalence between stepwise-decomposability and stepwise-solvability can be established. We conjecture that abnormal decision networks can be transformed into equivalent normal decision networks.

1.10 Who would be interested and why

Generally speaking, if you anticipate a solution to your problem by Bayesian decision theory, you should find this thesis interesting. Because it provides, in a sense, the most general framework — decision networks — for applying Bayesian decision theory. Problems representable as MDP's can be solved in (stepwise-decomposable) decision networks in the same way as before. Problems representable in influence diagrams can be solved in (stepwise-decomposable) decision networks at least as efficiently as, and usually more efficiently, than in influence diagrams. The reason for this efficiency improvement is that working with SDDN's relaxes the constraints imposed by influence diagrams, and allows one to apply more operations, such as pruning removable arcs, than previously allowed.

If you are a decision analyst, you might appreciate the ability of decision networks to represent independencies for decision nodes, to accommodate multiple cooperative decision makers, and to handle multiple value nodes. You might find it a relief that you do not have to completely order the decision nodes beforehand. Furthermore, you might appreciate the efficiency and other advantages of our algorithms.

If your problem falls into the category of MDP's, you might find the concept of decision networks helpful in assessing the transition probabilities and rewards. In the ski retailer problem (Section 1.6), many factors may affect the transition probabilities and rewards, for example deterioration of stock, delivery lag, payment upon delivery by the retailer and by customers, refusal to enter backlog by customers (Denardo 1982).

Within MDP, one needs to figure out the dynamic programming functional equation for each combination of the factors, which may be complicated. In decision networks, consideration of one more factor simply corresponds to the addition of one more node. This allows one to consider more factors than before. The representation advantage of decision networks may benefit control theory in general.

AI researchers who are concerned with planning, and diagnosis and treatment/repair should also find this thesis interesting.

Planning is a process of constructing courses of action in response to some objective. Since the planner might not have complete knowledge about the environment and about the effects of actions, planning are usually performed under uncertainty. Being a theory for rational choice of actions under uncertainty, Bayesian decision theory naturally comes into play. Preliminary research (Dean and Wellman 1992) has indicated that successful application of Bayesian decision theory in planning under uncertainty calls for a framework that combines characteristics of influence diagrams and and those of MDP's. Research on diagnosis and treatment (Provan and Clarke 1993) has pointed to the same direction. The concept of decision network introduced in this thesis may prove to be a good combination of influence diagrams and MDP's. Also, the ability of decision networks to represent conditional independencies for decision nodes may be computationally essential for those areas.

Chapter 2

Decision networks: the concept

This chapter introduces the concept of decision networks and addresses some of the foundational issues. Formal definitions will be provided in Chapter 3.

The concept of decision networks is intuitively illustrated through an example in section 2.1. Section 2.2 exposes the way by which other authors develop the concept of Bayesian networks from joint probabilities by means of the chain rule of probabilities, and by using the concept of conditional independencies. Section 2.3 derives the concept of decision network, through the concept of Bayesian networks, from the Bayesian decision theory setup by considering multiple decision problems. Section 2.4 discusses the fundamental constraints that decision networks need to satisfy and argues that decision networks are the most general representation framework for solving multiple-decision problems in Bayesian decision theory.

2.1 Decision networks intuitively

In this section, we illustrate the concept of decision networks through an example.

Decision networks can be understood at two levels: relation and number. At the level of relation, decision networks are directed graphs consisting of three types of nodes: decision nodes, random nodes and value nodes; and they are used to graphically represent the structures of decision problems. This directed graph is called a *decision network skeleton*. Consider the further extended oil wildcatter problem:

Figure 2.11: A decision network skeleton for the extended oil wildcatter problem.

> An oil wildcatter is deciding whether or not to drill in a new area. To aid his decision, he can order a seismic structure test. His decision about drill will depend on the test results if a test is ordered. If the oil wildcatter does decide to drill, crude oil and natural gas will be produced. Then, the oil wildcatter will decide his gas sale policy and oil sale policy on the basis of the quality and quantity of crude oil and natural gas produced, and on the basis of market information.

The structures of this decision problem can be represented by the decision network skeleton shown in Figure 2.11[1], where decision nodes are drawn as rectangles, random nodes as ovals, and value nodes as diamonds.

Briefly, here are the semantics of a decision network. Arcs into random nodes indicate probabilistic dependencies. A random node depends on all its parents, and is independent

[1]The figure is the same as Figure 1.8. The duplication is to save the reader from flipping back and forth.

of all its non-descendants given the values of its parents. In the extended oil wildcatter problem, `test-result`, for instance, probabilistically depends on `seismic-structure` and the decision to `test`, but is independent of `gas-underground` and `oil-underground` given `seismic-structure` and `test`.

Arcs into decision nodes indicate both information availabilities and functional dependencies. In our example, the arc from `oil-produced` to `oil-sale-policy` means that the oil wildcatter will have learned the quantity and quality of crude `oil-produced` when he decides his `oil-sale-policy`, and he thinks that the quantity and quality of `oil-produced` should affect his `oil-sale-policy`. There is no arc from `oil-underground` to `oil-sale-policy` because information about `oil-underground` is not directly available. There is no arc from `test-result` to `oil-sale-policy`, because the oil wildcatter figures that the information about the `test-result` should not affect his `oil-sale-policy` since that he will already have learned the quality and quantity of crude `oil-produced` at the time the policy is to be made.

Arcs into value nodes indicate functional dependencies. A value node is characterized by a function of its parents; the function take real number values, which represent the decision maker's utilities. In the extended oil wildcatter problem, `oil-sales` is a function of `oil-produced`, `oil-market` and `oil-sale-policy`. It depends on no other nodes. For each possible values of `oil-produced`, of `oil-market`, and of `oil-sale-policy`, the value of this function stands for the corresponding expected oil-sales. The total utility is the sum of all the value nodes; namely the sum of `test-cost`, `drill-cost`, `oil-sale` and `gas-sale`.

At the level of number, a decision network specifies a frame, i.e a set of possible values, for each variable. For example, the frame of `drill` my be {YES, NO}, and the frame of `oil-sales` may be the set of real numbers.

There is also a conditional probability for each random node given its parents and prior probability of each random node that does not have any parents. In our example, we need to specify, for instance, $P($`oil-underground`$)$, $P($`oil-produced|oil-underground`$)$, and etc ....

Further, we need to specify a utility function for each value node. In our example, the utility function for `oil-sales` is a real function of `oil-produced`, `oil-market`, and `oil-sale-policy`.

In summary, a decision network consist of (1) a skeleton which is an directed graph with three type of nodes, (2) a frame for each node, (3) a conditional probability for each random node, and (4) a utility or value function for each value node.

In a decision network, the decision about a decision node is made knowing the values of the parents of the node. Optimal decisions are decisions that maximize the expected total utility. The goals of decision analysis are to find the optimal decisions and to determine the optimal expected total utility.

2.1.1 A note

Note that the term "decision network" has been previously used in Hastings and Mello (1977). The meaning of the term in this thesis is different. In this thesis, the nodes in a decision network are variables, while nodes in a Hastings and Mello decision network are states, or values of variables. In a sense, one can say that we are working at a higher level of abstraction than Hastings and Mello. The relationship between our decision networks and Hastings and Mello's decision networks is the same as the relationship between influence diagrams and decision trees.

As observed by Smith *et al* (1993), influence diagrams gain much of their advantages over decision trees from the fact that they graphically capture conditional independencies at the level of relation (among variables). The same can be said for our decision networks and Hastings and Mello's decision networks. As the reader will see, the efficiency of our algorithms heavily depends on the fact that nodes in our decision networks are variables, instead of values of variables.

2.2 Bayesian networks

One way to understand decision networks is to think of them as developed from the standard Bayesian decision theory setup. We shall explain this in the next section. As a preparation, this section develops the concept of Bayesian networks from joint probabilities by means of the chain rule of probabilities and the concept of conditional independency (Howard and Matheson 1984, Pearl 1988).

Let X be a set of random variables. Let $P(X)$ be the joint probability of the variables in X. It is usually difficult, if possible at all, to assess the joint probability directly. One way to assess the joint probability indirectly is first to choose an ordering over the variable set X, say x_1, x_2, $\ldots$, x_n, then to expand the joint probability by the chain rule as follows:

$$P(x_1, x_2, \ldots, x_n) = P(x_1)P(x_2|x_1)\ldots P(x_n|x_1, \ldots, x_{n-1}). \tag{2.3}$$

We shall refer to the ordering as an *expansion ordering* . Because of equation (2.3), to assess the joint probability $P(X)$, it suffices to assess $P(x_i|x_1, \ldots, x_{i-1})$ for each $i \in \{1, \ldots, n\}$.

Often a decision maker is able to determine a proper subset π_{x_i} of $\{x_1, \ldots, x_{i-1}\}$ that are "directly related" to x_i such that other variables in $\{x_1, \ldots, x_{i-1}\}$ are only "indirectly related" to x_i via π_{x_i}. Translating into the language of the probability theory, this means that x_i is independent of other variables in $\{x_1, \ldots, x_{i-1}\}$ given π_{x_i}. Formally that is

$$P(x_i|x_1, \ldots, x_{i-1}) = P(x_i|\pi_{x_i}). \tag{2.4}$$

This equation further reduces the assessment task.

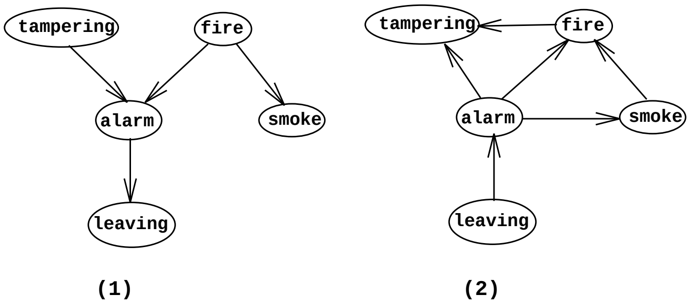

Figure 2.12: Two Bayesian networks for the joint probability $P(alarm, fire, tampering, smoke, leaving)$.

Given an expansion ordering $x_1, \ldots, x_n$ and the $\pi'_{x_i}s$, we construct a directed graph by the following rule:

For any x_i and x_j, draw an arc from x_i to x_j if and only if $x_i \in \pi_{x_j}$.

The acyclic directed graph such constructed, together with the conditional probabilities $P(x_i|\pi_{x_i})$, is called a *Bayesian network* for the joint probability $P(X)$.

As an example, consider the following decision scenario which is borrowed from (Poole and Neufeld 1991). The scenario involves five variables: `alarm`, `fire`, `tampering`, `smoke`, and `leaving`, denoting respectively the following propositions: the alarm is on, there is a fire, somebody is tampering; there is smoke and people are leaving. An expansion ordering for the joint probability $P(\texttt{alarm}, \texttt{fire}, \texttt{tampering}, \texttt{smoke}, \texttt{leaving})$ could be (`fire`, `tampering`, `alarm`, `leaving`, `smoke`). Suppose it is reasonable to set $\pi_{\texttt{tampering}} = \emptyset$, $\pi_{\texttt{alarm}} = \{\texttt{fire}, \texttt{tampering}\}$, $\pi_{\texttt{leaving}} = \{\texttt{alarm}\}$, and $\pi_{\texttt{smoke}} = \{\texttt{fire}\}$. Then we get the Bayesian network shown in Figure 2.12 (1). Another expansion ordering could be (`leaving`, `alarm`, `smoke`, `fire`, `tampering`). Suppose it is reasonable to set $\pi_{\texttt{alarm}} = \{\texttt{leaving}\}$, $\pi_{\texttt{smoke}} = \{\texttt{alarm}\}$, $\pi_{\texttt{fire}} = \{\texttt{alarm}, \texttt{smoke}\}$, and $\pi_{\texttt{tampering}} = \{\texttt{fire}, \texttt{alarm}\}$. Then we get the Bayesian network shown in Figure 2.12 (2). This network has more arcs than the one in (1).

How should one choose an expansion ordering? The answer provided by Howard and Matheson (1984) is that the ordering should be chosen such that the decision maker would feel natural and comfortable in assessing the π_{x_i}'s and the $P(x_i|\pi_{x_i})$'s. For example, it probably is easier to assess $P(\texttt{alarm}|\texttt{fire}, \texttt{tampering})$ than to assess $P(\texttt{tampering}|\texttt{fire}, \texttt{alarm})$. Smith (1989) says that one should choose the ordering to minimize the number of arcs in the resulting directed graph. In our example, the network in Figure 2.12 (1) is preferred to the network in (2). Pearl (1988, pp. 50-51) claims that when there are cause-effect relationships among the variables, the structure of a Bayesian network can be directly determined from the cause-effect relationships. For example, `tampering` and `fire` cause `alarm`, `fire` causes `smoke`, `alarm` causes `leaving`.

2.3 Decision networks

In this section, decision networks are developed as a way to represent of the knowledge (beliefs) and utilities that are needed in order to solve multiple-decision problems in Bayesian decision theory. Let us begin with a standard setup of Bayesian decision theory.

2.3.1 A general setup of Bayesian decision theory

Here is a setup of Bayesian decision theory (Gärdenfors *et al* 1988b, Fishburn 1988) that is more general than the one given in Section 1.2:

1. There is a set X of (random and decision) variables, which are relevant to a decision problem;

2. there is a set Δ of policies and for each possible policy $\delta \in \Delta$, there is a corresponding probability $P_\delta(X)$;

3. and there is a utility function $\mu(X)$, which specifies the decision maker's preferences about the possible states of affairs.

The problem is to decide on a policy δ^0 that maximizes the expected utility, that is

$$\sum_X P_{\delta^0}(X)\mu(X) = max_\delta\{\sum_X P_\delta(X)\mu(X)\}, \tag{2.5}$$

where $\sum_X$ means summation over all the possible values of X.

The setup given in Section 1.2 can be fitted into the setup given here by letting $X = \{o, s, d\}$, and for each policy $\delta : O \rightarrow \Omega_d$ setting

$$P_\delta(o, s, d) = \begin{cases} P(s)P(o|s) & \text{if } d = \delta(o) \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

In equation (2.5), summation is used instead of integration because we deal only with discrete variables in this thesis. However, most of our results can be easily extended to the case of continuous variables.

2.3.2 Multiple-decision problems

In applications, the decision maker usually needs to set the values for a number of variables $d_1, \ldots, d_k$. Let $OBS(d_i)$ denote the set of all the variables whose values will be observed by the decision maker at the time of decision d_i is to be made.

Sometimes, as in MDP's, the decision maker is able to qualitatively tell that some of those observed variables are irrelevant to d_i. On other occasions, the decision maker may be forced, for instance by computational complexity, to approximate the world by by making such irrelevance assumptions. Let $\pi^0_{d_i}$ be a subset of $OBS(d_i)$, such that the variables in $OBS(d_i) - \pi^0_{d_i}$ are, according to the decision maker, irrelevant to the decision d_i given $\pi^0_{d_i}$.

Before one can solve a problem, one needs first to clearly state the problem. The concept of multiple-decision problems is introduced as a way to pose a decision problem. A *multiple-decision problem* is a set $\mathcal{D} = \{< d_i, \pi^0_{d_i} > | 1 \leq i \leq k\}$, where the d_i's are decision variables and for each i, $\pi^0_{d_i}$ is the set of variables depending upon whose values the decision maker is to choose a value for d_i.

The further extended oil wildcatter problem (Figure 2.11) is a multiple-decision problem. The decision maker needs to decide on a value for each of the following variables: `test`, `drill`, `gas-sale-policy`, and `oil-sale-policy`. The π^0's are as follows: $\pi^0_{\texttt{test}} = \emptyset$, $\pi^0_{\texttt{drill}} = \{\texttt{test-result}\}$, $\pi^0_{\texttt{gas-sale-policy}} = \{\texttt{gas-produced, gas-market}\}$, and $\pi^0_{\texttt{oil-sale-policy}} = \{\texttt{oil-produced, oil-market}\}$.

Given a multiple-decision problem $\mathcal{D}$, define a partial ordering among its variables as follows: for any two variables x and y, we say that x *precedes* y if $x \in \pi^o_y$, or if there is another variable z such that $x \in \pi^o_z$ and z precedes y.

The fundamental constraint that a multi-decision problem must obey is the so-called *acyclicity constraint*, which require that there do not exist two variables x and y such that both x precedes y and y precedes x. The reason for this constraint is that the precedence relationship defined above implies time precedence. More explicitly, if x precedes y, then the value of x is observed or determined before the value of y.

2.3.3 Technical preparations

Given a multiple decision problem $\mathcal{D}$, let X be the set of all the variables in $\mathcal{D}$ and other variables that are relevant to the problem. For the further extended oil wildcatter problem, X also contains `oil-underground`, `gas-underground`, and `seismic-structure` in addition to the variables appeared in the problem statement, namely `test`, `drill`, `gas-sale-policy`, `oil-sale-policy`, `test-result`, `gas-produced`, `gas-market`, `oil-produced`, and `oil-market`.

For any variable $x \in X$, let Ω_x be the *frame* of x, i.e. the set of all possible values of x. For any subset $B \subseteq X$, let $\Omega_B = \prod_{x \in B} \Omega_x$.

To determine a value for d_i based on the values of the variables in $\pi^0_{d_i}$ is to choose a function $\delta_i : \Omega_{\pi^0_{d_i}} \rightarrow \Omega_{d_i}$. Such a function is called a *decision function (table)* for d_i. Let Δ_i denote the set of all the decision functions for d_i. The *policy space* is the Cartesian product $\Delta = \prod_{i=1}^{k} \Delta_i$. An element of Δ is called a *policy* .

2.3.4 Deriving the concept of decision networks

One needs to have the necessary knowledge to solve a problem. This subsection develops decision networks as a framework for specifying the knowledge (beliefs) and utilities that are required in order to solve a multiple-decision problem.

If the decision maker wants to solve a multiple-decision problem $\mathcal{D}$ in the setup given in Subsection 2.3.1, then s/he needs, according to the second item of the setup, to come up with a probability $P_\delta(X)$ for each policy δ. When obtained, P_δ would contain more information than is conveyed by $\mathcal{D}$ and δ. The portion of information conveyed by P_δ that is not conveyed by $\mathcal{D}$ and δ should originate from the decision maker's knowledge and beliefs about the uncertainties involved in the decision situation. Equipped with Bayesian networks, we are able to explicitly spell out this portion of information, as demonstrated in the following.

Assume $P_\delta(X)$ were somehow obtained. An expansion ordering for $P_\delta(X)$ *conforms to* $\mathcal{D}$ if for each d_i, variables in $\pi^0_{d_i}$ precede d_i in the ordering. One can easily verify that such an ordering is possible since $\mathcal{D}$ must be acyclic.

Given an expansion ordering ρ: $x_1, \ldots, x_n$ that conforms to $\mathcal{D}$, we could, as in the previous section, expand $P_\delta(X)$, determine the π_{x_i}'s, and construct a Bayesian network. Denoted by $\mathcal{N}_\delta$, this Bayesian network would contain the following information:

1. For each decision node d_i, the conditional probability $P_\delta(d_i|\pi_{d_i})$ and the fact that (Fact1:) d_i is independent of all the variables that come before d_i in ρ given the variables in π_{d_i}; and

2. for each random node c, the conditional probability $P_\delta(c|\pi_c)$ and the fact that (Fact2:) c is independent of all the variables that come before c in ρ given the variables in π_c.

Since π_{d_i} and $\pi^0_{d_i}$ have the same semantics, we have $\pi_{d_i}{=}\pi^0_{d_i}$. Hence Fact1 would have come from the problem statement $\mathcal{D}$; and the conditional probability $P_\delta(d_i|\pi_{d_i})$ would have come from the policy δ.

One the other hand, Fact2 and the conditional probability $P_\delta(c|\pi_c)$ do not follow from either $\mathcal{D}$ or δ, and hence must have come from the decision maker. They represent the decision maker's knowledge and beliefs about the uncertainties involved in the decision situation and need to be elicited before the decision problem $\mathcal{D}$ can be solved in Bayesian decision theory.

We now turn to utility. According to item 3 in the setup of Subsection 2.3.1, the decision maker needs also to express his preferences about the possible state of affairs by a utility function $\mu(X)$. $\mu(X)$ can sometimes be decomposed into the sum of a number of components, each of which depends only on a number of variables. Suppose $\mu(X)$ decomposes into m components $\mu_1(Z_1) + \ldots + \mu_m(Z_m)$, where Z_i is the set of variable of which μ_i depends upon. Introduce a value variable v_i for each μ_i, and attach v_i to the Bayesian network $\mathcal{N}_\delta$ by drawing arcs from each of the variables of Z_i to v_i. In the following, we shall write Z_i as π_{v_i}, and $\mu_i(Z_i)$ as $\mu_{v_i}(\pi_{v_i})$.

To summarize the discussions above and in Subsection 2.3.2, the decision maker needs to do the following in order to solve a multiple-decision problem in Bayesian decision theory:

1. specify the decision variables whose values are to be to determined, and the random variables and value variables that are related to those decision variables;
2. for each decision variable d_i, specify the set π_{d_i} of observed variables whose values are relevant to d_i,
3. determine an ordering ρ among all the variables such that ρ conforms to the problem statement $\{< d_i, \pi^0_{d_i} > |i\}$. Let $\rho[< x]$ denote the set of nodes that come before x in the ordering ρ.
4. for each random variable c, specify a subset π_c of $\rho[< c]$ such that c is $P(c|\rho[< c]) = P(c|\pi_c)$, and specify the conditional probability $P(c|\pi_c)$;
5. for each value variable v, specify the subset π_v of variables in $\rho[< v]$ that v depends upon and specify the utility function $\mu_{v_i}(\pi_{v_i})$.

We call the collection of all the information specified in items 1, 2, 4, and 5 a *decision network* . Thus, a decision network represents the decision maker's knowledge (beliefs) and preferences (utilities) that are needed in order to solve a multiple-decision problem in Bayesian decision theory. The ordering ρ is not included as part of the decision network because it can be arbitrary as long as it conforms to $\{< d_i, \pi^0_{d_i} > |i\}$.

Smith (1989) presents a way of developing the concept of influence diagrams (decision networks) in terms of the so-called third part semantics. In this section, the concept of decision networks has been developed directly from a standard setup of Bayesian decision theory without using the third part semantics.

2.3.5 An example

As an example, consider a decision scenario where a decision maker needs to decide whether to `bring-umbrella` in light of weather `forecast`. An additional variable, `rain`,

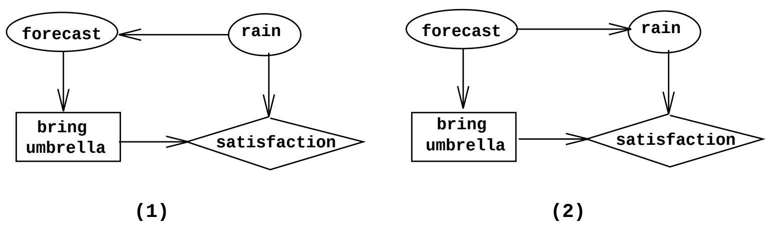

Figure 2.13: Two decision networks for the rain-and-umbrella problem.

which takes the value "yes" if it does turn out to rain and "no" otherwise, is believed to be relevant to the decision and hence is included in our analysis. For each decision function $\delta : \Omega_{\texttt{forecast}} \rightarrow \Omega_{\texttt{bring-umbrella}}$, the decision maker needs to come up with a joint probability $P_\delta(\texttt{rain}, \texttt{forecast}, \texttt{bring} - \texttt{umbrella})$. The expansion ordering `rain`, `forecast`, `bring-umbrella` conforms to the decision problem. If the decision maker's utility — `satisfaction` — is a function of `rain` and `bring-umbrella`, then the decision network is as shown in Figure 2.13 (1). To complete the specification of this network, one needs to assess the prior probability of `rain` and the conditional probability for `forecast` given `rain`. One also needs to assess the utility function.

The expansion ordering `forecast`, `rain`, `bring-umbrella` also conforms to the decision problem. It gives rise to the decision network shown on Figure 2.13 (2). The reader will see later that one can go between those two networks by reversing the arc between `forecast` and `rain` using Bayes' theorem (see Howard and Matheson 1984 and Section 5.5).

2.4 Fundamental constraints

In the introduction we have seen that among the five constraints that define influence diagrams, the regularity, the no-forgetting, and the single value node constraints should be lifted. This short section considers the remaining two constraints, namely the acyclicity and the no-children-to-value-node constraints.

In the derivation of decision networks in the previous section, we first had a Bayesian network $\mathcal{N}_\delta$ consisting of decision and random nodes. Then each value node v was attached to $\mathcal{N}_\delta$ by drawing arc from those nodes in $\mathcal{N}_\delta$ that v depends upon. Thus, the value nodes do not have children. In other words, decision networks always satisfy the no-children-to-value-node constraint.

There is one issue that needs to be addressed. In the last section, we have assumed the set of value nodes does not intersect with the set of random and decision nodes. This may not be the case sometimes; there may be nodes that are value nodes and decision or random nodes at the same time. For example, the amount of money x one spends the next month is a value variable. In the meantime, x is also a decision variable, and it affects how much one will be willing to spend the month after. In such a case, we will have two copies of x: one copy x_d functions as a decision node, while the other x_v functions as a value node. Since x_d is a decision node, one can set its value at his will and this value affects later decisions. On the other hand, the value node x_v depends on x_d and it does not affect any other nodes. By appropriately introducing copies of variables, we can always ensure that the set of value nodes does not intersect with the set of random and decision nodes.

We now turn to consider the acyclicity constraint. Decision networks must always be acyclic because multiple-decision problems are acyclic (subsection 2.3.2) and Bayesian networks acyclic. In the derivation of the last section, we began with a joint probability $P_\delta(X)$ which one must have in order to solve the multiple-decision problem $\mathcal{D}$ in Bayesian decision theory. Because $\mathcal{D}$ is acyclic, we were able to have an expansion ordering ρ for $P_\delta(X)$ that conforms to $\mathcal{D}$. The ordering ρ led to a Bayesian network $\mathcal{N}_\delta$. For any arc $x \rightarrow y$ in the Bayesian network, x comes earlier than y in the ordering ρ. Therefore $\mathcal{N}_\delta$ must be acyclic. A decision network was obtained from $\mathcal{N}_\delta$ by adding value nodes. Since the value nodes do not have any children, the decision network must also be acyclic.

The acyclicity and the no-children-to-value-node constraints are the only two constraints we impose on decision networks. We have just argued that those two constraints are fundamental and are indispensable to decision networks. In this sense, we say that decision networks are the most general representation framework for solving multiple-decision problems in Bayesian decision theory.

Chapter 3

Decision networks: formal definitions

The previous chapter has introduced the concept of decision networks. This chapter gives the exact definitions. We first formally define Bayesian networks (Section 3.1) and give two properties of Bayesian networks that will be useful later in a number of places (Section 3.2). Then we present the formal definitions of decision networks and of their evaluation (Section 3.3). A naive algorithm for evaluating decision networks is provided in Section 3.4. This algorithm is very inefficient because it simultaneously considers all the decision nodes. A decision network is stepwise-solvable if it can be evaluated by considering one decision node at a time (Section 3.5). Obviously, stepwise-solvability is a desirable computational property. In the next three chapters, we shall discuss when a decision network is stepwise-solvable and how to evaluate a stepwise-solvable decision network. For that purpose, we need the auxiliary concept of semi-decision networks (Section 3.6).

Starting from this chapter, we shall introduce various mathematical symbols. To help the reader to keep track of them, we have listed all the major symbols at the beginning of the thesis.

3.1 Formal definition of Bayesian networks

Before getting started, let us note that in this thesis, standard graph theory terms such as acyclic directed graphs, parents (direct predecessors), children (direct successors), predecessors, descendants (successors), leaves (nodes with no children), and roots (nodes

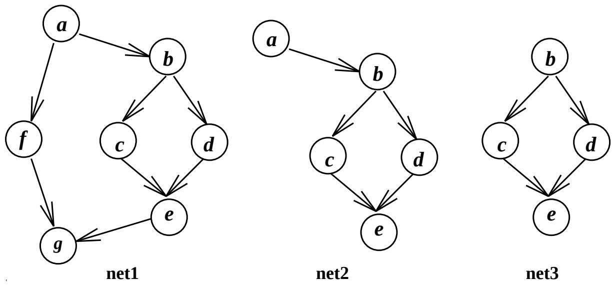

Figure 3.14: Bayesian network and irrelevant variables.

with no parents) will be used without giving the definitions. The reader is directed to Lauritzen *et al* (1990) for exact definitions. We shall use π_x to denote the set of parents of a node x in a directed graph.

A *Bayesian network*[1] (Pearl 1988) $\mathcal{N}$ is a triplet $\mathcal{N} = (X, A, \mathcal{P})$, where

1. X is a set of random nodes (variables); each $x \in X$ has a frame Ω_x — the set of possible values of x;
2. A is a set of arcs over X such that (X, A) is an acyclic directed graph; and
3. $\mathcal{P}$ is a set $\{P(x|\pi_x)|x \in X\}$[2] of conditional probabilities of the variables given their respective parents[3].

Figure 3.14 show a simple Bayesian network `net1` with seven variables a, b, c, d, e, f, and g. The network contains the following prior and conditional probabilities: $P(a)$, $P(f|a)$, $P(b|a)$, $P(c|b)$, $P(d|b)$, $P(e|c,d)$, and $P(g|f,e)$.

[1] Bayesian networks are also known as belief networks, Bayesian belief networks, and probabilistic influence diagrams.

[2] A conditional probability $P(x|\pi_x)$ is a mapping $P(x|\pi_x) : \Omega_{\{x\}\cup\pi_x} \rightarrow [0,1]$ such that $\sum_{\omega_x \in \Omega_x} P(x{=}\omega_x|\pi_x{=}\beta) = 1$ for each value β of π_x.

[3] When x is a root, π_x is empty. When it is the case, $P(x|\pi_x)$ stands for the prior probability of x.

The *prior joint probability* $P_{\mathcal{N}}(X)$[4] of a Bayesian network $\mathcal{N}=(X,A,\mathcal{P})$ is defined by

$$P_{\mathcal{N}}(X)=\prod_{x\in X}P(x|\pi_x). \tag{3.7}$$

In words, $P_{\mathcal{N}}(X)$ is the pointwise multiplication of all the conditional probabilities in $\mathcal{N}$. For any subset $B\subseteq X$, the *marginal probability* $P(B)$ is defined by

$$P(B)=\sum_{X-B}P(X), \tag{3.8}$$

where $\sum_{X-B}$ means summation over all the possible values of the variables in the set $X{-}B$.

A note about notation usage: In equation (3.7) the range of the multiplication is specified by the sign "$\in$"; $x{\in}X$ means x ranges over X. In equation (3.8) there is no "$\in$" sign. As a convention, we use $\sum_{X-B}P(X)$ as an abbreviation of $\sum_{\omega_{X-B}\in\Omega_{X-B}}P(\omega_B,\omega_{X-B})$, where ω_{X-B} stands for a general member of Ω_{X-B}, ω_B stands for a general member of Ω_B, and (ω_B,ω_{X-B}) is thus a general member of Ω_X. We shall always follow this convention about notation usage throughout this thesis.

For any two subsets $B_1,B_2\subseteq Y$ of variables, the *conditional probability* $P(B_1|B_2)$ is a function that satisfies

$$P(B_1{=}\beta_1,B_2{=}\beta_2)=P(B_2{=}\beta_2)P(B_1{=}\beta_1|B_2{=}\beta_2)\ \ \forall\beta_1{\in}\Omega_{B_1},\forall\beta_2{\in}\Omega_{B_2}. \tag{3.9}$$

For technical convenience, we also introduce the auxiliary concept of semi-Bayesian networks. A *semi-Bayesian network* is a Bayesian network except that the prior probabilities of some of the root nodes are missing. More precisely, a semi-Bayesian network is a quadruplet $\mathcal{N}=(X,A,\mathcal{P}|S)$, where (X,A) is a acyclic directed graph, $\mathcal{P}=\{P(x|\pi_x)|x{\in}X{-}S\}$ is set of conditional probabilities, and S is the set of root nodes whose prior probabilities are missing.

[4] A function from Ω_X to $[0,1]$.

It follows from the definition that Bayesian networks are semi-Bayesian networks.

As in Bayesian networks, we can define $P_{\mathcal{N}}(X)$ as follows,

$$P_{\mathcal{N}}(X) = \prod_{x \in (X-S)} P(x|\pi_x). \tag{3.10}$$

Unlike in Bayesian networks, here $P_{\mathcal{N}}(X)$ usually is not a probability; it may not sum to one. Thus, it is called the *prior joint potential* instead of the prior joint probability. Marginal and conditional potentials can be defined from the joint potential in the same way as marginal probabilities are defined from joint probabilities.

Note that since there are no arcs from $X-S$ to S, the prior joint potential $P_{\mathcal{N}}(X)$ is nothing but the conditional probability of the variables in $X-S$ given variables in S. For example, `net3` in Figure 3.14 is a not Bayesian network if we have only $P(c|b)$, $P(d|b)$, and $P(e|c,d)$ but not $P(b)$. In this case, `net3` is a semi-Bayesian network. The multiplication of all the conditional probabilities yields the conditional probability $P(c,d,e|b)$.

3.2 Variables irrelevant to a query

Given a (semi-)Bayesian network $\mathcal{N}$, one can pose a query $?P_{\mathcal{N}}(B_1|B_2)$. It is often possible to graphically identify certain variables being irrelevant to the query $?P_{\mathcal{N}}(B_1|B_2)$. This issue is addressed in Geiger *et al* (1990), Lauritzen *et al* (1990), and Baker and Boult (1990). The materials in the reminder of this section are extracted from those papers.

To *remove* a node x from a semi-Bayesian network $\mathcal{N} = (X, A, \mathcal{P}|S)$ is to: (1) remove x from X, (2) remove from A all the arcs that contain x, (3) remove from $\mathcal{P}$ all the items that involve x, and (4) those nodes that were not roots and become roots because of the removal are added to S.

We notice that removing a node from a Bayesian network may create root nodes which do not have prior probabilities. This is why we need the concept of semi-Bayesian network.

A leaf node is *barren* w.r.t a query $?P_{\mathcal{N}}(B_1|B_2)$, if it is not in $B_1 \cup B_2$. In `net1` (Figure 3.14), g is barren w.r.t $?P_{net1}(e|b)$. The following proposition says that if a leaf node is barren w.r.t a query, then it is irrelevant to the query, and hence can be harmlessly removed.

Lemma 3.1 *Suppose $\mathcal{N}$ is a semi-Bayesian network, and x is a leaf node. Let $\mathcal{N}'$ be the semi-Bayesian network obtained from $\mathcal{N}$ by removing x. If x is barren w.r.t to the query $?P_{\mathcal{N}}(B_1|B_2)$, then*

$$P_{\mathcal{N}}(B_1|B_2) = P_{\mathcal{N}'}(B_1|B_2). \tag{3.11}$$

Consider computing $P_{net1}(e|b)$. The node g is barren w.r.t the query and hence irrelevant. According to Lemma 3.1, g can be harmlessly removed. This creates a new barren node f. After the removal of g and f, `net1` becomes `net2`. Thus the query $?P_{net1}(e|b)$ is reduced to the query $?P_{net2}(e|b = b_0)$.

Let $An(B_1 \cup B_2)$ be the *ancestral set* of $B_1 \cup B_2$, i.e the set of nodes in $B_1 \cup B_2$ and the ancestors of those nodes. By repeatedly applying Lemma 3.1, we get

Proposition 3.1 *All the nodes outside $An(B_1 \cup B_2)$ are irrelevant to the query $?P(B_1|B_2)$.*

Let $G = (X, A)$ be a directed graph. An arc from x to y is written as an ordered pair (x, y). The *moral graph* $m(G)$ of G is an undirected graph $m(G) = (X, E)$ whose edge set E is given by

$$E = \{\{x, y\} | (x, y) \text{ or } (y, x) \in A, \text{ or } \exists z \text{ such that } (x, z) \text{ and } (y, z) \in A\}.$$

In words, $\{x, y\}$ is an edge in the moral graph if either there is an arc between the two vertices or they share a common child. The term moral graph was chosen because two nodes with a common child are "married" into an edge (Lauritzen and Spiegehalter 1988).

In an undirected graph, two nodes x and y are *separated* by a set of nodes S if every path connecting them contains at least one node in S. In a directed graph G, x and y are *m-separated* by S if they are separated by S in the moral graph $m(G)$[5]. Note that any node set separates itself from any other set.

Proposition 3.2 *Suppose $\mathcal{N}$ is a semi-Bayesian network. Let $\mathcal{N}'$ be the semi-Bayesian network obtained from $\mathcal{N}$ by removing all the nodes that are not in B_2 and are m-separated from B_1 by B_2. Then*

$$P_{\mathcal{N}}(B_1|B_2) = P_{\mathcal{N}'}(B_1|B_2). \tag{3.12}$$

In our example, since a is m-separated from e by b in `net2`, the query can be further reduced to $P_{net3}(e|b)$. Note that a is not m-separated from e by b in `net1`.

It can be proved (Lauritzen *et al* 1990 and Geiger *et al* 1990) that all the nodes irrelevant to a query `?`$P_{\mathcal{N}}(B_1|B_2)$ can be recognized and removed by applying Proposition 3.1 and Proposition 3.2.

3.3 Formal definitions of decision networks

A *decision network skeleton* is an acyclic directed graph $\mathcal{K} = (Y, A)$, which consists of three types of nodes: *random nodes*, *decision nodes*, and *values nodes*; and where the value nodes have no children.

A decision network skeleton describes a decision problem at the level of relation. It contains the set of parents π_d for each decision node d, the set of parents π_c for each random nodes, and the set of parents π_v for each value nodes. See Subsection 2.3.4.

A *decision network* $\mathcal{N}$ is a quadruplet $\mathcal{N}$=$(Y, A, \mathcal{P}, \mathcal{F})$ where

[5]To relate m-separation to d-separation (Pearl 1988), Lauritzen *et al* (1990) have shown that S d-separates B_1 and B_2 if and only if S m-separates B_1 and B_2 in the ancestral set $An(B_1 \cup S \cup B_2)$.

1. (Y, A) is a decision network skeleton. Let us use C to denote the set of random nodes, D to denote the set of decision nodes, and V to denote the set of value nodes.

2. Each $y{\in}Y$ has a frame Ω_y — the set of possible values of y.

3. $\mathcal{P}$ is a set $\{P(c|\pi_c)|c{\in}C\}$ of conditional probabilities of the random nodes given their respective parents.

4. $\mathcal{F}$ is a set $\{\mu_v : \Omega_{\pi_v}{\rightarrow}R^1|v{\in}V\}$ of *value (utility) functions* for the value nodes, where R^1 stands for the real line.

A decision network is obtained from a decision network skeleton by providing numerical information, i.e by specifying a frame for each variable, providing a conditional probability of each random node, and a value function for each value node. We say that $(Y, A, \mathcal{P}, \mathcal{F})$ is a *decision network over the skeleton* (Y, A) , and that (Y, A) is the *skeleton of the decision network* $(Y, A, \mathcal{P}, \mathcal{F})$.

A *decision function (table)* for a decision node d_i is a mapping $\delta_i : \Omega_{\pi_{d_i}} \rightarrow \Omega_{d_i}$. The *decision function space* Δ_i for d_i is the set of all the decision functions for d_i. Let $D = \{d_1, \ldots, d_k\}$ be the set of all the decision nodes. The Cartesian product $\Delta = \prod_{i=1}^{k} \Delta_i$ is called the *policy space* for $\mathcal{N}$, and a member $\delta{=}(\delta_1, \ldots, d_k) \in \Delta$ is called a *policy* .

Note that while a decision function δ_i is a function, a policy δ is a vector of decision functions.

The relationship between a decision node d_i and its parents π_{d_i} as indicated by a decision function $\delta_i : \Omega_{\pi_{d_i}} \rightarrow \Omega_{d_i}$ is equivalent to the relationship as represented by the conditional probability $P_{\delta_i}(d_i|\pi_{d_i})$ given by

$$P_{\delta_i}(d_i{=}\alpha|\pi_{d_i}{=}\beta) = \begin{cases} 1 & \text{if } \delta_i(\beta){=}\alpha \\ 0 & \text{otherwise,} \end{cases} \tag{3.13}$$

for all $\alpha \in \Omega_{d_i}$ and $\beta \in \Omega_{\pi_{d_i}}$.

Since $\delta = (\delta_1, \ldots, \delta_k)$, we sometimes write $P_\delta(d_i|\pi_{d_i})$ for $P_{\delta_i}(d_i|\pi_{d_i})$. Because of equation (3.13), we will abuse the symbol δ by letting it also denote the set $\{P_\delta(d_i|\pi_{d_i})|d_i \in D\}$ of conditional probabilities of the decision nodes.

In a decision network $\mathcal{N} = (Y, A, \mathcal{P}, \mathcal{F})$, let $X = C \cup D$. Let A_X be the set of all the arcs of A that lie completely in X. Then the triplet $(X, A_X, \mathcal{P} \cup \delta)$ is a Bayesian network, where δ denotes a set of conditional probabilities of the decision nodes. We shall refer to this Bayesian network the *Bayesian network induced from $\mathcal{N}$ by the policy δ*, and write it as $\mathcal{N}_\delta$. The prior joint probability $P_\delta(X)$ of $\mathcal{N}_\delta$ is given by

$$P_\delta(X) = \prod_{x \in C} P(x|\pi_x) \prod_{x \in D} P_\delta(x|\pi_x). \tag{3.14}$$

We shall refer to $P_\delta(X)$ as the *joint probability over X induced by δ* .

Because the value nodes do not have children, for any value node v, π_v contains no value nodes. Hence $\pi_v \subseteq X$. The expectation $E_\delta[v]$ of the value function $\mu_v(\pi_v)$ of v under P_δ is given by

$$\begin{aligned} E_\delta[v] &= \sum_X P_\delta(X)\mu_v(\pi_v) \\ &= \sum_{\pi_v} P_\delta(\pi_v)\mu_v(\pi_v). \end{aligned} \tag{3.15}$$

The *expected value* $E_\delta[\mathcal{N}]$ of $\mathcal{N}$ under the policy δ is defined by

$$E_\delta[\mathcal{N}] = \sum_{v \in V} E_\delta[v] \tag{3.16}$$

$$= \sum_X P_\delta(X) \sum_{v \in V} \mu_v(\pi_v). \tag{3.17}$$

Let us point it out again that $\sum_{\pi_v}$ and $\sum_X$ mean summation over all the possible values of π_v and X respectively, while $\sum_{v \in V}$ means summation over the set V. See Section 3.1 for a note about notation usage.

The *optimal expected value* $E[\mathcal{N}]$ of $\mathcal{N}$ is defined by

$$E[\mathcal{N}] = max_{\delta\in\Delta} E_{\delta}[\mathcal{N}]. \tag{3.18}$$

The optimal value of a decision network that does not have any value nodes is zero. An *optimal policy* δ^o=$(\delta_1^o, \ldots, \delta_k^o)$ is one that satisfies

$$E_{\delta^o}[\mathcal{N}] = E[\mathcal{N}]. \tag{3.19}$$

We call δ_i^o an *optimal decision function (table)* of d_i. For a decision network that does not have any value nodes, all policies are optimal.

In this thesis, we shall only consider variables with finite frames. Hence there are only finitely many possible policies. Consequently, there always exists at least one optimal policy.

To evaluate a decision network is to

1. find an optimal policy, and

2. find the optimal expected value.

3.4 A naive algorithm

A straightforward approach to the evaluation of decision networks is to simply follow the definitions of optimal policy and of optimal expected value, and exhaustively search through the policy space Δ. This idea is made explicit by the following algorithm.

Procedure NAIVE-EVALUATE:

- Input: $\mathcal{N}$ — a decision network.
- Output: An optimal policy and the optimal expected value of $\mathcal{N}$.

Let Δ be the policy space of $\mathcal{N}$.

1. Pick one policy from Δ and denoted it by δ^o. Set $\Delta = \Delta - \{\delta^o\}$.
2. Compute $E_{\delta^o}[\mathcal{N}]$.
3. **While** $\Delta \neq \emptyset$, **do**
   - Pick one policy from Δ and denoted if by δ. Set $\Delta = \Delta - \{\delta\}$.
   - Compute $E_{\delta}[\mathcal{N}]$.
   - **If** $E_{\delta}[\mathcal{N}] > E_{\delta^o}[\mathcal{N}]$, set $\delta^o = \delta$.

   end-while
4. Output δ^o and $E_{\delta^o}[\mathcal{N}]$.

Though simple, this naive algorithm is very inefficient. The main reason is that it simultaneously considers all the decision nodes. This results in an exhaustive search through the policy space Δ, which can be computationally prohibitive. Suppose there are k decision nodes, each has l parents, and suppose all the variables are binary. Then for each decision node d, the cardinality of Ω_{π_d} is 2^l; hence there are $2^{(2^l)}$ possible decision functions for d. Consequently there are $(2^{(2^l)})^k$ policies in Δ. The procedure NAIVE-EVALUATE computes the expected value of $\mathcal{N}$ for each of the $(2^{(2^l)})^k$ policies!

There are decision networks whose evaluation necessitates simultaneous consideration of all the decision nodes. As an example, consider the decision network (skeleton) in Figure 3.15. Enemy movements may be observed by both agent1 and agent2. An agent decides whether or not to report enemy movements according to the instructions established beforehand by the intelligence office. If an agent reports, there is a chance that s/he may be exposed.

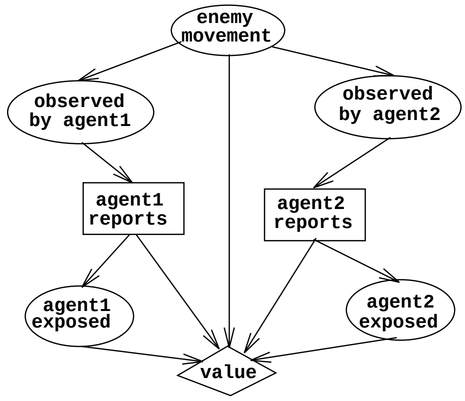

Figure 3.15: A decision network whose evaluation may require simultaneous consideration of all the decision nodes.

For the sake of illustration, assume all the variables either take the value `YES` or `NO`, except for the variable `value`, which takes real numbers. To complete the specification, we need to give the following (conditional) probabilities: P(`enemy-movement`), P(`observed-by-agent1|enemy-movement`), P(`observed-by-agent2|enemy-movement`), P(`agent1-exposed|agent1-reports`), and P(`agent2-exposed|agent2-reports`). We need also to give value function μ_{value}(`enemy-movement`, `agent1-exposed`, `agent2-exposed`).

There are four possible instructions (decision functions) for agent1:

1. If `observed-by-agent1`=YES, then `agent1-reports`=YES;
   If `observed-by-agent1`=NO, then `agent1-reports`=YES.

2. If `observed-by-agent1`=YES, then `agent1-reports`=YES;
   If `observed-by-agent1`=NO, then `agent1-reports`=NO.

3. If `observed-by-agent1`=YES, then `agent1-reports`=NO;
   If `observed-by-agent1`=NO, then `agent1-reports`=YES.

4. If `observed-by-agent1`=YES, then `agent1-reports`=NO;

   If `observed-by-agent1`=NO, then `agent1-reports`=NO.

Similarly, there are four possible instructions for agent2. The policies (instructions for both agents) for the problem are given as follows:

1. If `observed-by-agent1`=YES, then `agent1-reports`=YES;

   If `observed-by-agent1`=NO, then `agent1-reports`=YES.

   and

   If `observed-by-agent2`=YES, then `agent2-reports`=YES;

   If `observed-by-agent2`=NO, then `agent2-reports`=YES.

2. If `observed-by-agent1`=YES, then `agent1-reports`=YES;

   If `observed-by-agent1`=NO, then `agent1-reports`=NO.

   and

   If `observed-by-agent2`=YES, then `agent2-reports`=YES;

   If `observed-by-agent2`=NO, then `agent2-reports`=YES.

3. and so on . . .

One can easily see that the policy space consist of $(2^{(2^1)})^2 = 4 * 4 = 16$ possible policies.

In assessing his utilities, the intelligence office needs to keep both agents in mind. For example, it may be the case that information about an particular enemy movement is important enought to risk one agent but not both. The instructions for such a situation should allow one and only one agent to report. The instructions can require, for instance, agent1 to report when the information is deemed important enough to risk one agent, and require agent2 to report only when the information is deemed important enough to risk both agents. Such instructions can be arrived at only by considering the two agents simultaneously. In Chapter 8, we shall formally prove that with appropriate probabilities and value functions, optimal policies for the decision network in Figure 3.15 can be found only by considering the two decisions at the same time.

On the other hand, however, there are decision networks which allow more efficient algorithms than NAIVE-EVALUATE. The best case is when a decision network can be evaluated by considering one decision node at a time. This leads to the concept of stepwise-solvability.

3.5 Stepwise-solvability

Let $\mathcal{N}$ be a decision network. Let d_1, d_2, …, d_k be all the decision nodes in $\mathcal{N}$, and let $\delta{=}(\delta_1, \delta_2, \ldots, \delta_k)$ be a policy of $\mathcal{N}$, where δ_j is a decision function of d_j. The expected value $E_\delta[\mathcal{N}] = E_{(\delta_1,\delta_2,\ldots,\delta_k)}[\mathcal{N}]$ is a function of δ_1, δ_2 …, and δ_k.

For any $i{\in}\{1, 2, \ldots, k\}$, if we fix the value of δ_j for all $j{\in}\{1, 2, \ldots, k\}$ such that $j{\neq}i$, then $E_{(\delta_1,\ldots,\delta_{i-1},\delta_i,\delta_{i+1},\ldots,\delta_k)}[\mathcal{N}]$ becomes a function of δ_i. Rank all the possible values of δ_i, i.e all the possible decision functions of d_i, according to the value $E_{(\delta_1,\ldots,\delta_{i-1},\delta_i,\delta_{i+1},\ldots,\delta_k)}[\mathcal{N}]$. If the decision function (for d_i) that is ranked the highest remains the same regardless of the values of the δ_j's ($j{\neq}i$), then we say that d_i is a *stepwise-solvability candidate node* , or simply an *SS candidate node* of $\mathcal{N}$.

A *deterministic node* is a random node whose conditional probability takes the value either 0 or 1. To *replace a decision node* d_j by a deterministic node characterized by a function $\delta_j : \Omega_{\pi_{d_j}} \rightarrow \Omega_d$ (Shachter 1988) is to replace d_j by a deterministic node with the same frame, and to set $P(d_j|\pi_{d_j})$ to be the conditional probability that represents δ_j in the sense of equation (3.13).

If d_i is an SS candidate node, then an optimal decision function δ_i^o can be found as follows. For all $j{\in}\{1, 2, \ldots, k\}$ such that $j{\neq}i$, replace the decision node d_j by a deterministic random node characterized by an arbitrary decision function δ_j, resulting in a decision network with only one decision node d_i. Then find a policy δ_i^o of d_i that satisfies

$$E_{(\delta_1,\ldots,\delta_{i-1},\delta_i^o,\delta_{i+1},\ldots,\delta_k)}[\mathcal{N}] = max_{\delta_i\in\Delta_i} E_{(\delta_1,\ldots,\delta_{i-1},\delta_i,\delta_{i+1},\ldots,\delta_k)}[\mathcal{N}]. \qquad (3.20)$$

Proposition 3.3 *If d_i is an SS candidate node of a decision network $\mathcal{N}$, then an decision function δ_i^o that satisfies equation (3.20) is an optimal decision function of d_i.*

Proof: Let $(\delta_1^*,\ldots,\delta_{i-1}^*,\delta_i^*,\delta_{i+1}^*,\ldots,\delta_k^*)$ be an optimal policy of $\mathcal{N}$. Since d_i is an SS candidate node and δ_i^o satisfies (3.20), we have

$$E_{(\delta_1^*,\ldots,\delta_{i-1}^*,\delta_i^o,\delta_{i+1}^*,\ldots,\delta_k^*)}[\mathcal{N}] \geq E_{(\delta_1^*,\ldots,\delta_{i-1}^*,\delta_i^*,\delta_{i+1}^*,\ldots,\delta_k^*)}[\mathcal{N}].$$

Therefore, $(\delta_1^*,\ldots,\delta_{i-1}^*,\delta_i^o,\delta_{i+1}^*,\ldots,\delta_k^*)$ must also be an optimal policy of $\mathcal{N}$. Consequently, δ_i^o must be an optimal decision function of d_i. The proposition is proved. □

A decision network is *stepwise-solvable* if it contains no decision nodes, or if

1. there exists an SS candidate node d_i such that
2. if d_i is replaced by a deterministic node characterized by an optimal decision function of d, the resulting decision network (with one less decision node) is stepwise-solvable.

A decision network skeleton is *stepwise-solvable* if all the decision networks over the skeleton are stepwise-solvable.

If a decision network $\mathcal{N}$ is stepwise-solvable, then it can be evaluated as follows. Find an SS candidate node d_i and find an optimal decision function δ_i^o of d_i in the way as specified in equation (3.20). Replace d_i by a deterministic node characterized by δ_i^o, resulting in another stepwise-solvable decision network $\mathcal{N}'$ with one less decision node.

Then recursively apply the procedure to $\mathcal{N}'$, an so on so forth. We see here that $\mathcal{N}$ is evaluated by considering one decision node at a time.

Suppose $\mathcal{N}$ is a stepwise-solvable decision network with k decision nodes. Suppose each decision node has l parents, and suppose all the variables are binary. Then to evaluate $\mathcal{N}$, we need to compute the expected values of $\mathcal{N}$ for $(2^{(2^l)}) * k$ policies, instead of $(2^{(2^l)})^k$ policies as in the case of NAIVE-EVALUATE.

Note that the aforementioned evaluation method is not the best. The term $2^{(2^l)}$ can easily be prohibitively large. We shall show that when a decision network is stepwise-solvable, it can be solved not only by considering one decision node at a time, by also by considering one, usually small, part of the network at a time. The complexity can be reduced to that of computing no more than $2k + m$ marginal probabilities of no more than $l + 1$ variables or computing no more than $(2k + m)2^{l+1}$ numbers, where m stands for the number of value nodes.

So, stepwise-solvability is a very desirable property for a decision network to possess. In the next three chapters, we shall investigate when a decision network is stepwise-solvable and what is the best way to evaluate a stepwise-solvable decision network. For this purpose, we need the technical concept of semi-decision network.

3.6 Semi-decision networks

The reader has seen that removing nodes from a Bayesian network may create root nodes which do not have prior probabilities. This is why we need the concept of semi-Bayesian network. We shall also be discussing removing nodes from decision networks, which necessitates the concept of semi-decision networks.

A *semi-decision network* is a decision network except that the prior probabilities of some of the root random nodes are missing. We use $\mathcal{N} = (Y, A, \mathcal{P}, \mathcal{F}|S)$ to denotes a semi-decision network, where S is the set of root random nodes whose prior probabilities are missing.

As before, let $X = C \cup D$ be the set of random and decision nodes. Similarly to the case of decision networks, a policy δ induces a semi-Bayesian network $(X, A_X, \mathcal{P}\cup\delta|S)$, which will be referred to as the *semi-Bayesian network induced from $\mathcal{N}$ by the policy δ* , and which will be written as $\mathcal{N}_\delta$. Let $P_\delta(X)$ be the prior joint potential of $\mathcal{N}_\delta$.

For any value node $v\in V$, $\pi_v \subseteq X$. The *expected value* $E_\delta[\mathcal{N}]$ of $\mathcal{N}$ under the joint potential $P_\delta(X)$ is defined by

$$E_\delta[\mathcal{N}] = \sum_X P_\delta(X) \sum_{v\in V} \mu_v(\pi_v). \tag{3.21}$$

The *optimal expected value* $E[\mathcal{N}]$ of $\mathcal{N}$ is defined by

$$E[\mathcal{N}] = max_{\delta\in\Delta} E_\delta[\mathcal{N}].$$

An *optimal policy* δ^o is one that satisfies

$$E_{\delta^o}[\mathcal{N}] = E[\mathcal{N}].$$

Unlike in the case of decision networks, we also define the concept of conditional expected value for semi-decision networks. Th *conditional expected value* $E_\delta[\mathcal{N}|S]$ of $\mathcal{N}$ given S is defined by

$$E_\delta[\mathcal{N}|S] = \sum_{X-S} P_\delta(X) \sum_{v\in V} \mu_v(\pi_v). \tag{3.22}$$

Obviously $E_\delta[\mathcal{N}|S]$ is a function of S.

We chose the term "conditional expected value" because that the prior joint potential $P_\mathcal{N}(X)$ is nothing but the conditional probability of the variables in $X-S$ given S (see the note at the end of Section 3.1) .

The *optimal conditional expected value* $E[\mathcal{N}|S]$ of $\mathcal{N}$ given S is defined by

$$E[\mathcal{N}|S] = max_{\delta\in\Delta}E_{\delta}[\mathcal{N}|S].$$

An *optimal conditional policy* δ^o is one that satisfies

$$E_{\delta^o}[\mathcal{N}|S] = E[\mathcal{N}|S],$$

for all possible values of S.

Proposition 3.4 *A conditionally optimal policy of a semi-decision network, if exits, is always an optimal policy.*

Proof: From equations (3.21) and (3.22), we have

$$E_{\delta}[\mathcal{N}] = \sum_S E_{\delta}[\mathcal{N}|S].$$

Let δ^o be a conditional optimal policy and let δ be an arbitrary policy of $\mathcal{N}$. Then

$$E_{\delta^o}[\mathcal{N}] = \sum_S E_{\delta^o}[\mathcal{N}|S] \geq \sum_S E_{\delta}[\mathcal{N}|S] = E_{\delta}[\mathcal{N}].$$

Therefore

$$E_{\delta^o}[\mathcal{N}] \geq max_{\delta\in\Delta}E_{\delta}[\mathcal{N}].$$

In words, δ^o is an optimal policy of $\mathcal{N}$. $\Box$

For a semi-decision network, there always exists at least one optimal policy. But there may not necessarily exist any conditional optimal policies.

Given a semi-decision network $\mathcal{N}$, if every optimal policy of $\mathcal{N}$ is also a conditionally optimal policy, then we say that $\mathcal{N}$ is *uniform* .

We shall investigate when a semi-decision network is uniform later.

Chapter 4

Divide and conquer in decision networks

This chapter and the next two chapters constitute the heart of this thesis; they introduce and study one subclass of decision networks, namely stepwise-decomposable decision networks (SDDN's). SDDN's are important because they are stepwise-solvable, and as we shall show in Chapter 8 stepwise-decomposability is the weakest graph-theoretical criterion that guarantees stepwise-solvability.

This chapter investigates when and how a decision network can be decomposed into two subnetworks such that the optimal expected value and an optimal policy of the decision network can be computed by evaluating the two subnetworks. The next two chapters are concerned with how to evaluate decision networks that can be decomposed into n — the number of decision nodes — subnetworks such that the optimal expected values and optimal policies of the decision networks can be computed by evaluating the n subnetworks.

The organization of this chapter is as follows. Section 4.1 discusses the relationship between independence in a decision network and separation in the underlying decision network skeleton. Section 4.2 defines a concept of decomposability for decision networks, and Section 4.3 shows that this concept of decomposability implies a divide and conquer evaluation strategy.

Since manipulation of decision networks gives rise to semi-decision networks and decision networks are special semi-decision networks, exposition in this chapter will be carried out in terms of semi-decision networks.

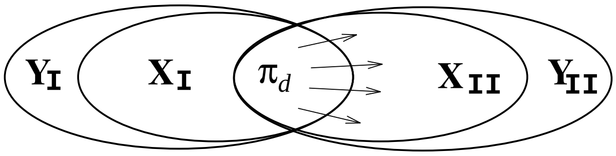

Figure 4.16: The relationships among the sets Y, Y_I, Y_{II}, X_I, X_{II}, and π_d. The three sets Y_I, Y_{II} and π_d constitute a partition of Y — the set of all the nodes; while X_I, X_{II} and π_d constitute a partion of $C \cup D$ — the set of all the random and decision nodes. When the network is smooth at d, there are no arcs going from X_{II} to π_d.

4.1 Separation and independence

The main goal of this chapter is to prove Theorem 4.1, one of the most important theorems in this thesis. In preparation, this section exposes the relationship between graph-theoretic separation and probabilistic independence in the context of decision networks.

Suppose $\mathcal{K} = (Y, A)$ is decision network skeleton and d is a decision node in $\mathcal{K}$. Let $Y_I(d, \mathcal{K})$, or simply Y_I be the set of all the nodes that are m-separated from d by π_d, with the nodes in π_d itself excluded. Let $Y_{II}(d, \mathcal{K})$, or simply Y_{II} be the set of all the nodes that are not m-separated from d by π_d. We observe that Y_I, π_d, and Y_{II} forms a partition of Y. Let $X_I(d, \mathcal{K}) = Y_I(d, \mathcal{K}) \cap (C \cup D)$ and $X_{II}(d, \mathcal{K}) = Y_{II}(d, \mathcal{K}) \cap (C \cup D)$.

The relationships among the sets are illustrated in Figure 4.16. In the following, we shall refer to Y_I as the *upstream set* of π_d, Y_{II} as the *downstream set* of π_d. We shall also refer to X_I as the *set of random and decision nodes in the upstream* of π_d, and X_{II} as the *set of random and decision nodes in the downstream* of π_d.

Consider a semi-decision network $\mathcal{N} = (Y, A, \mathcal{P}, \mathcal{F}|S)$. Let δ be a policy of $\mathcal{N}$. As pointed out by Lauritzen *et al* (1990), m-separation in the skeleton (Y, A) implies conditional independence for $P_\delta(X)$ — the joint potential over X induced by δ. Since π_d

m-separates X_I and X_{II}, we have that $P_\delta(X_{II}|X_I, \pi_d) = P_\delta(X_{II}|\pi_d)$. Therefore

$$\begin{aligned} P_\delta(X_I, \pi_d, X_{II}) &= P_\delta(X_I, \pi_d) P_\delta(X_{II}|X_I, \pi_d) \\ &= P_\delta(X_I, \pi_d) P_\delta(X_{II}|\pi_d). \end{aligned} \tag{4.23}$$

The rest of this section seeks an explicit representation of $P_\delta(X_I, \pi_d)$ and $P_\delta(X_{II}|\pi_d)$ in terms of conditional probabilities.

A decision network is *smooth* at the decision node d, if there are no arcs going from the downstream set Y_{II} of π_d to nodes in π_d. In other words, arcs between π_d and Y_{II} only go from π_d to Y_{II}.

As an example, consider the decision network skeleton for the further extended oil wildcatter problem (Figure 2.11). The downstream set of $\pi_{\texttt{oil-sale-policy}}$ consists of `oil-sale-policy` and `oil-sales`. There are no arcs from `oil-sales` to nodes in $\pi_{\texttt{oil-sale-policy}}$. So, the skeleton is smooth at `oil-sale-policy`. One the other hand, the downstream set of $\pi_{\texttt{drill}}$ contains all the nodes except `test-cost` and the nodes in $\pi_{\texttt{drill}}$. In particular, the downstream set contains the node `seismic-structure`. Because of the arc from `seismic-structure` to `test-result` ($\in \pi_{\texttt{drill}}$), the decision network skeleton is not smooth at `drill`.

Let $d_1, \ldots, d_j$ be all the decision nodes in $X_I \cup \pi_d$ and $d_{j+1}, \ldots, d_k$ be all the decision nodes in X_{II}. Note that $d \in \{d_{j+1}, \ldots, d_k\}$. For a policy $\delta = (\delta_1, \ldots, \delta_k)$, let $\delta_I = (\delta_1, \ldots, \delta_j)$ and $\delta_{II} = (\delta_{j+1}, \ldots, \delta_k)$.

Suppose the semi-decision network $\mathcal{N}$ is smooth at d. It follows from Proposition 3.1 that

$$P_\delta(X_I, \pi_d) = \prod_{x \in C \cap (X_I \cup \pi_d)} P(x|\pi_x) \prod_{i=1}^{j} P_{\delta_i}(d_i|\pi_{d_i}). \tag{4.24}$$

And it follows from Proposition 3.2 that

$$P_\delta(X_{II}|\pi_d) = \prod_{x \in C \cap X_{II}} P(x|\pi_x) \prod_{i=j+1}^{k} P_{\delta_i}(d_i|\pi_{d_i})). \tag{4.25}$$

Those two equations give us the following lemma.

Lemma 4.1 *If $\mathcal{N}$ is smooth at d, then $P_\delta(X_I, \pi_d)$ only depends on δ_I, and $P_\delta(X_{II}|\pi_d)$ only depends on δ_{II}. From now on, we shall write $P_\delta(X_I, \pi_d)$ as $P_{\delta_I}(X_I, \pi_d)$, and $P_\delta(X_{II}|\pi_d)$ as $P_{\delta_{II}}(X_{II}|\pi_d)$.*

4.2 Decomposability of decision networks

Also in preparation for Theorem 4.1, this section introduces a concept of decomposability for decision networks and shows how to divide a decomposable decision network into two subnetworks.

A decision network skeleton $\mathcal{K}=(Y, A)$ is *decomposable* at a decision node d if the number of decision node in $Y_{II}(d, \mathcal{K})$ is less than the number of decision nodes in $\mathcal{K}$. A semi-decision network is *decomposable* at a decision node d if the underlying skeleton is.

When a decision network skeleton $\mathcal{K}$ is decomposable and smooth at d, we define the *downstream component* of $\mathcal{K}$ w.r.t d, denoted by $\mathcal{K}_{II}(d, \mathcal{K})$ or simply $\mathcal{K}_{II}$, to be the decision network skeleton obtained by restricting $\mathcal{K}$ to $\pi_d \cup Y_{II}$ and then removing those arcs that connect two nodes in π_d.

Also, we define the *upstream component* of $\mathcal{K}$ w.r.t d, denoted by $\mathcal{K}_I(d, \mathcal{K})$ or simply K_I, to be the decision network skeleton obtained by restricting $\mathcal{K}$ to $Y_I \cup \pi_d$ and then adding a node u and drawing an arc from each node in π_d to u. The node u is to be used to store the value of the downstream component, and is thus called the *downstream-value node* .

Figure 4.17 shows the downstream and upstream components, w.r.t `oil-sale-policy`, of the decision network skeleton in Figure 2.11.

Note that while Y_I and Y_{II} are sets of nodes, $\mathcal{K}_I$ and $\mathcal{K}_{II}$ are decision network skeletons. $\mathcal{K}_I$ and $\mathcal{K}_{II}$ contain the nodes in π_d, while Y_I and Y_{II} do not.

Let $\mathcal{N}$ be a decision network over $\mathcal{K}$, and suppose $\mathcal{N}$ (or $\mathcal{K}$) is decomposable and smooth at d. The *downstream component* of $\mathcal{N}$ w.r.t d, denoted by $\mathcal{N}_{II}(d, \mathcal{N})$ or simply by $\mathcal{N}_{II}$, is a semi-decision network over $\mathcal{K}_{II}$. The value functions for all the value nodes of $\mathcal{N}_{II}$ remain the same at in $\mathcal{N}$. The conditional probabilities of the random nodes of $\mathcal{N}_{II}$ that lie outside π_d also remain the same as in $\mathcal{N}$. The nodes in π_d, random or decision, are viewed in $\mathcal{N}_{II}$ as random nodes whose prior probabilities are missing.

The *upstream component* of $\mathcal{N}$ w.r.t d, denoted by $\mathcal{N}_I(d, \mathcal{N})$ or simply by $\mathcal{N}_I$, is a semi-decision network over $\mathcal{K}_I$. The conditional probabilities of all the random nodes remain the same as in $\mathcal{N}$. The values functions of the value nodes other than u also remain the same as in $\mathcal{N}$. The value function $\mu(\pi_d)$ of the downstream-value node u is the optimal conditional expected value $E[\mathcal{N}_{II}|\pi_d]$ of the downstream component $\mathcal{N}_{II}$.

Since the decision node d is not in the upstream component $\mathcal{N}_I$, the number of decision nodes in $\mathcal{N}_I$ is less than the number of decision nodes in $\mathcal{N}$. Since the decision nodes in π_d, if any, are treated as random nodes in $\mathcal{N}_{II}$ and since the $\mathcal{N}$ decomposes at d, the number of decision nodes in $\mathcal{N}_{II}$ is also less than the number of decision nodes in $\mathcal{N}$. Furthermore the number of decision nodes in $\mathcal{N}_I$ plus the number of decision nodes in $\mathcal{N}_{II}$ equals the number of decision nodes in $\mathcal{N}$.

We shall later define the concepts of downstream and upstream components for the case when $\mathcal{K}$ and $\mathcal{N}$ are not smooth at d.

4.2.1 Properties of upstream and downstream components

This subsection gives several properties of upstream and downstream components of decomposable decision networks. Those properties will be useful in the proof of Theorem 4.1.

Given a policy $\delta_{II} = (\delta_{j+1}, \ldots, \delta_k)$, the downstream component $\mathcal{N}_{II}$ is a semi-Bayesian network. Let $P_{\mathcal{N}_{II}, \delta_{II}}(\pi_d, X_{II})$ denote the prior joint potential of this semi-Bayesian

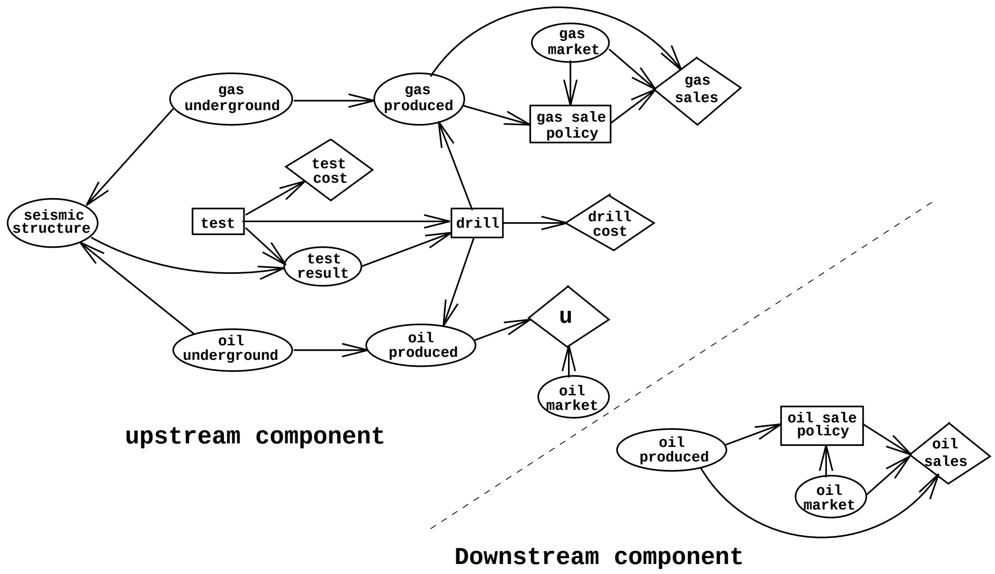

Figure 4.17: Downstream and upstream components: The downstream component is a semi-decision network, where the prior probabilities for `oil-produced` and `oil-market` are missing. In the upstream component, u is the downstream-value node, whose value function is the optimal conditional expected value of the downstream component.

network. From the definition of the downstream component, one can see that if $\mathcal{N}$ is smooth at d, then

$$P_{\mathcal{N}_{II},\delta_{II}}(\pi_d, X_{II}) = \prod_{x \in C \cap X_{II}} P(x|\pi_x) \prod_{i=j+1}^{k} P_{\delta_i}(d_i|\pi_{d_i})),$$

which is the same as the right hand side of equation (4.25). Therefore we have

Lemma 4.2 *Suppose a semi-decision network $\mathcal{N}$ is decomposable and smooth at decision node d. Let $P_{\delta_{II}}(X_{II}|\pi_d)$ be as in Lemma 4.1. Then*

$$P_{\mathcal{N}_{II},\delta_{II}}(\pi_d, X_{II}) = P_{\delta_{II}}(X_{II}|\pi_d). \tag{4.26}$$

Similarly, given a policy $\delta_I = (\delta_1, \ldots, \delta_j)$, the upstream $\mathcal{N}_I$ is a Bayesian network. Let $P_{\mathcal{N}_I,\delta_I}(X_I, S)$ denote the joint probability of this Bayesian network. From the definition of the upstream component, one can see that if $\mathcal{N}$ is smooth at d, then

$$P_{\mathcal{N}_I,\delta_I}(X_I, S) = \prod_{x \in C \cap (X_I \cup \pi_d)} P(x|\pi_x) \prod_{i=1}^{j} P_{\delta_i}(d_i|\pi_{d_i}),$$

which is equal to the right hand side of equation (4.24). Therefore we have

Lemma 4.3 *Suppose a semi-decision network $\mathcal{N}$ is decomposable and smooth at decision node d. Let $P_{\delta_I}(X_I, \pi_d)$ be as in Lemma 4.1. Then*

$$P_{\mathcal{N}_I,\delta_I}(X_I, \pi_d) = P_{\delta_I}(X_I, \pi_d). \tag{4.27}$$

The following proposition is especially important to the proof of Theorem 4.1.

Proposition 4.1 *Suppose a semi-decision network $\mathcal{N}$ is decomposable and smooth at decision node d. Let $P_{\delta_{II}}(X_{II}|\pi_d)$ and $P_{\delta_I}(X_I, \pi_d)$ be as in Lemma 4.1. Let V_I and V_{II} be the set of value nodes in $\mathcal{N}_I$ an $\mathcal{N}_{II}$ respectively. Then the conditional expected value of $\mathcal{N}_{II}$ under policy δ_{II} satisfies*

$$E_{\delta_{II}}[\mathcal{N}_{II}|\pi_d] = \sum_{X_{II}} P_{\delta_{II}}(X_{II}|\pi_d) \sum_{v \in V_{II}} \mu_v(\pi_v), \tag{4.28}$$

and the expected value of $\mathcal{N}_I$ under policy δ_I satisfies

$$E_{\delta_I}[\mathcal{N}_I] = \sum_{X_I,\pi_d} P_{\delta_I}(X_I, \pi_d)\{\sum_{v \in V_I} \mu_v(\pi_v) + E[\mathcal{N}_{II}|\pi_d]\}. \tag{4.29}$$

Proof: By definition, we have

$$E_{\delta_{II}}[\mathcal{N}_{II}|\pi_d] = \sum_{X_{II}} P_{\mathcal{N}_{II},\delta_{II}}(\pi_d, X_{II}) \sum_{v \in V_{II}} \mu_v(\pi_v).$$

Thus equation (4.28) follows from equation (4.26).

Again by definition, we have

$$E_{\delta_I}[\mathcal{N}_I] = \sum_{X_I,\pi_d} P_{\mathcal{N}_I,\delta_I}(X_I, \pi_d)\{\sum_{v \in V_I} \mu_{v_i}(\pi_{v_i}) + E[\mathcal{N}_{II}|\pi_d]\}.$$

Thus equation (4.29) follows from equation (4.27). □

4.3 Divide and conquer

This section shows how decomposability of (semi-)decision networks leads to a divide and conquer evaluation strategy.

Theorem 4.1 *Suppose a (semi-)decision network $\mathcal{N}$ is decomposable and smooth at decision node d. Let $\mathcal{N}_{II}$ be the downstream component of $\mathcal{N}$ w.r.t d, and $\mathcal{N}_I$ be the upstream component. If $\mathcal{N}_{II}$ is uniform, then*

1. *If δ^o_{II} is an optimal policy for $\mathcal{N}_{II}$ and δ^o_I is an optimal policy for $\mathcal{N}_I$, then $\delta^o =_{def} (\delta^o_I, \delta^o_{II})$ is an optimal policy for $\mathcal{N}$.*
2. *The optimal expected value $E[\mathcal{N}_I]$ of the body $\mathcal{N}_I$ is the same as the optimal expected value $E[\mathcal{N}]$ of $\mathcal{N}$.*

The theorem divides the task of evaluating a (semi-)decision network $\mathcal{N}$ into two subtasks: the task of evaluating the downstream component $\mathcal{N}_{II}$ and the task of evaluating the upstream component $\mathcal{N}_I$.

Applying the theorem to the decision network in Figure 4.17, we get that optimal decision functions for `oil-sale-policy` can be found in the downstream component, which is much smaller than the original network. The optimal decision functions for all the other decision nodes can be found in the upstream component, which is also smaller than the original network. Furthermore, we can repeatedly apply the theorem to the upstream component.

The following mathematical proof may test the reader's patience, but it is the key to understanding the correctness of our later algorithms.

Proof: For any policy δ of $\mathcal{N}$, we have

$$E_\delta[\mathcal{N}] = \sum_{X_I, \pi_d, X_{II}} P_\delta(X_I, \pi_d, X_{II}) \sum_{v \in V} \mu_v(\pi_v) \qquad \text{(By definition)}$$

$$\begin{aligned}
&= \sum_{X_I,\pi_d,X_{II}} P_{\delta_I}(X_I,\pi_d)P_{\delta_{II}}(X_{II}|\pi_d)\sum_{v\in V}\mu_v(\pi_v) \qquad \text{(By equations 4.23 and Lemma 4.1)}\\
&= \sum_{X_I,\pi_d} P_{\delta_I}(X_I,\pi_d)\{\sum_{v\in V_I}\mu_v(\pi_v)\sum_{X_{II}}P_{\delta_{II}}(X_{II}|\pi_d) + \sum_{X_{II}}P_{\delta_{II}}(X_{II}|\pi_d)\sum_{v\in V_{II}}\mu_v(\pi_v)\}\\
&= \sum_{X_I,\pi_d} P_{\delta_I}(X_I,\pi_d)\{\sum_{v\in V_I}\mu_v(\pi_v) + E_{\delta_{II}}[\mathcal{N}_{II}|\pi_d]\} \quad \text{(By equation 4.28).} \qquad (4.30)
\end{aligned}$$

Since δ^o_{II} is an optimal policy for $\mathcal{N}_{II}$ and $\mathcal{N}_{II}$ is uniform,

$$E_{\delta_{II}}[\mathcal{N}_{II}|\pi_d] \leq E_{\delta^o_{II}}[\mathcal{N}_{II}|\pi_d] = E[\mathcal{N}_{II}|\pi_d]. \qquad (4.31)$$

Noticing that $P_{\delta_I}(X_I,\pi_d)$ is non-negative, we have

$$\begin{aligned}
E_\delta[\mathcal{N}] &= \sum_{X_I,\pi_d} P_{\delta_I}(X_I,\pi_d)\{\sum_{v\in V_I}\mu_v(\pi_v) + E_{\delta_{II}}[\mathcal{N}_{II}|\pi_d]\} && \text{(By equation 4.30)}\\
&\leq \sum_{X_I,\pi_d} P_{\delta_I}(X_I,\pi_d)\{\sum_{v\in V_I}\mu_v(\pi_v) + E_{\delta^o_{II}}[\mathcal{N}_{II}|\pi_d]\} && \text{(By equation 4.31)}\\
&= \sum_{X_I,\pi_d} P_{\delta_I}(X_I,\pi_d)\{\sum_{v\in V_I}\mu_v(\pi_v) + E[\mathcal{N}_{II}|\pi_d]\} && \text{(By equation 4.31)}\\
&= E_{\delta_I}[\mathcal{N}_I] && \text{(By equation 4.29)}\\
&\leq E_{\delta^o_I}[\mathcal{N}_I] && \text{(Optimality of } \delta^o_I\text{)}\\
&= \sum_{X_I,\pi_d} P_{\delta^o_I}(X_I,\pi_d)\{\sum_{v\in V_I}\mu_v(\pi_v) + E[\mathcal{N}_{II}|\pi_d]\} && \text{(By equation 4.29)}\\
&= \sum_{X_I,\pi_d} P_{\delta^o_I}(X_I,\pi_d)\{\sum_{v\in V_I}\mu_v(\pi_v) + E_{\delta^o_{II}}[\mathcal{N}_{II}|\pi_d]\} && \text{(Optimality of } \delta^o_{II}\text{)}\\
&= E_{\delta^o}[\mathcal{N}]. && \text{(By equation 4.30)}
\end{aligned}$$

Therefore, δ^o is indeed an optimal policy for $\mathcal{N}$. The first statement of the theorem is proved.

The foregoing derivation has also shown that

$$E_\delta[\mathcal{N}] \leq E_{\delta^o_I}[\mathcal{N}_I] \leq E_{\delta^o}[\mathcal{N}].$$

Letting δ be δ^o, we get

$$E_{\delta^o}[\mathcal{N}] = E_{\delta^o_I}[\mathcal{N}_I].$$

Therefore $E[\mathcal{N}] = E[\mathcal{N}_I]$. The proof is completed. $\square$

Chapter 5

Stepwise-decomposable decision networks

This chapter introduces and studies the most important concept of this thesis, namely stepwise-decomposable decision networks (SDDN). Roughly speaking, a SDDN is a decision network that can be decomposed into n — the number of decision nodes — subnetworks such that each subnetwork contains only one decision node and that the original network can be evaluated through the evaluation of those subnetworks (Section 5.4). A first reason why SDDN's are computationally desirable is that the subnetworks may be substantially smaller than the original network.

A second reason why SDDN's are computationally desirable is that each of the subnetworks is a semi-decision network with only one decision node. Single-decision-node semi-decision networks can be evaluated by enumerating the values of the parents of the decision node instead of enumerating all the possible policies or decision functions (see Section 5.5). Suppose that the decision node has n parents and that all the variables are binary. Then, the parents can assume 2^n possible values, while there are $2^{(2^n)}$ decision functions!

The organization of this chapter is as follows. The definition of SDDN's is given in Section 5.1, and Section 5.2 shows that smooth SDDN's are stepwise-solvable. The issue of testing stepwise-decomposability is addressed in Section 5.3. In Section 5.4, we discuss how to evaluate a smooth SDDN by using the divide and conquer strategy outlined in the previous chapter. An algorithm is presented in Section 5.6, which makes use of the subroutine given in Section 5.5 for evaluating simple semi-decision networks.

Non-smooth SDDN's are treated in the next chapter.

5.1 Definition

This section defines stepwise-decomposable decision networks.

In a decision network skeleton $\mathcal{K}$, a decision node d is a *stepwise-decomposability candidate node* or simply an *SD candidate node* if π_d m-separates d from all other decision nodes and their parents. A decision node is an *SD candidate node* in a decision network $\mathcal{N}$ if it is an SD candidate node in the skeleton of $\mathcal{N}$.

As an example, consider the decision network skeleton in Figure 2.11). Both `oil-sale-policy` and `gas-sale-policy` are SD candidate nodes, while `drill` and `test` are not. The decision nodes `oil-sale-policy` and `gas-sale-policy` are not m-separated from `drill` (`test`) by $\pi_{\texttt{drill}}$ ($\pi_{\texttt{test}}$).

Lemma 5.1 *Suppose d is an SD candidate in a decision network skeleton $\mathcal{K}$. Then the downstream set $Y_{II}(d, \mathcal{K})$ contains only one decision node, which is d itself. So, if $\mathcal{K}$ contains more than one decision node, then it is decomposable at d.* $\Box$

When d is an SD candidate node in decision network $\mathcal{N}$ and $\mathcal{N}$ is smooth at d, the upstream component $\mathcal{N}_I$ of $\mathcal{N}$ w.r.t d is called the *body* of $\mathcal{N}$ w.r.t d, and the downstream component $\mathcal{N}_{II}$ of $\mathcal{N}$ w.r.t d is called the *tail* of $\mathcal{N}$ w.r.t d. Moreover, the downstream-value node in $\mathcal{N}_I$ will be referred to as the *tail-value node* .

A decision network skeleton $\mathcal{K}$ is *stepwise-decomposable* if either it contains zero or one decision node, or

1. There exists an SD candidate decision node d, and

2. The body $\mathcal{K}_I$ of $\mathcal{K}$ w.r.t d is stepwise-decomposable.

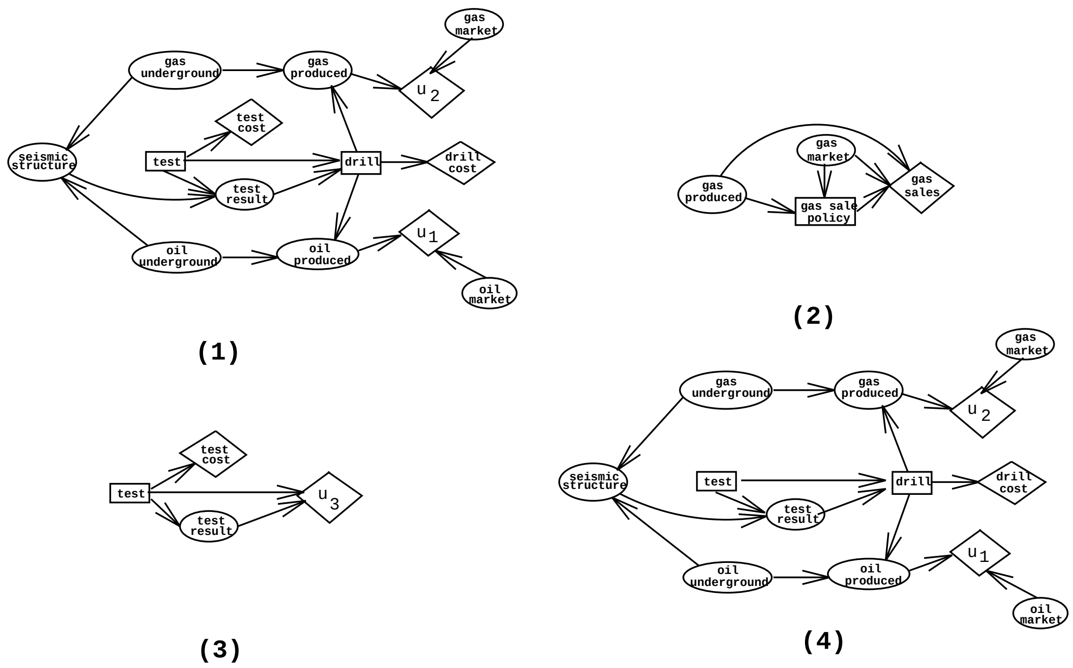

Figure 5.18: Step by step decomposition of the decision network in Figure 2.11.

A (semi-)decision network is *stepwise-decomposable* if its underlying skeleton is. The term "stepwise-decomposable decision network" will be abbreviated to SDDN.

Suppose a decision network $\mathcal{N}$ is stepwise-decomposable. If $\mathcal{N}$ contains more than one decision node, then it has at least one candidate node d. According to Lemma 5.1, $\mathcal{N}$ is decomposable at d and it decomposes into a body $\mathcal{N}_I$ and a tail. $\mathcal{N}_I$ is again stepwise-decomposable. If $\mathcal{N}_I$ contains more than one decision node, we can again decompose $\mathcal{N}_I$ into a body and a tail, an so and so forth, till there is only one decision node left in the body. In other words, we can decompose an SDDN into a series of subnetworks (tails) in a step-by-step fashion. This is why the term "stepwise-decomposable" was chosen.

As an example, let $\mathcal{N}$ be the decision network in Figure 2.11. $\mathcal{N}$ is stepwise-decomposable. The node `oil-sale-policy` is an SD candidate node in $\mathcal{N}$, and $\mathcal{N}$ decomposes at `oil-sale-policy` into a body $\mathcal{N}_I$ and at tail, as shown in Figure 4.17.

The node `gas-sale-policy` is an SD candidate node in $\mathcal{N}_I$, and $\mathcal{N}_I$ decomposes at `gas-sale-policy` into a tail and a body. The body is shown in Figure 5.18 (1), and the tail Figure 5.18(2). In Figure 5.18 (1), `drill` is an SD candidate node, and the network decomposes at `drill` into a body as shown in Figure 5.18 (3) and a tail as shown in Figure 5.18 (4).

The decision network skeleton in Figure 3.15 contains two decision nodes, but no SD candidate nodes. So, it is not stepwise-decomposable.

5.1.1 Another way of recursion

In the definition of decomposability, the number of decision nodes is recursively reduced by cutting tails that contain a single decision node. Another way to recursively reduce the number of decision nodes is to replace them one by one with deterministic random nodes. Let us first prove a lemma.

Lemma 5.2 *Let d be an SD candidate node in a decision network skeleton $\mathcal{K}$. Let $\mathcal{K}_I$ be the body of $\mathcal{K}$ w.r.t d, and let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by replacing d by a deterministic node. Then $\mathcal{K}_I$ is stepwise-decomposable if and only if $\mathcal{K}'$ is.*

Proof: We prove this lemma by induction on the number of decision nodes in $\mathcal{K}$. When d is the only decision node in $\mathcal{K}$, both $\mathcal{K}_I$ and $\mathcal{K}'$ contain zero decision nodes, and hence both are stepwise-decomposable.

Suppose the lemma is true for the case of $k-1$ decision nodes. Consider the case of k decision nodes. One can easily verify that a decision node is an SD candidate node in $\mathcal{K}_I$ if and only if it is an SD candidate node in $\mathcal{K}'$.

Suppose a decision node d' ($\neq d$) is a SD candidate node in $\mathcal{K}_I$ (hence in $\mathcal{K}'$). There are two cases depending on whether or not d is in the downstream set of $\pi_{d'}$ in $\mathcal{K}$. When

d is in the downstream set of $\pi_{d'}$, the body of $\mathcal{K}_I$ w.r.t d' is the same as that of $\mathcal{K}'$; hence $\mathcal{K}_I$ is stepwise-decomposable if and only if $\mathcal{K}'$ is. When d is not in the downstream set of $\pi_{d'}$, let $\mathcal{K}_I^*$ be the body of $\mathcal{K}_I$ w.r.t d', and $\mathcal{K}'^*$ be that of $\mathcal{K}'$. Then $\mathcal{K}'^*$ is the body of $\mathcal{K}_I^*$ w.r.t d. By the induction hypothesis, $\mathcal{K}'^*$ is stepwise-decomposable if and only if $\mathcal{K}_I^*$ is. Therefore, $\mathcal{K}'$ is stepwise-decomposable if and only if $\mathcal{K}_I$ is. The lemma is proved. □

This lemma leads directly to the following proposition.

Proposition 5.1 *A decision network skeleton is stepwise-decomposable if and only if either it contains no decision nodes or*

1. *There exists an SD candidate decision node d, and*
2. *If d is replaced by a deterministic node, the resulting decision network skeleton (with one less decision node) is stepwise-decomposable.*

One can view Proposition 5.1 as an alternative definition of stepwise-decomposability. The original definition is based an recursive construct that will be used directly in the algorithms, while the recursive construct of this alternative definition is the same as that of the definition of stepwise-solvability, which makes it convenient to study the relationship between stepwise-decomposability and stepwise-solvability, as the reader will see in the next section.

5.2 Stepwise-decomposability and stepwise-solvability

A decision network is *smooth* if it is smooth at every decision node.

Theorem 5.1 *A smooth decision network is stepwise-solvable if it is stepwise-decomposable.*

Proof: Let $\mathcal{N}$ be a smooth decision network and d be a decision node. Because of Proposition 5.1, it suffices to show that if d is an SD candidate node, then it is also an SS candidate node.

Suppose d is an SD candidate node. Let X_1 be the set of random and decision nodes in the upstream of π_d; let $\mathcal{N}_I$ and $\mathcal{N}_{II}$ be respectively the body and tail of $\mathcal{N}$ w.r.t d; let V_I be the set of value nodes $\mathcal{N}_I$; let δ_I be a policy of $\mathcal{N}_I$; and and let δ_{II} be a policy of $\mathcal{N}_{II}$, i.e a decision function of d. By equation (4.30) we have that

$$E_{(\delta_I,\delta_{II})}[\mathcal{N}] = \sum_{X_1,\pi_d} P_{\delta_I}(X_1,\pi_d)\{\sum_{v\in V_I}\mu_v(\pi_v) + E_{\delta_{II}}[\mathcal{N}_{II}|\pi_d]\}.$$

Fixing δ_I, we can rank all the possible policies δ_{II} of $\mathcal{N}_{II}$ according to the value $E_{(\delta_1,\delta_{II})}[\mathcal{N}]$. Since $P_{\delta_I}(X_1,\pi_d)$ is non-negative, this ranking does not depend on the value of δ_I. Therefore d is an SS candidate node. The theorem is proved. □

We shall show later that the theorem is true also for non-smooth decision networks, and that under "normal" conditions stepwise-solvability implies stepwise-decomposability as well (Chapter 8).

The remainder of this chapter is devoted to the following two questions: How can one test stepwise-decomposability? How can one evaluate a smooth SDDN?

5.3 Testing stepwise-decomposability

In a decision network, we say that a decision node d *precedes* another decision node d' if there is a directed path from d to d'. Decision nodes that precede no other decision nodes are called *leaf decision nodes* .

We say d *weakly precedes* another d' if d' is in the downstream set of π_d. Decision nodes that weakly precede no other decision nodes are called *weak leaf decision nodes* . The following lemma follows from the definition of SD candidate node and the definition of downstream sets.

Lemma 5.3 *In a decision network, if node is an SD candidate decision node, then it is a weak leaf decision node.*

Proof: Straightforward. □

Lemma 5.4 *Let d and d' be two decision nodes in a decision network. If d precedes d', then d weakly precedes d'.*

Proof: Since d precedes d', there is a directed path from d to d'. No nodes in this path can be in π_d, because otherwise there would be a cycle in the network. Hence, d' is not m-separated from d by π_d. Consequently, d' is in the downstream set of d. The lemma is therefore proved. □

Combining the forgoing two lemmas, we get

Proposition 5.2 *In a decision network, an SD candidate decision node must be a leaf decision node. In other words, if a node is not a leaf decision node, then it cannot be a candidate node.*

Proof: Suppose d is a decision node but not a leaf decision node. Then there exists another decision node d' such that d precedes d'. By Lemma 5.4, d weakly precedes d', hence d is not a weak leaf decision node. By Lemma 5.3, d cannot be a candidate node. □

This proposition leads to the following algorithm for testing if a decision network skeleton is stepwise-decomposable.

Procedure TEST-STEPWISE-DECOMPOSABILITY($\mathcal{K}$):

- Input: $\mathcal{K}$ — a decision network skeleton.
- Output: "YES" or "NO" depending on whether $\mathcal{K}$ is stepwise-decomposable.

If there are no decision nodes in $\mathcal{K}$, return "YES".

Else

1. Find a leaf decision node d of $\mathcal{K}$.
2. Check if d is an SD candidate decision node.
   - **If** d is not, find another leaf decision node d' and go to 2 with d'. **If** there are no more leaf decision nodes, return "NO".
   - **If** d is an SD candidate decision node, compute the body $\mathcal{K}_I$ of $\mathcal{K}$ w.r.t d, and recursively call TEST-STEPWISE-DECOMPOSABILITY($\mathcal{K}_I$).

What is the running time of TEST-STEPWISE-DECOMPOSABILITY? Let n be the total number of nodes in $\mathcal{K}$, k be the number of decision nodes, a be the number of arcs, e be the number of edges in the moral graph of $\mathcal{K}$. Finding a leaf decision node takes at most $O(a)$ time. Testing if a decision node is an SD candidate node and the computation of a body are of the same order complexity as testing the connectivity of the moral graph of $\mathcal{K}$, which is $O(n+e)$ by either breadth-first search or depth-first search. In worst case, all the decision nodes are leaf nodes and there is only one SD candidate node. If the only candidate node is always tested the last, then the complexity of TEST-STEPWISE-DECOMPOSABILITY is $O(k^2(n+e+a)) = O(k^2(n+e))$. On the other hand, if every leaf decision node tested is an SD candidate node, the complexity is $O(k(n+e))$.

5.4 Recursive tail cutting

In Section 5.6, we shall give an algorithm for evaluating smooth SDDN's. In preparation, this section shows that an optimal policy for a smooth SDDN can be computed by recursively evaluating tail semi-decision networks, and the next section studies how to evaluate a tail semi-decision network.

Theorem 5.2 *Let $\mathcal{N}$ be a SDDN and d be an SD candidate node. Let $\mathcal{N}_{II}$ be the tail of $\mathcal{N}$ w.r.t d, and $\mathcal{N}_I$ be the body. Let δ^o_{II} be an optimal policy for $\mathcal{N}_{II}$, i.e an optimal* *decision function of d, and let and δ_I^o be an optimal policy for $\mathcal{N}_I$. If $\mathcal{N}$ is smooth at d, then*

1. *$\delta^o =_{def} (\delta_I^o, \delta_{II}^o)$ is an optimal policy for $\mathcal{N}$.*
2. *The optimal expected value $E[\mathcal{N}_I]$ of the body $\mathcal{N}_I$ is the same as the optimal expected value $E[\mathcal{N}]$ of $\mathcal{N}$.*

The proof is postponed to the end of this section. The theorem implies the following strategy of evaluating a smooth SDDN $\mathcal{N}$: first compute and evaluate a tail $\mathcal{N}_{II}$ of $\mathcal{N}$, then compute and evaluate a tail of $\mathcal{N}_I$, and so on so forth. We shall refer to this strategy as *recursive tail cutting* .

5.4.1 Simple semi-decision networks

This subsection introduces the concept of simple semi-decision networks. The concept is important to the proof of Theorem 5.2, as well as to our later algorithms.

A semi-decision network $\mathcal{N} = (Y, A, \mathcal{P}, \mathcal{F}|S)$ is *simple* if it contains only one decision node d and $\pi_d{=}S$.

Proposition 5.3 *Suppose $\mathcal{N}$ is a smooth SDDN and d is an SD candidate. Then*

1. *The tail $\mathcal{N}_{II}$ of $\mathcal{N}$ w.r.t d is a simple semi-decision network, and*
2. *The body $\mathcal{N}_I$ of $\mathcal{N}$ w.r.t d is again a smooth SDDN.*

Proof: The proof is straightforward. □

Suppose $\mathcal{N} = (Y, A, \mathcal{P}, \mathcal{F}|S)$ is a simple semi-decision network. Let δ_d be a decision function of the only decision node d of $\mathcal{N}$.

The conditional expected value $E_{\delta_d}[\mathcal{N}|S]$ depend on S and δ_d. Since δ_d is a function from Ω_S to Ω_d, $E_{\delta_d}[\mathcal{N}|S{=}\beta]$ may depends possible on $\delta_d(\beta)$ for all $\beta{\in}\Omega_S$.

Lemma 5.5 *For any value $\beta\in\Omega_S$, $E_{\delta_d}[\mathcal{N}|S{=}\beta]$ depends only on the value $\delta_d(\beta)$ of the decision function δ_d at β, in the sense that fixing $\delta_d(\beta)$ fixes $E_{\delta_d}[\mathcal{N}|S{=}\beta]$, no matter what the $\delta_d(\beta')$'s ($\beta'\in\Omega_S, \beta'\neq\beta$) are.*

The proof is postponed till the end of this section. The implication of this lemma is that $E_{\delta_d}[\mathcal{N}|S{=}\beta]$ is really a function of β and the value $\delta_d(\beta)$ of δ_d at β. To make this more explicit, let $\delta_d(\beta){=}\alpha$, then $E_{\delta_s}[\mathcal{N}|S{=}\beta]$ is a function of β and α. To signify this fact, we shall sometimes write $E_{\delta_d}[\mathcal{N}|S{=}\beta]$ as $E_{\delta_d:\delta_d(\beta)=\alpha}[\mathcal{N}|S{=}\beta]$.

The following proposition will also be proved at the end of this section.

Proposition 5.4 *Simple semi-decision networks are uniform.*

The corollary below follows from Theorem 5.2 and Lemma 5.5

Corollary 5.1 *Let $\mathcal{N} = (Y, A, \mathcal{P}, \mathcal{F}|S)$ be a simple semi-decision network and let d be the only decision node. Then an optimal decision function δ_d^o for d is given by*

$$\delta_d^o(\beta) = arg\, max_{\alpha\in\Omega_d} E_{\delta_d:\delta_d(\beta)=\alpha}[\mathcal{N}|S{=}\beta]. \tag{5.32}$$

□

5.4.2 Proofs

Proof of Theorem 5.2: Because of Theorem 4.1, it suffices to show that $\mathcal{N}_{II}$ is uniform. By Proposition 5.3, the tail $\mathcal{N}_{II}$ is a simple semi-decision network. According to Proposition 5.4, $\mathcal{N}_{II}$ must be uniform. The theorem is therefore proved. □.

Proof of Lemma 5.5: Since $\pi_d{=}S$, we can write $P_{\delta_d}(d|S)$ for $P_{\delta_d}(d|\pi_d)$. Let C and V be the set of random and value nodes respectively. The joint potential $P_{\delta_d}(C, d)$ is given by

$$P_{\delta_d}(C, d) = P_{\delta_d}(d|S{=}\beta) \prod_{c\in C} P(c|\pi_c).$$

Therefore

$$E_{\delta_d}[\mathcal{N}|S{=}\beta] = \sum_{C\cup\{d\}-S} P_{\delta_d}(d|S{=}\beta) \prod_{c\in C} P(c|\pi_c) \sum_{v\in V} \mu_v(\pi_v). \tag{5.33}$$

The only term that contains δ_d is $P_{\delta_d}(d|S{=}\beta)$, which only depends on the value of δ_d at β according to equation (3.13). Thus, the lemma is proved. □

Proof of Proposition 5.4: Suppose $\mathcal{N} = (Y, A, \mathcal{P}, \mathcal{F}|S)$ is a simple semi-decision network. Let d be the only decision node. We need to show that for any decision function δ_d^o of d, if

$$E_{\delta_d{}^o}[\mathcal{N}] = max_{\delta_d\in\Delta_d} E_{\delta_d}[\mathcal{N}], \tag{5.34}$$

then for any value $\beta\in\Omega_S$

$$E_{\delta_d{}^o}[\mathcal{N}|S{=}\beta] = max_{\delta_d} E_{\delta_d\in\Delta_d}[\mathcal{N}|S{=}\beta]. \tag{5.35}$$

For the sake of contradiction, assume there were a decision function δ_d^o that satisfies (5.34) which did not satisfy (5.35). Then, there must exist another decision function δ_d^* and a value $\beta \in \Omega_S$ such that

$$E_{\delta_d{}^o}[\mathcal{N}|S{=}\beta] < E_{\delta_d^*}[\mathcal{N}|S{=}\beta].$$

Construct a new decision function δ_d^n which is the same as δ_d^* at β and which is the same as δ_d^o at all other values β' of S. By Lemma 5.5, $E_{\delta_d^n}[\mathcal{N}|S{=}\beta] = E_{\delta_d^*}[\mathcal{N}|S{=}\beta]$, and $E_{\delta_d^n}[\mathcal{N}|S{=}\beta'] = E_{\delta_d^o}[\mathcal{N}|S{=}\beta']$ for any $\beta'\in\Omega_S$ such that $\beta'\neq\beta$. Hence, we have

$$\begin{aligned}
E_{\delta_d^n}[\mathcal{N}] &= \sum_{\beta'\in\Omega_S} E_{\delta_d^n}[\mathcal{N}|S{=}\beta']. \\
&= E_{\delta_d^*}[\mathcal{N}|S{=}\beta] + \sum_{\beta'\in\Omega_S,\beta'\neq\beta} E_{\delta_d^o}[\mathcal{N}|S{=}\beta'] \\
&> E_{\delta_d^o}[\mathcal{N}|S{=}\beta] + \sum_{\beta'\in\Omega_S,\beta'\neq\beta} E_{\delta_d^o}[\mathcal{N}|S{=}\beta'] \\
&= \sum_{\beta'\in\Omega_S} E_{\delta_d^o}[\mathcal{N}|S{=}\beta'] \\
&= E_{\delta_d^o}[\mathcal{N}].
\end{aligned}$$

A contradiction. Therefore, it must be the case that (5.34) implies (5.35). The proposition is proved. □

5.5 Evaluating simple semi-decision networks

This section presents an algorithm for evaluating simple semi-decision networks.

Let $\mathcal{N} = (Y, A, \mathcal{P}, \mathcal{F}|S)$ be a simple semi-decision network, and let d be the only decision node.

Let $\mathcal{N}_0$ be the semi-Bayesian network obtained from $\mathcal{N}$ by removing all the value nodes, and by deleting all the arcs into d and treating d as a root random node without prior probability. Let P_0 be the joint potential of $\mathcal{N}_0$, i.e the product of all the conditional probabilities in $\mathcal{N}$.

For any value node v of $\mathcal{N}$, all its parents—all the nodes in π_v—are in $\mathcal{N}_0$. Thus, we can compute the marginal potential $P_0(\pi_v, S, d)$. We define the *evaluation functional* $e(d, S)$ of $\mathcal{N}$ by

$$e(d, S) = \sum_{v \in V} \sum_{\pi_v - S \cup \{d\}} P_0(\pi_v, S, d)\mu_v(\pi_v). \tag{5.36}$$

Theorem 5.3 *Let $\mathcal{N} = (Y, A, \mathcal{P}, \mathcal{F}|S)$ be a simple semi-decision network; let d be the only decision node; and let $e(d, S)$ be the evaluation functional of $\mathcal{N}$. Then*

1. *An optimal decision function δ_d^o can be found by*

$$\delta_d^o(\beta) = arg\ max_{\alpha \in \Omega_D} e(d{=}\alpha, S{=}\beta), \forall \beta \in \Omega_S, \tag{5.37}$$

2. *The conditional optimal expected value $E[\mathcal{N}|S]$ is given*

$$E[\mathcal{N}|S{=}\beta] = max_{\alpha \in \Omega_d} e(d{=}\alpha, S{=}\beta), \forall \beta \in \Omega_S. \tag{5.38}$$

The proof is postponed to the end of this section.

The theorem suggests the following procedure for evaluating simple semi-decision networks.

Procedure S-EVALUATE($\mathcal{N}$):

- Input: $\mathcal{N}$ — A simple semi-decision network.
- Output: An optimal decision function for the only decision node d, and the optimal conditional expected value of $\mathcal{N}$.

1. Construct the semi-Bayesian network $\mathcal{N}_0$.
2. Compute the marginal potential $P_0(\pi_v, S, d)$ for each the value nodes v.
3. Obtain the evaluation functional $e(d, S)$ by equation (5.36).
4. Compute the optimal conditional expected value by equation (5.38) and an optimal policy by equation (5.37).

Note 1). For those who are familiar with Bayesian network inferences, the clique tree propagation approach (Jensen *et al* 1990, Lauritzen and Spiegehalter 1988, and Shafer and Shenoy 1988) can be used to compute all marginal potentials $P_0(\pi_v, S, d)$'s. All the marginal potentials can be computed by traversing the clique tree twice. When there is only one value node, there is only one marginal potential to compute, which can be done by traversing the clique tree only once.

Note 2). Relating back to the point made in the introduction of the chapter about the evaluation of single-decision-node semi-decision networks, we see from Equation (5.37) that S-EVALUATE does indeed evaluate a simple semi-decision network by enumerating the values of the parents of the only decision node, rather than enumerating all the decision functions, as the procedure NAIVE would do.

An interesting question is: Can we do the same for (semi-) decision networks with more than one decision node? The answer is no, unless all the decision nodes have the same parents. Essentially, what goes on in S-EVALUATE is that for any $\beta \in \Omega_{\pi_d}$, we instantiate the parents π_d of the only decision node d to β and figure out the value for d that maximizes the expected value of the simple decision network. When there are at least two decision nodes, say d_1 and d_2, with different parents, we can not do the same because instantiating the parents of d_1, for instance, would mean that all the parents of d_1 are observed at the time the decision d_2 is to be made. This may not be true at all.

Proof of Theorem 5.3: Let δ_d be a decision function of d. Let $P_{\delta_d}(X)$ be the potential over the set X of all the random and decision nodes of $\mathcal{N}$ under policy δ_d. We have

$$P_{\delta_d}(X) = P_0(X) P_{\delta_d}(d|S). \tag{5.39}$$

Therefore, we have

$$\begin{aligned}
E_{\delta_d}[\mathcal{N}|S] &= \sum_{X-S} P_{\delta_d}(X) \sum_{v \in V} \mu_v(\pi_v) && \text{(By definition)} \\
&= \sum_{X-S} P_{\delta_d}(d|S) P_0(X) \sum_{v \in V} \mu_v(\pi_v) && \text{(By equation (5.39))} \\
&= \sum_{v \in V} \sum_{d} P_{\delta_d}(d|S) \sum_{X-(S \cup \{d\})} P_0(X) \mu_v(\pi_v) \\
&= \sum_{v \in V} \sum_{d} P_{\delta_d}(d|S) \sum_{\pi_v} \sum_{X-(\pi_v \cup S \cup \{d\})} P_0(X) \mu_v(\pi_v) \\
&= \sum_{v \in V} \sum_{d} P_{\delta_d}(d|S) \sum_{\pi_v} P_0(\pi_v, S, d) \mu_v(\pi_v) && \text{(Marginal potential)}
\end{aligned}$$

From equation (3.13), we see that given S, $P_{\delta_d}(d|S)$ is the characteristic function $\chi_{\{d|d=\delta_d(S)\}}(d)$ of the set $\{d|d = \delta_d(S)\}$. Therefore

$$E_{\delta_d}[\mathcal{N}|S] = \sum_{v \in V} \sum_{\pi_v} P_0(\pi_v, S, \delta_d(S)) \mu_v(\pi_v). \tag{5.40}$$

Hence,

$$E_{\delta_d:\delta_d(\beta)=\alpha}[\mathcal{N}|S{=}\beta] = e(d{=}\alpha, S{=}\beta). \tag{5.41}$$

The first item of the theorem follows from Corollary 5.1 and equation (5.41). The second item follows from the first item. The theorem is proved. □

5.6 The procedure EVALUATE

We are now ready to present our algorithm for evaluating smooth SDDN's. The correctness of the algorithm is guaranteed by Theorem 5.2 and Proposition 5.3.

Procedure EVALUATE($\mathcal{N}$):

- Input: $\mathcal{N}$ — a smooth SDDN.
- Output: An optimal policy for and the optimal expected value of $\mathcal{N}$.

If there are no decision nodes, Call N-EVALUATE($\mathcal{N}$) to compute the expected value, and stop.

Else

1. Find an SD candidate decision node d,
2. Compute the tail $\mathcal{N}_{II}$ of $\mathcal{N}$ w.r.t d,
3. Call S-EVALUATE($\mathcal{N}_{II}$) to compute an optimal policy for and the optimal conditional expected value $E[\mathcal{N}_{II}|\pi_d]$ of $\mathcal{N}_{II}$,
4. Compute the body $\mathcal{N}_I$ of $\mathcal{N}$ w.r.t d ($E[\mathcal{N}_{II}|\pi_d]$ is used here), and
5. Recursively call EVALUATE($\mathcal{N}_I$).

In EVALUATE, the subroutine N-EVALUATE is a procedure for evaluating decision networks that contain no decision nodes, which is given below.

Procedure N-EVALUATE($\mathcal{N}$):

- Input: $\mathcal{N}$ — a decision network with no decision nodes.
- Output: the optimal expected value of $\mathcal{N}$.

If there are no value nodes in $\mathcal{N}$, return 0.

Else

1. Let $v_1, \ldots, v_m$ be all the value nodes. Compute $P(\pi_{v_i})$ for all the v_i's.
2. Return
$$\sum_{i=1}^{m} \sum_{\pi_{v_i}} P(\pi_{v_i}) \mu_{v_i}(\pi_{v_i}).$$

Note that in EVALUATE, there is the subtask of finding an SD candidate node, which is also in the TEST-STEPWISE-DECOMPOSABILITY. In implementation, one should avoid doing this twice.

Also note that in N-EVALUATE one can, as in S-EVALUATE, compute the marginal probabilities $P(\pi_{v_i})$ by using the clique tree approach. All those marginal probabilities can be computed by traversing the clique tree only twice. When there is only one value node, one pass through the clique tree is enough.

Finally, no complexity analysis of EVALUATE is carried out here because a more general version of EVALUATE will be given in the next chapter.

Chapter 6

Non-Smooth SDDN's

The previous chapter has discussed how to evaluate a smooth SDDN. This chapter deals with non-smooth SDDN's. We extend the concepts of tail and body to the non-smooth case in such a way that, as in the smooth case, optimal policies of the tail and optimal policies of the body together form optimal policies of the original network (Section 6.2), and thereby obtain a procedure called EVALUATE1 that is very similar to EVALUATE (Section 6.3). The correctness of EVALUATE1 is proved in Section 6.4. Both the presentation and the proof of EVALUATE1 rely upon the preparatory Section 6.1, which discusses how to transform a non-smooth SDDN into an equivalent smooth SDDN by a series of arc reversals.

Several algorithms have been previously developed for evaluating influence diagrams. Being special SDDN's, influence diagrams can also be evaluated by EVALUATE1. Section 6.5 compares EVALUATE1 with the previous algorithms for evaluating influence diagrams.

6.1 Smoothing non-smooth SDDN's

An algorithm for evaluating non-smooth SDDN's will be given in Section 6.3. In preparation, this section shows how to transform a non-smooth SDDN into an equivalent smooth SDDN by a series of arc reversals.

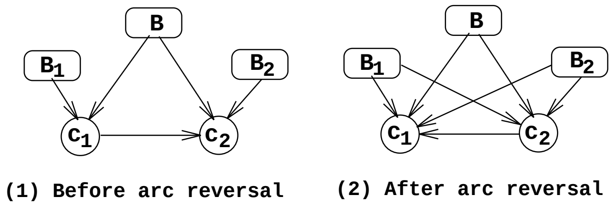

Figure 6.19: The concept of arc reversal: At the beginning the parent set of c_1 is $B \cup B_1$ and the parent set of c_2 is $B \cup B_2 \cup \{c_1\}$. After reversing the arc $c_1 \rightarrow c_2$, the parent set of c_1 becomes $B \cup B_1 \cup B_2 \cup \{c_2\}$ and the parent set of c_2 becomes $B \cup B_2 \cup B_1$. There are no graphical changes otherwise.

6.1.1 Equivalence between decision networks

Two decision networks are *equivalent* if

1. They have the same decision nodes, the same policy space, and

2. For each policy, they have the same expected value.

Lemma 6.1 *If two decision networks are equivalent, then they have the same optimal policies and the same optimal expected value.* □

Note that a decision node can have the same decision function space in two different decision networks even when its parents vary from one network to the other. For example, consider the case where a decision node d has only one parent x in one decision network, while two parents y_1 and y_2 in the other. If the frame of x is the same as the Cartesian product of the frames of y_1 and of y_2, then d has the same decision function space in the two networks. This is why, in the foregoing definition of equivalence between two decision networks, we do not require that a decision node have the same parents in both networks. This note will be useful in Chapter 9.

6.1.2 Arc reversal

Arc reversal is an operation that transforms one decision network into another different but equivalent decision network (Howard and Matheson 1984, Shachter 1986). We introduce arc reversals at two levels: first the level of skeleton, and then the level of number.

Let c_1 and c_2 be two random nodes in a decision network skeleton $\mathcal{K}$. Suppose there is an arc from c_1 to c_2. When there is no other directed path from c_1 to c_2, we say that the arc $c_1 \rightarrow c_2$ is *reversible* . .

Let $B = \pi_{c_1} \cap \pi_{c_2}$, $B_1 = \pi_{c_1} - \pi_{c_2}$, and $B_2 = \pi_{c_2} - (\pi_{c_1} \cup \{c_1\})$. In a decision network skeleton, to *reverse a reversible arc* $c_1 \rightarrow c_2$ is to delete that arc, draw an arc from c_2 to c_1, an arc from each node in B_2 to c_1, and an arc from each node in B_1 to c_2. Figure 6.19 illustrates this concept.

Let $\mathcal{K}'$ be the decision network skeleton resulting from reversing $c_1 \rightarrow c_2$ in $\mathcal{K}$. Let π'_x denote the set of parents of a node x in $\mathcal{K}'$. Then $\pi'_{c_1} = B \cup B_1 \cup B_2 \cup \{c_2\}$ and $\pi'_{c_2} = B \cup B_2 \cup B_1$.

Let $\mathcal{N}$ be a decision network and $\mathcal{K}$ be the underlying skeleton. *To reverse a reversible arc* $c_1 \rightarrow c_2$ *in* $\mathcal{N}$ is to reverse that arc in the underlying skeleton $\mathcal{K}$, and to set the conditional probabilities $P(c_1|\pi'_{c_1})$ and $P(c_2|\pi'_{c_2})$ to be as follows:

$$P(c_2|\pi'_{c_2}) = P(c_1|B, B_1, B_2, c_2) \quad =_{def} \quad \sum_{c_1} P(c_1, c_2|B, B_1, B_2), \tag{6.42}$$

$$P(c_1|\pi'_{c_1}) = P(c_2|B, B_1, B_2) \quad =_{def} \quad \frac{P(c_1, c_2|B, B_1, B_2)}{P(c_2|B, B_1, B_2)}, \tag{6.43}$$

where $P(c_1, c_2|B, B_1, B_2) = P(c_1|\pi_{c_1})P(c_2|\pi_{c_2})$, and $P(c_1|\pi_{c_1})$ and $P(c_2|\pi_{c_2})$ are in turn the conditional probabilities of c_1 and c_2 in $\mathcal{N}$ respectively. In (6.43), $P(c_2|B, B_1, B_2)$ may be zero. When it is the case, $P(c_1|B, B_1, B_2, c_2)$ is defined to be constant 1.

Note that arc reversals at the level of skeleton do not involve numerical computations, while arc reversals in decision networks do. The following lemma reveals some properties of arc reversals in decision networks, which will be useful later.

Lemma 6.2 *Suppose an arc $c_1 \rightarrow c_2$ in a decision network $\mathcal{N}$ is reversible. Let $\mathcal{N}'$ be the decision network resulting from reversing $c_1 \rightarrow c_2$ in $\mathcal{N}$. Let π'_x denote the set of parents of a node x in $\mathcal{N}'$, and let $P'(c|\pi'_c)$ denote the conditional probability of a random node c in $\mathcal{N}'$. Then*

1. *For any node x that is not a random node, $\pi'_x = \pi_x$;*
2. *For any random node c other than c_1 and c_2,*

$$\pi'_c = \pi_c \text{ and } P'(c|\pi'_c) = P(c|\pi_c);$$

3. *And*

$$P'(c_1|\pi'_{c_1})P'(c_2|\pi'_{c_2}) = P(c_1|\pi_{c_1})P(c_2|\pi_{c_2}).$$

Proof: The lemma follows directly from the definition of arc reversal. □

Proposition 6.1 *Let $\mathcal{N}$ be a decision network. Let $\mathcal{N}'$ be the decision network obtained from $\mathcal{N}$ by reversing a reversible arc. Then $\mathcal{N}'$ and $\mathcal{N}$ are equivalent.*

Proof: According to Lemma 6.2 (1), $\mathcal{N}$ and $\mathcal{N}'$ have the same decision nodes, and that each decision node has the same parents. So, $\mathcal{N}$ and $\mathcal{N}'$ have the same policy space. By Lemma 6.2 (2) and (3), we have that $E_\delta[\mathcal{N}] = E_\delta[\mathcal{N}']$ for any policy δ. The proposition is thus proved. □

6.1.3 Disturbance nodes, disturbance arcs, and disturbance recipients

Consider a decision network skeleton $\mathcal{K}$. Suppose d is a decision node in $\mathcal{K}$. If $\mathcal{K}$ is not smooth at d, then there are arcs from the downstream set $Y_{II}(d, \mathcal{K})$ to nodes in π_d. A *disturbance node* of d is a node in Y_{II} from which there is a directed path to at least one node in π_d. The arcs on such a path are called *disturbance arcs* of d, because they go

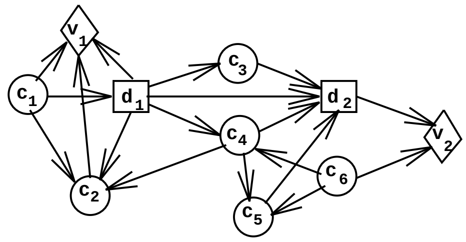

Figure 6.20: A non-smooth decision network skeleton.

against the "stream". The nodes in π_d that are pointed to by disturbance arcs are called *disturbance recipients* of d.

As an example, let $\mathcal{K}$ be the decision network skeleton in Figure 6.20. The downstream set $Y_{II}(d_2, \mathcal{K})$ consists of d_2, c_6 and v_2. The node c_6 is a disturbance node of d_2, the arcs $c_6 \rightarrow c_4$ and $c_6 \rightarrow c_5$ are disturbance arcs of d_2, and c_4 and c_5 are disturbance recipients of d_2.

Lemma 6.3 *Let $\mathcal{K}$ be a decision network skeleton and d be an SD candidate decision node. Let X_{II} be the set of random and decision nodes in the downstream set $Y_{II}(d, \mathcal{K})$.*

1. *For any $c \in X_{II}$, $\pi_c \subseteq X_{II} \cup \pi_d$.*
2. *For any $c_2 \in X_{II}$ and any $c_1 \in \pi_d$, if $\mathcal{K}$ contains the arc $c_2 \rightarrow c_1$, then c_2 and c_1 are both random nodes. So are the ancestors of c_2 in X_{II}.*

Proof: The first part follows immediately from the definition of downstream set.

We now prove the second part. First of all, c_1 cannot be a value node since value nodes have no children and c_1 has the child d. Also since π_d does not separate d from c_2 and c_2 is a parent of c_1, c_1 can not be decision node either, for this would contradict the fact that d is an SD candidate node of $\mathcal{K}$. Therefore c_1 must be a random node.

Following a similar line of reasoning, one can show that c_2 and its ancestors in X_{II} are all random nodes. $\square$

Corollary 6.1 *Suppose d is an SD candidate decision node of a decision network skeleton. Then all the disturbance nodes and disturbance recipients of d are random nodes.* □

6.1.4 Tail-smoothing skeletons

Let d be an SD candidate node in a decision network skeleton $\mathcal{K}$. Suppose $\mathcal{K}$ is not smooth at d. This subsection presents a procedure for smoothing $\mathcal{K}$ at d.

A *leaf disturbance node of* d is a disturbance node of d such that none of its children are disturbance nodes of d.

Let c be a leaf disturbance node of d. Let $c \rightarrow c_1$, $c \rightarrow c_2$, ..., $c \rightarrow c_m$ be all the disturbance arcs emitting from c. An disturbance arc $c \rightarrow c_i$ is *the most senior* if there is no other disturbance arc $c \rightarrow c_j$ such that c_j is an ancestor of c_i.

Since $\mathcal{K}$ is acyclic, if there are disturbance arcs emitting from c, then one of them must be the most senior. Since c is a leaf disturbance node of d, the most senior disturbance arc $c \rightarrow c_i$ is reversible.

Procedure TAIL-SMOOTHING-K($\mathcal{K}, d$)

- Input: $\mathcal{K}$ — an SDDN skeleton,
  d — an SD candidate of $\mathcal{K}$.
- Output: An SDDN skeleton that is smooth at d.

While1 there are disturbance nodes of d, find a leaf disturbance node c, break ties arbitrarily.

while2 there are disturbance arcs of d emitting from c, pick and reverse a most senior one, break ties arbitrarily. **end-while2**

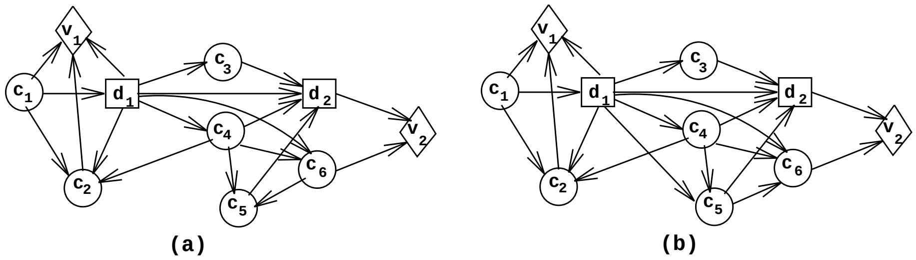

Figure 6.21: The application of TAIL-SMOOTHING-K to the decision network skeleton in Figure 6.20 with the input candidate node being d_2: (a) after reversing $c_6 \rightarrow c_4$, (b) after reversing $c_6 \rightarrow c_5$.

end-while1.

As an example, let $\mathcal{K}$ be the decision network skeleton in Figure 6.20. Figure 6.21 shows the working of TAIL-SMOOTHING-K($\mathcal{K}, d_2$). The node c_6 is a leaf disturbance node of d_2. There are two disturbance arcs emitting from c_6: $c_6 \rightarrow c_4$ and $c_6 \rightarrow c_5$, among which $c_6 \rightarrow c_4$ is the most senior. So, the arc $c_6 \rightarrow c_4$ is first reversed, resulting in the decision network skeleton in Figure 6.21 (a). The arc $c_6 \rightarrow c_5$ is reversed thereafter, resulting in the decision network skeleton in Figure 6.21 (b), which is smooth at d_2.

Proposition 6.2 *The procedure TAIL-SMOOTHING-K terminates and is correct.*

A proof can be found at the end of this section.

Let $\mathcal{K}'$ be the output decision network skeleton of TAIL-SMOOTHING-K($\mathcal{K}, d$). For any disturbance recipient r of d (in $\mathcal{K}$), the set of parents π'_r of r in $\mathcal{K}'$ is different from the set of parents π_r of r in $\mathcal{K}$. In our example, $\pi'_{c_5} = \{d_1, c_4\}$, while $\pi_{c_5} = \{c_4, c_6\}$. The following lemma gives us some idea about what nodes π'_r consists of. The lemma is useful in presenting EVALUATE1.

Lemma 6.4 *Let π'_r and π_r be as in the previous paragraph and let π_d be the set of parents of d in $\mathcal{K}$. Then $\pi_r \cap \pi_d \subseteq \pi'_r \subseteq \pi_d$, and each $x \in \pi'_r - \pi_r$ is not a descendent of r in $\mathcal{K}$.*

A proof can be found at the end of this section.

6.1.5 Tail smoothing decision networks

The arc reversals in TAIL-SMOOTHING-K are at the level of skeleton. There are no numerical computations whatsoever. The following algorithm for smooth a decision network at a decision node is the same as TAIL-SMOOTHING-K, except now numerical computations are involved.

Procedure TAIL-SMOOTHING($\mathcal{N}, d$)

- Input: $\mathcal{N}$ — an SDDN ,
  d — an SD candidate of $\mathcal{N}$.
- Output: An equivalent SDDN that is smooth at d.

While1 there are disturbance nodes of d, find a leaf disturbance node c, break ties arbitrarily.

> **while2** there are disturbance arcs of d emitting from c, pick and reverse a most senior one, break ties arbitrarily. **end-while2**

end-while1.

As an example, let $\mathcal{N}$ be a decision network over the skeleton in Figure 6.20. Consider the working of TAIL-SMOOTHING($\mathcal{N}, d_2$). As in the case of TAIL-SMOOTHING-K, the arc $c_6 \rightarrow c_4$ is first reversed, resulting in a decision network with underlying skeleton as in Figure 6.21 (a). The conditional probabilities of c_4 and c_6 in the resulting network are as follows:

$$P(c_4|d_1) = \sum_{c_6} P(c_4|d_1, c_6)P(c_6), \tag{6.44}$$

$$P(c_6|d_1, c_4) = \frac{P(c_4|d_1, c_6)P(c_6)}{\sum_{c_6} P(c_4|d_1, c_6)P(c_6)}. \tag{6.45}$$

Then the arc $c_6 \rightarrow c_5$ is reversed, resulting in a decision network with underlying skeleton as in Figure 6.21 (b). The conditional probability of c_5 in the resulting network is as follows:

$$P(c_5|d_1, c_4) = \sum_{c_6} P(c_5|c_4, c_6)P(c_6|d_1, c_4) = \frac{\sum_{c_6} P(c_5|c_4, c_6)P(c_4|d_1, c_6)P(c_6)}{\sum_{c_6} P(c_4|d_1, c_6)P(c_6)}. \tag{6.46}$$

Note that no complexity analysis of TAIL-SMOOTHING is carried out because it will be used only in proofs, never in evaluation algorithm.

6.1.6 Smoothing non-smooth SDDN's

This subsection is for the benefit of Chapter 9; it gives an algorithm that smooths non-smooth SDDN's.

Procedure SMOOTHING($\mathcal{N}$)

- Input: $\mathcal{N}$ — an SDDN.
- Output: A smooth SDDN that is equivalent to $\mathcal{N}$.

If $\mathcal{N}$ contains no decision node, **return** $\mathcal{N}$.

Else

1. Find an SD candidate decision node d of $\mathcal{N}$.
2. Call TAIL-SMOOTHING($\mathcal{N}, d$). Let Let $\mathcal{N}'$ denote the resulting decision network.
3. In $\mathcal{N}'$, treat d as a random node[1]. (Thus $\mathcal{N}'$ contains one less decision nodes than $\mathcal{N}$.) Recursively call SMOOTHING($\mathcal{N}'$).

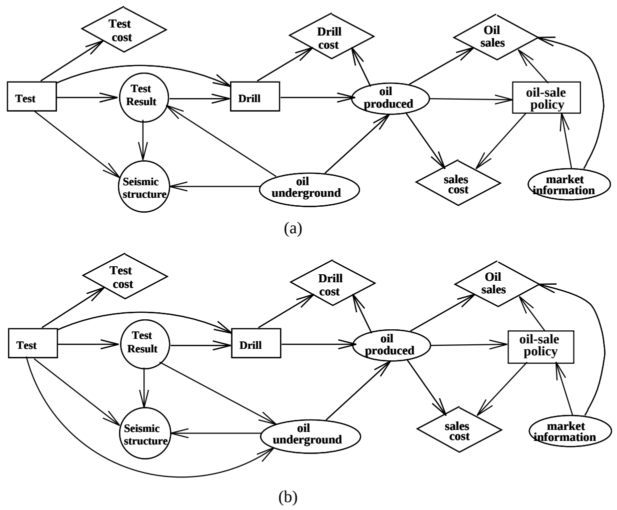

Figure 6.22: The effects of applying SMOOTHING to the SDDN in Figure 1.7: (a) after the arc from `seismic-structure` to `test-result` is reversed, (b) the final SDDN, which is smooth.

As an example, consider the SDDN in Figure 1.7. The network is smooth at `oil-sale-policy`, so SMOOTHING does nothing in the first recursion. In the second recursion, `oil-sale-policy` is treated as a random node, rendering `drill` an SD candidate node. The SDDN is not smooth at `drill`. So TAIL-SMOOTHING will enter its while loops. There is only one leaf disturbance node of `drill`, namely `seismic-structure`. Thus the arc from `seismic-structure` to `test-result` is reversed, introducing an arc from `oil-underground` to `test-result` and an arc from `test` to `seismic-structure`. See Figure 6.22 (a). Now, `oil-underground` becomes a leaf disturbance node of `drill`. The arc from `oil-underground` to `test-result` is reversed, introducing an arc from `test` to `oil-underground`. The final SDDN is shown in Figure 6.22 (b), which is smooth.

Note that no complexity analysis of SMOOTHING is carried out because it will be used only in proofs, never in evaluation algorithm.

[1]The decision node d is treated as a random node only within the scope of SMOOTHING. It is treated again as a decision after the termination of SMOOTHING.

Theorem 6.1 *The procedure SMOOTH terminates and is correct.*

A proof will be provided in the next subsection.

6.1.7 Proofs

Proof of Proposition 6.2: To prove that the procedure TAIL-SMOOTHING-K terminates, we need to show that the procedure does not get trapped in the while-loops. The procedure will eventually exit the inner while-loop, because the number of disturbance arcs emitting from the leaf disturbance node c is reduced by one during each execution of the loop.

The procedure will also exit the outer while-loop since reversing all the disturbance arcs emitting from a leaf disturbance node c does not produce any new disturbance nodes, and c is no longer a disturbance node thereafter. Therefore the number of disturbance nodes is reduced by one during each execution of the outer while-loop. Since there are only a finite number of disturbance nodes, the procedure will eventually leave the outer while-loop.

TAIL-SMOOTHING-K changes neither the downstream set nor the upstream set of π_d. So, the resulting decision network is also stepwise-decomposable.

Since the procedure exits the outer while-loop only when there are no more disturbance nodes of d, the resulting network produced by the procedure is smooth at d. The proposition is proved. $\Box$

Proof of Lemma 6.4: Let $\pi_r(t)$ be the set of parents of r at time step t during the execution of TAIL-SMOOTHING($\mathcal{K}, d$). We show by induction on t that (1) $\pi_r \cap \pi_d \subseteq \pi_r(t)$, (2) $\pi_r(t) \cap Y_I(\mathcal{K}, d) = \emptyset$, and (3) each $x \in (\pi_r(t) \cap \pi_d) - \pi_r$ is not a descendant of r in $\mathcal{K}$.

At the beginning, $\pi_r(0) = \pi_r$. So (1) and (3) are trivially true. (2) is true because at least one node in π_r is in $Y_{II}(\mathcal{K}, d)$, since r is a disturbance recipient of d. Hence none of the nodes in π_r can be in the upstream set Y_I.

Suppose (1-3) are true at time step t. Consider time step t+1. Suppose at this time step, the disturbance arc reversed is $c \rightarrow r'$. If $r' \neq r$, then $\pi_r(t+1) = \pi_r(t)$, hence (1-3) are true. When $r' = r$, let x be a node in $\pi_r(t+1) - \pi_r(t)$. Then x must be parent of c that is not a parent of r. Since the arc $c \rightarrow r$ is reversible, x cannot be a descendant of r. So x does not lead to the violation of (3). Since c is in the downstream set Y_{II} of π_d, x can only be either in π_d or in Y_{II}. In both case, x does not lead to the violation of any of (2). By the definition of arc reversal, $\pi_r(t) - \pi_r(t+1) = \{c\}$. Again because c is in Y_{II}, (1) remains true. In other words, (1-3) are true for the case of $t+1$. Consequently, (1-3) are true for all t's.

At the end of the execution of TAIL-SMOOTHING$(\mathcal{K}, d)$, $\pi_r(t) = \pi'_r$. Since $\mathcal{K}'$ is smooth at d, none of the nodes of π'_r are in the downstream set Y_{II}, hence $\pi'_r \cap \pi_d = \pi'_r$. Consequently, it follows from (1-3) that $\pi_r \cap \pi_d \subseteq \pi'_r \subseteq \pi_d$, and each $x \in \pi'_r - \pi_r$ is ancestor of r in $\mathcal{K}$. $\Box$

We prove the correctness of SMOOTHING by induction on the number of decision nodes. When there are no decision nodes, SMOOTHING is trivially correct. Suppose SMOOTHING is correct in the case of $k-1$ decision nodes. Now consider the case of k decision nodes.

Let d be an SD candidate node of $\mathcal{N}$. Let $\mathcal{N}'$ be the output network of TAIL-SMOOTHING$(\mathcal{N}, d)$. According to Proposition 6.2, d remains an SD candidate node in $\mathcal{N}'$ and $\mathcal{N}'$ is smooth at d and equivalent to $\mathcal{N}$.

Treating d as a random node in $\mathcal{N}'$, we let d' be an SD candidate node of $\mathcal{N}'$. Let $\mathcal{N}^*$ be the output network of SMOOTHING$(\mathcal{N}')$. Then $\mathcal{N}^*$ is equivalent to $\mathcal{N}'$ with d regarded as a random node. Consequently, $\mathcal{N}^*$ is also equivalent to $\mathcal{N}'$ when d is treated as a decision node.

By the induction hypothesis, $\mathcal{N}^*$ is a smooth and stepwise-decomposable when d is regarded as a random node. By proposition 5.1, what remains to be proved is that when treated as a decision node, d is an SD candidate node of $\mathcal{N}^*$ and $\mathcal{N}^*$ is smooth at d.

Let $\mathcal{N}''$ be the output network of TAIL-SMOOTHING($\mathcal{N}'$, d'). Since $\mathcal{N}'$ is smooth at d, the tail of $\mathcal{N}'$ w.r.t d is not touched in the execution of TAIL-SMOOTHING($\mathcal{N}'$, d'). Thus, d is an SD candidate node in $\mathcal{N}''$ and $\mathcal{N}''$ is smooth at d.

Suppose d'' becomes an SD candidate node of $\mathcal{N}''$ if both d and d' are treated as random nodes. Let $\mathcal{N}'''$ be the output network of TAIL-SMOOTHING($\mathcal{N}''$, d''). Repeating the argument in previous paragraph, we can show that d is an SD candidate node in $\mathcal{N}'''$ and $\mathcal{N}'''$ is smooth at d. Continuing the argument, we can eventually show that d is an SD candidate node in $\mathcal{N}^*$ and $\mathcal{N}^*$ is smooth at d. The theorem is proved. $\Box$

6.2 Tail and body

The procedure TAIL-SMOOTHING suggests the following approach for evaluating a non-smooth SDDN $\mathcal{N}$: Find an SD candidate node d, use TAIL-SMOOTHING to smooth $\mathcal{N}$ at d, decompose $\mathcal{N}$ at d into a tail and a body, find an optimal decision function for d in the tail, and repeat the process for the body. An disadvantage of this approach is that TAIL-SMOOTHING demands a series of arc reversals, which may be inefficient (Shenoy 1992, Ndilikilikesha 1991). The motivation behind EVALUATE1 is to avoid arc reversals. This section paves the way to EVALUATE1.

Let $\mathcal{N}$ be a decision network and d an SD candidate node in $\mathcal{N}$. In Sections 5.1 and 4.1, we have defined the concepts of tail (or downstream component) and body (or upstream component) for the case when $\mathcal{N}$ is smooth at d. In this section, we extend the concepts of tail and body to the case when $\mathcal{N}$ is not smooth at d.

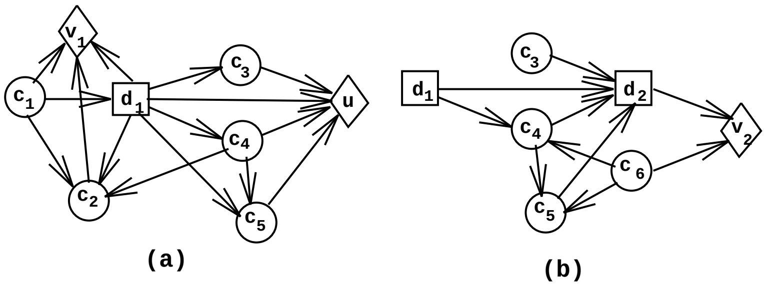

Figure 6.23: Tail and body for the non-smooth decision network skeleton in Figure 6.20 [FINAL CHECK]: (a) shows its body w.r.t d_2 and (b) shows its tail w.r.t d_2.

6.2.1 Tail and body at the level of skeleton

When $\mathcal{K}$ is smooth at d, there are no disturbance recipients of d. When $\mathcal{K}$ is not smooth at d, some of the nodes in π_d are disturbance recipients. Disturbance recipients require special attention when extending the definition of tail and body to the non-smooth case.

Suppose d is an SD candidate node of $\mathcal{K}$. The *tail* of $\mathcal{K}$ w.r.t d, denoted by $\mathcal{K}_{II}(d, \mathcal{K})$ or simply by $\mathcal{K}_{II}$, is the decision network skeleton obtained from $\mathcal{K}$ by restricting $\mathcal{K}$ onto $Y_{II} \cup \pi_d$ and removing all those arcs among nodes in π_d that do not point at disturbance recipients of d.

As an example, let $\mathcal{K}$ be the decision network skeleton in Figure 6.20. Figure 6.23 (b) shows the tail of $\mathcal{K}$ w.r.t d_2. The restriction of $\mathcal{K}$ onto $Y_{II}(d_2, \mathcal{K}) \cup \pi_{d_2}$ contains the following three arcs among nodes in π_{d_2}: $d_1 \rightarrow c_3$, $d_1 \rightarrow c_4$, and $c_4 \rightarrow c_5$. The arc $d_1 \rightarrow c_3$, which is removed because c_3 is not a disturbance recipient of d_2. On the other hand, the arcs $d_1 \rightarrow c_4$ and $d_4 \rightarrow c_5$ are retained because both c_4 and c_5 are disturbance recipients of d_2.

In the definition of tail, why do we need to handle disturbance recipients of a decision node d in a different manner from other parents of d? Consider, for instance, the disturbance recipient c_4 of d_2 in Figure 6.20. The conditional probability $P(c_4|d_1, c_6)$ of c_4

involves the node c_6. Since c_6 is in the downstream set $Y_{II}(d, \mathcal{K})$, $P(c_4|d_1, c_6)$ is placed in the tail (Subsection 6.2.2). Consequently, the arc $d_1 \rightarrow c_4$ has to be retained. On the other hand, c_3 is not a disturbance recipient of d_2. Its conditional probability $P(c_3|d_1)$ does not involve nodes in the downstream set Y_{II}, is hence placed in the body (see Subsection 6.2.2). So, we delete the arc $d_1 \rightarrow c_3$ from the tail.

To extend the concept of body to the non-smooth case, let $\mathcal{K}'$ be the output decision network skeleton of TAIL-SMOOTHING-K$(\mathcal{K}, d)$. Since $\mathcal{K}'$ is smooth at d, its body $\mathcal{K}'_I$ w.r.t d is defined (Sections 4.1 and 5.1). We define the *body of* $\mathcal{K}$ w.r.t d to simply be the body $\mathcal{K}'_I$ of $\mathcal{K}'$ w.r.t d, and we denote it by $\mathcal{K}_I(d, \mathcal{K})$ or simply by $\mathcal{K}_I$.

As an example, let $\mathcal{K}$ be the decision network skeleton in Figure 6.20. Figure 6.21 (b) shows the output decision network skeleton of TAIL-SMOOTHING-K$(\mathcal{K}, d_2)$, from which we obtain the body $\mathcal{K}_I$ of $\mathcal{K}$ w.r.t d_2. $\mathcal{K}_I$ is as shown in Figure 6.23 (a).

The reader is encouraged to verify that the general definitions of tail and body (at the level of skeleton) given in this subsection are consistent with the corresponding definitions for the smooth case given Sections 4.1 and 5.1. In doing so, s/he needs to keep in mind that in the smooth case there are no disturbance recipients.

6.2.2 Tail of decision networks

Having defined tail at the level of skeleton, we can now define tail for decision networks by providing the necessary numerical information. Suppose d is an SD candidate node in a decision network $\mathcal{N}$. Let $\mathcal{K}$ be the underlying skeleton. The *tail* of $\mathcal{N}$ w.r.t d, denoted by $\mathcal{N}_{II}(d, \mathcal{N})$ or simply by $\mathcal{N}_{II}$, is a semi-decision network over $\mathcal{K}_{II}(d, \mathcal{K})$. The value functions of all the value nodes in $\mathcal{N}_{II}$ remain the same as in $\mathcal{N}$. The conditional probabilities of random nodes outside π_d also remain the same as in $\mathcal{N}$. Since d is an SD candidate node, Corollary 6.1 assures us that the disturbance recipients of d are all random nodes. The conditional probabilities of the disturbance recipients of d again remain the same as in $\mathcal{N}$. The nodes in $S =_{def}\{x{\in}\pi_d \mid x \textit{ is not disturbance recipient of } d\}$ are all viewed as root random nodes without prior probabilities.

As an example, let $\mathcal{N}$ be a decision network over the skeleton shown in Figure 6.20. Then the tail $\mathcal{N}_{II}(d_2, \mathcal{N})$ is a semi-decision network over the skeleton shown in Figure 6.23 (b). $\mathcal{N}_{II}$ contains conditional probabilities $P(c_4|d_1, c_6)$, $P(c_5|c_4, c_6)$, and $P(c_6)$ of random nodes c_4, c_5, and c_6, which are respectively the same as the conditional probabilities of c_4, c_5, and c_6 in $\mathcal{N}$. $\mathcal{N}_{II}$ also contains a value function $\mu_{v_2}(d_2, c_6)$ of v_2, which is the same as the value function of v_2 in $\mathcal{N}$. The root random node c_3 does not have prior probability. The node d_1 is treated as a root random node without prior probability.

Let X_{II} be the set of random and decision nodes in the downstream set $Y_{II}(d, \mathcal{N})$. Let $P_0(X_{II}, \pi_d)$ be the product of all the conditional probabilities in $\mathcal{N}_{II}$. For any subset B of $X_{II}{\cup}\pi_d$, $P_0(B)$ is obtained from $P_0(X_{II}, \pi_d)$ by summing out the variables in $X_{II}{\cup}\pi_d{-}B$. Define the *evaluation functional* $e(d, \pi_d)$ of $\mathcal{N}_{II}$ as follows:

$$e(d, \pi_d) = \frac{1}{P_0(\pi_d, d)} \sum_{v \in V_{II}} \sum_{\pi_v - \pi_d \cup \{d\}} P_0(\pi_v, \pi_d, d)\mu_v(\pi_v), \tag{6.47}$$

where V_{II} stands for the set of value nodes in $\mathcal{N}_{II}$.

To continue our example, $Y_{II}(d_2, \mathcal{N}) = \{d_2, c_6, v_2\}$. So $X_{II} = \{d_2, c_6\}$. $P_0(X_{II}, \pi_{d_2})$ is given by

$$P_0(X_{II}, \pi_{d_2}) = P_0(d_2, c_6, d_1, c_3, c_4, c_5) = P(c_4|d_1, c_6)P(c_5|c_4, c_6)P(c_6).$$

So, the evaluation functional e of $\mathcal{N}_{II}$ is given by

$$e(d_2, d_1, c_3, c_4, c_5) = \frac{1}{\sum_{c_6} P(c_4|d_1, c_6)P(c_5|c_4, c_6)P(c_6)} \sum_{c_6} P(c_4|d_1, c_6)P(c_5|c_4, c_6)P(c_6)\mu_{v_2}(d_2, c_6).$$

A note about consistency in the definition of evaluation functional. According to the note at the end of Section 3.1, when $\mathcal{N}$ is smooth at d, $P_0(X_{II}, \pi_d)$ is the conditional probability $P(X_{II}-\{d\}|\pi_d, d)$. Thus $P_0(\pi_d, d) = \sum_{X_{II}-\{d\}} P(X_{II}-\{d\}|\pi_d, d) = 1$. Consequently, when $\mathcal{N}$ is smooth at d, the definition of evaluation functional given here is the same as the definition given in Section 5.5.

Theorem 6.2 *Suppose d is an SD candidate node in a decision network $\mathcal{N}$. Let $e(d, \pi_d)$ be the evaluation functional of the tail $\mathcal{N}_{II}(d, \mathcal{N})$. The optimal decision functions δ_d^o of d can be found through*

$$\delta_d^o(\beta) = arg\ max_{\alpha\in\Omega_d} e(d{=}\alpha, \pi_d{=}\beta), \forall\beta \in \Omega_{\pi_d}. \tag{6.48}$$

A proof will be provided in Section 6.4.

6.2.3 Body of decision networks

As in the case of tail, the body of a decision network $\mathcal{N}$ w.r.t to an SD candidate node d is obtained from the body of its underlying skeleton w.r.t d by providing the necessary numerical information. Let $\mathcal{K}$ be the skeleton underlying $\mathcal{N}$. The *body* of $\mathcal{N}$ w.r.t d, denoted by $\mathcal{N}_I(d, \mathcal{N})$ or simply by $\mathcal{N}_I$, is a semi-decision network over $\mathcal{K}_I(d, \mathcal{K})$. The value functions of all the value nodes other than u remain the same as in $\mathcal{N}$. The value function μ_u of the tail-value node u is defined by

$$\mu_u(\pi_d{=}\beta) = max_{\alpha\in\Omega_d} e(d{=}\alpha, \pi_d{=}\beta), \forall\beta \in \Omega_{\pi_d}. \tag{6.49}$$

The conditional probabilities of random nodes that are not disturbance recipients of d also remain the same as in $\mathcal{N}$.

What remain to be provided are the conditional probabilities of the disturbance recipients of d. Let us first note that a disturbance recipient of d has different parents in the body $\mathcal{K}_I$ from in the original skeleton $\mathcal{K}$. For example, the parents of c_5 in Figure 6.20 are c_4 and c_6, while in Figure 6.23 (a) the parents of c_5 are d_1 and c_4. For any disturbance recipient r of d, let π_r^I be the set of the parents of r in $\mathcal{K}_I$. In Figure 6.23 (a), for instance, $\pi_{c_5}^I = \{d_1, c_4\}$, while $\pi_{c_5} = \{c_4, c_6\}$.

Let $\mathcal{N}'$ be the output decision network of TAIL-SMOOTHING($\mathcal{N}, d$), and let r be a disturbance recipient of d. We want to define the conditional probability $P(r|\pi_r^I)$ of r in $\mathcal{N}_I$ to be the conditional probability of r in $\mathcal{N}'$, but the sake of computational efficiency we do not wish to explicitly compute $\mathcal{N}'$. The following definition resolves our dilemma.

The conditional probability $P(r|\pi_r^I)$ of r in $\mathcal{N}_I$ is defined by

$$P(r|\pi_r^I) =_{def} \frac{P_0(\pi_r^I, r)}{P_0(\pi_r^I)}, \tag{6.50}$$

where P_0 is as in the previous subsection.

We shall show in Section 6.4 that $P(r|\pi_r^I)$ as defined by equation (6.50) is indeed the conditional probability of r in $\mathcal{N}'$. Here is an example. Recall that c_4 and c_5 are the only two disturbance recipients of d_2 in Figure 6.20. Let us compute the conditional probabilities of c_4 and c_5 in Figure 6.23 (a).

$$\begin{aligned} P(c_4|\pi_4^I) = P(c_4|d_1) &= \frac{P_0(c_4, d_1)}{P_0(d_1)} \\ &= \frac{\sum_{c_6} P(c_4|d_1, c_6)P(c_6)}{\sum_{c_6, c_4} P(c_4|d_1, c_6)P(c_6)} = \sum_{c_6} P(c_4|d_1, c_6)P(c_6), \end{aligned} \tag{6.51}$$

$$P(c_5|\pi_5^I) = P(c_5|d_1, c_4) = \frac{P_0(c_5, d_1, c_4)}{P_0(d_1, c_4)} = \frac{\sum_{c_6} P(c_4|d_1, c_6)P(c_5|c_4, c_6)P(c_6)}{\sum_{c_6} P(c_4|d_1, c_6)P(c_6)}. \tag{6.52}$$

A comparison between equations (6.51) and (6.52) with equations (6.44) and (6.46) reveals that the conditional probabilities of c_4 and c_5 obtained through equation (6.50) are indeed the same as the conditional probabilities of c_4 and c_5 in $\mathcal{N}'$.

According to Lemma 6.4, $\pi_r^I \subseteq \pi_d$ for any disturbance recipient r of d. This observation about π_r^I leads to the following formula for computing $P(r|\pi_r^I)$:

$$P(r|\pi_r^I) = \frac{\sum_{\pi_d - \pi_r \cup \{r\}} P_0(\pi_d)}{\sum_{\pi_d - \pi_r \cup \{r\}} P_0(\pi_d)}. \tag{6.53}$$

In words, in order compute $P(r|\pi_r^I)$, we can compute $P_0(\pi_d)$ from $P_0(X_{II}, \pi_d)$ by summing out the nodes in $X_{II} \cup \pi_d - \pi_d$ and obtain $P(r|\pi_r^I)$ through equation (6.53).

Theorem 6.3 *Suppose d is an SD candidate node in a decision network $\mathcal{N}$. Then the optimal decision functions for decision nodes other than d are the same in $\mathcal{N}$ as in the body $\mathcal{N}_I(d, \mathcal{N})$.*

A proof will be provided in Section 6.4.

6.3 The procedure EVALUATE1

Theorems 6.2 and 6.3 lead to the following procedure for evaluating SDDN's, smooth or non-smooth.

Procedure EVALUATE1($\mathcal{N}$):

- Input: $\mathcal{N}$ — an SDDN, smooth or non-smooth.
- Output: An optimal policy and the optimal expected value of $\mathcal{N}$.

If there are no decision nodes, call N-EVALUATE($\mathcal{N}$) to compute the expected value, and stop.

Else

1. Find an SD candidate node d,
2. Compute the tail $\mathcal{N}_{II}$ of $\mathcal{N}$ w.r.t d. Let P_0 denote the product of all the conditional probabilities in $\mathcal{N}_{II}$.
   (a) Compute the marginal potentials $P_0(\pi_d)$ and $P_0(\pi_d, d)$, and the marginal potential $P_0(\pi_d, d, \pi_v)$ for each value node v in $\mathcal{N}_{II}$.

(b) Compute the evaluation functional $e(d, \pi_d)$ by

$$e(d, \pi_d) = \frac{1}{P_0(\pi_d, d)} \sum_{v \in V_{II}} \sum_{\pi_v - \pi_d \cup \{d\}} P_0(\pi_v, \pi_d, d)\mu_v(\pi_v). \qquad (6.54)$$

where V_{II} is the set of value nodes in $\mathcal{N}_{II}$.

(c) Compute an optimal decision function δ_d^o of d by

$$\delta_d^o(\beta) = arg\ max_{\alpha \in \Omega_d} e(d{=}\alpha, \pi_d{=}\beta), \forall \beta \in \Omega_{\pi_d}. \qquad (6.55)$$

(d) Compute the body $\mathcal{N}_I$ of $\mathcal{N}$ w.r.t d (equation (6.53) is used here).

3. Recursively call EVALUATE1($\mathcal{N}_I$).

What is the running time of EVALUATE1? Let n be the total number of nodes in $\mathcal{N}$, k be the number of decision nodes, a be the number of arcs, e be the number of edges in the moral graph of $\mathcal{N}$. According to the complexity analysis of TEST-STEPWISE-DECOMPOSABILITY, the time EVALUATE1 spends on finding candidate nodes and computing tails and bodies is $O(k^2(n + e))$.

If we use the clique tree propagation approach to compute the marginal potentials in step (a), we need only to traverse the clique tree twice. If there are l cliques and the maximum number of nodes in a clique is q, the runing time is $O(l\lambda^q)$, where λ stands for the maximum number of values a variable can assume. So, EVALUATE1 spends $O(kl\lambda^q)$ time computing marginal potentials.

The time for computing the evaluation functional and optimal decision functions from the evaluation functional is dominated by the time for computing marginal potentials, except for the numerical divisions. For each (candidate) node d, the factor $P_0(\pi_d, d)$ is divided from an expression to arrive at the evaluation functional $e(\pi_d, d)$. Numerical divisions also happen once for each disturbance recipient in the computation of body[2].

[2]This can be avoided via a subtle technical trick.

6.4 Correctness of EVALUATE1

To prove the correctness of the procedure EVALUATE1, it suffices to show that Theorems 6.2 and 6.3 are true.

Proof of Theorem 6.2: Let $\mathcal{N}'$ be output network of TAIL-SMOOTHING($\mathcal{N}$, d). Then d is also an SD candidate node of $\mathcal{N}'$ and $\mathcal{N}'$ is smooth at d.

Let P'_0 be the product of the conditional probabilities in the tail $\mathcal{N}'_{II}(d, \mathcal{N}')$. According the Theorem 5.3, optimal decision functions δ^o_d for d can be found through

$$\delta^o_d(\beta) = arg\ max_{\alpha \in \Omega_d} e'(d{=}\alpha, \pi_d{=}\beta), \forall \beta \in \Omega_{\pi_d}, \tag{6.56}$$

where the evaluation functional $e'(d, \pi_d)$ of $\mathcal{N}'_{II}$ is given by

$$e(\alpha, \beta) \;=\; \sum_{v \in V_{II}} \sum_{\pi_v - \pi_d \cup \{d\}} P'_0(\pi_v, \pi_d, d)\mu_v(\pi_v) \;\; \forall \alpha \in \Omega_d, \forall \beta \in \Omega_{\pi_d}, \tag{6.57}$$

where V_{II} stands for the set of value node in $\mathcal{N}'_{II}$.

Let P_0 be the product of all the conditional probabilities in the tail $\mathcal{N}_{II}(d, \mathcal{N})$. By Lemma 6.2, we conclude that arc reversals do not change joint probabilities. Hence they do not change conditional probabilities either. Consequently, for each value node v in $\mathcal{N}_{II}$ (or in $\mathcal{N}'_{II}$) we have

$$P_0(\pi_v|\pi_d, d) = P'_0(\pi_v|\pi_d, d).$$

Since $\mathcal{N}'$ is smooth at d, we have

$$P'_0(\pi_v|\pi_d, d) = P'_0(\pi_v, \pi_d, d).$$

Therefore

$$P'_0(\pi_v, \pi_d, d) = P_0(\pi_v|\pi_d, d). \tag{6.58}$$

Consequently,

$$\begin{aligned} e'(d, \pi_d) &= \sum_{v \in V} \sum_{\pi_v - \pi_d \cup \{d\}} P_0(\pi_v | \pi_d, d) \mu_v(\pi_v) \\ &= \frac{1}{P_0(\pi_d, d)} \sum_{v \in V_{II}} \sum_{\pi_v} P_0(\pi_v, \pi_d, d) \mu_v(\pi_v) \\ &= e(d, \pi_d), \end{aligned} \tag{6.59}$$

where $e(d, \pi_d)$ is the evaluation functional of $\mathcal{N}_{II}$. This proves Theorem 6.2. □

Proof of Theorem 6.3: Let $\mathcal{N}'$ be the output network of TAIL-SMOOTHING($\mathcal{N}$, d). Then d is also an SD candidate node of $\mathcal{N}'$ and $\mathcal{N}'$ is smooth at d. Let π'_x be the set of parents of a node x in $\mathcal{N}'$ and let $P'(c|\pi'_c)$ be the conditional probability of a random node c in $\mathcal{N}'$.

Let $\mathcal{K}$ be the skeleton underlying $\mathcal{N}$. Recall that in the definition of $\mathcal{N}_I$, we executed TAIL-SMOOTHING-K($\mathcal{K}, d$). Suppose the ties were broken in the same way in both the execution of TAIL-SMOOTHING-K($\mathcal{K}, d$) and the execution TAIL-SMOOTHING($\mathcal{N}$, d). Then for any node x in $\mathcal{N}_I$ other than the tail-value node, $\pi_x^I = \pi'_x$. In particular, for any disturbance recipient r of d in $\mathcal{N}$, $\pi_r^I = \pi'_r$.

Because of Theorem 5.3, it suffices to show that the body $\mathcal{N}_I(d, \mathcal{N})$ of $\mathcal{N}$ is the same as the body $\mathcal{N}'_I(d, \mathcal{N}')$ of $\mathcal{N}'$.

First of all, because of equation (6.59) the value function of the tail-value node in $\mathcal{N}'_I$ is the same as the value function of the tail-value node in $\mathcal{N}_I$.

What remains to be proved is that for any disturbance recipient d of d,

$$P'(r|\pi'_r) = P(r|\pi_r^I). \tag{6.60}$$

Let R be the set of all the disturbance recipients of d in $\mathcal{N}$. Let C_{II} be the set of random nodes in the downstream set $Y_{II}(d, \mathcal{N})$. Consider the product of all the conditional probabilities of nodes in $C_{II} \cup R$. According to Lemma 6.2, this product is not changed by the arc reversals in TAIL-SMOOTHING(d, $\mathcal{N}$). Thus

$$\prod_{c \in C_{II} \cup R} P(c|\pi_c) = \prod_{c \in C_{II} \cup R} P'(c|\pi'_c).$$

Summing out all the nodes in C_{II} from both sides of the equation, we get

$$P_0(\pi_d) = \prod_{c \in R} P'(c|\pi'_c).$$

Thus for any $r \in R$, we have

$$P'(r|\pi'_r) = \frac{\sum_{\pi_d - (\{r\} \cup \pi'_r)} P_0(\pi_d)}{\sum_{\pi_d - \pi'_r} P_0(\pi_d)} = P(r|\pi^I_r).$$

Theorem 6.3 is therefore proved. $\square$

6.5 Comparison to other approaches

Influence diagrams are special SDDN's and hence can be evaluated by EVALUATE1. This section compares EVALUATE1 with previous approaches for evaluating influence diagrams. We identify a list of desirable properties and examine EVALUATE1 and each of the previous approaches with regard to those properties.

6.5.1 Desirable properties of evaluation algorithms

A list of desirable properties of algorithms for evaluating influence diagrams is given in the first row of Table 6.1. Due explanations follow.

Table 6.1 Comparisons among approachs for evaluating influence diagrams.

	facilitating arc removal	divide and conquer	separating BN inference	multiple value nodes	reversing arcs
EVALUATE1	yes	yes	yes	yes	no
Shachter 86	no	no	no	no	yes
Ndilikilikesha 92	no	no	no	no	no
Tatman and Shachter 90	no	no	no	yes	yes
Shachter 88	no	no	yes	no	no
Shenoy 90	n/a	no	no	no	no
Shenoy 92	n/a	no	no	yes	no

Facilitating the pruning of removable arcs

In a decision network, an arc into a decision node is *removable* if its removal does not affect the optimal expected value of the network. In Chapter 7, we shall present an algorithm that prunes from an influence diagram all the removable arcs that can be graphically identified.

There are a couple of advantages to pruning removable arcs: it results in a simpler network, and it reduces the sizes of the decision tables. Thus a desirable property for an evaluation algorithm to possess is to be able to facilitate the pruning of removable arcs.

It will be shown in Chapter 7 that pruning graphically identifiable removable arcs from influence diagrams results in SDDN's. Since EVALUATE1 is designed for evaluating SDDN's, it facilitates the pruning of removable arcs from influence diagrams.

Divide and conquer

It is desirable to decompose, prior to evaluation, an influence diagram into (overlapping) portions such that each portion corresponds to a decision node and optimal decision functions of a decision node can be computed in its corresponding portion. This is an application of the standard divide and conquer idea.

Suppose the influence diagram to be evaluated has been put through a preprocessing stage such that removable arcs have been pruned. Let $\mathcal{N}$ be the resulting SDDN. The procedure EVALUATE1 evaluates $\mathcal{N}$ recursively. At the first step of the recursion, EVALUATE1 finds an SD candidate node d and cuts $\mathcal{N}$ into two portions: the tail $\mathcal{N}_{II}$ and the body $\mathcal{N}_I$. EVALUATE1 computes an optimal decision function of d in $\mathcal{N}_{II}$, and then repeats the process for the body $\mathcal{N}_I$. In this sense, we say that EVALUATE1 works in a divide and conquer fashion.

Separating Bayesian network inference

When evaluating an influence diagram, it is desirable to separate Bayesian network (BN) inference from other computations. There have been intensive research on BN inference, and systems have been built. If an influence diagram evaluation approach can clearly separate BN inference from other computations, then it can be implemented on top of any system for BN inference. This is an application of the principle of separation of concern and the modularity principle.

EVALUATE1 clearly separates BN inference from other computations; all the BN inference tasks — the tasks of computing the marginal potentials $P_0(\pi_d)$, $P_0(\pi_d, d)$, and $P_0(\pi_d, d, \pi_v)$ — are collected in step (a). We have been suggesting to use the clique tree propagation method to compute the marginal potentials. However, other methods can be used as well.

Arc reversals and numerical divisions

Numerical divisions are slower than additions and multiplications; they should be avoided when possible. Arc reversal implies numerical divisions; they should also be avoided if possible.

The procedure EVALUATE1 does not require arc reversals. Furthermore, the only times EVALUATE1 does numerical divisions are when computing the evaluating functional $e(d, \pi_d)$ and the conditional probability $P(r|\pi_r^I)$ of a disturbance recipient r of some decision node.

Multiple value nodes

When the decision maker's utility function can be separated into several components (Tatman and Shachter 1990), it is important to take advantage of the separability by having multiple value nodes. This may imply substantial speed up of computation.

EVALUATE1 is designed for dealing with multiple value nodes.

6.5.2 Other approaches

This subsections examines the approaches by Shachter (1986), Ndilikilikesha (1992), Tatman and Shachter (1990), Shachter (1988), Shenoy (1990), Shachter and Peot (1992), and Shenoy (1992). The approaches by Howard and Matheson (1984), and Cooper (1989) will be discussed in Chapter 9.

Things that can be said for all

Until now, influence diagrams have always been assumed to be no-forgetting; there have been no methods for dealing with influence diagrams that violate the no-forgetting constraint. Even though several authors (Shachter 1988, Tatman and Shachter 1990, and

Shenoy 1992) have noticed and to some extent made use of the fact that some decision nodes may be independent of some of their parents, no one has proposed to prune removable arcs at the preprocessing stage. The reason is that pruning arcs from an influence results leads to the violation of the no-forgetting constraint.

Shenoy (1990) and (1992) proposes a new representation for decision problems, namely valuation-based systems. In this representation, the issue of removable arcs does not occur. We will come back to this point later.

Probably because they do not prune removable arcs by preprocessing, none of the previous approaches work in a divide and conquer fashion. The method by Shenoy (1990, 1992) does not work in a divide and conquer fashion either. The adoption of a divide and conquer strategy is the most important advantage of EVALUATE1 has over the previous approaches.

The rest of this subsection examines the previous approaches with regard to the three remaining properties: separating BN inference, multiple value nodes, and arcs reversals.

Shachter (1986), Ndilikilikesha (1992), and Tatman and Shachter (1990)

Before Shachter (1986), influence diagrams are evaluated in two stages—first transform them into decision trees, and then evaluate the decision trees (Howard and Matheson 1984). Shachter (1986) shows that influence diagrams can be evaluated without the transformation into decision trees, and presents an approach that evaluates an influence diagram by properly applying four operations: barren node removal, arc reversal, random node removal, and decision node removal.

As shown in the third row of Table 6.1, the approach by Shachter (1986) does not separate BN inference, does not deal with multiple value nodes, and requires arcs reversals.

By generalizing influence diagram into potential influence diagrams, the approach by Ndilikilikesha (1992) is able to evaluate an influence diagram by using only three operations: barren node removal, random node removal, and decision node removal. The operation of arc reversal is avoided. However, this approach still does not separate BN inference and does not deal with multiple value nodes (see the fourth row of Table 6.1).

Tatman and Shachter (1990) generalizes influence diagrams in another direction for the sake of dealing with multiple value nodes. The evaluation approach is very much like Shachter (1986), except that it has one more operation, namely the operation of merging value nodes. This approach does not separate BN inference, and it requires arc reversals (see the fifth row of Table 6.1).

Shachter (1988)

Let d be an SD candidate node in an influence diagram $\mathcal{N}$, and let v be the only value node in $\mathcal{N}$. Shachter (1988) and (1990) has noticed that optimal decision functions $\delta^o : \Omega_{\pi_d} \rightarrow \Omega_d$ of d can be obtained through

$$\delta^o_d(\beta) = arg\, max_{\alpha \in \Omega_d} E[v|\pi_d = \beta, d = \alpha], \tag{6.61}$$

for each $\beta \in \Omega_{\pi_d}$.

Further in this direction, Shachter and Peot (1992) (first half) proposes a way to scale the value function μ_v and change v into a observed random node, denoted by u (see also Cooper 1989). Formula (6.61) is transformed into

$$\delta^o_d(\beta) = arg\, max_{\alpha \in \Omega_d} P(\pi_d = \beta, d = \alpha | u = 1). \tag{6.62}$$

Thus, this approach separates BN inference.

Even though Shachter (1990) points out the possibility that the conditional expectation $E[v|\pi_d = \beta, d = \alpha]$ can be computed in one portion of the original network, the algorithm proposed by this paper does not work in a divide and conquer fashion. After the optimal decision function for d is computed, the decision node d is replace by a deterministic random node characterized by the optimal decision function. The resulting influence diagram contains one less decision nodes, but has the same number of nodes as the original network.

Finally, this approach deals with only one single value node. See the sixth row of Table 6.1.

Shenoy (1990), (1992), and Shachter and Peot (1992)

Shenoy (1990), (1992) propose a new representation for Bayesian decision problems, namely valuation-based systems. While a decision network consists of an acyclic directed graph, a set of conditional probabilities, and a set of value functions, a valuation-based system consists of an (undirected) hypergraph graph with a precedence relation, a set of potentials, and a set of valuations. The no-forgetting constraint is enforced by requiring the precedence relation to be transitive.

Influence diagrams can be readily represented as valuation-based systems.

Shenoy (1990) develops an approach for evaluating a valuation-based system by modifying the clique tree propagation algorithm for BN inference. No arc reversals are requires in this approach. The approach was developed for the case of multiple value nodes. However, there is an error. The paper concludes that the combination operation is commutative, but it is not. Consequently, the approach works only for the case of one single value node. Also, the approach does not separate BN inference (see the seventh row of Table 6.1).

Shachter and Peot (1992) (second half) present an algorithm for evaluating influence diagrams that is very similar to Shenoy (1990).

Shenoy (1992) proposes a node removal approach for valuation-based systems. This approach deals with multiple value nodes. It requires no arc reversals. However, numerical divisions become necessary when removing a random node that appears in at least one valuation, but not in all valuations. Thus the approach requires more numerical divisions than EVALUATE1, when there are at least two value nodes. When a random node to be removed appears in all the valuations, the valuations are combined into one single valuation. Thus, the approach makes less use of separability in the utility function than EVALUATE1.

This approach does not separate BN inference (see the eighth row of Table 6.1).

In a decision network, a decision is presumably to be made based on the values of the parents of the decision node. When a decision d is independent of a certain parent d, then the arc $c \rightarrow d$ is removable (Chapter 7). Thus arises the issue of removable arcs. In a valuation-based system, on the other hand, the set of variables that a decision depends upon is not explicitly specified. It is up to the evaluation algorithm to find it out. Thus, there is no issue of removable arcs here. This is why in Table 6.1 we state the issue of removable arcs does not apply to Shenoy (1990) and Shenoy (1992).

An overhead of our approach

Our approach has an overhead. Before doing any numerical computation, we need to identify removable arcs, and figure out the tail and the body. On the other hand, most previous approaches go directly to numerical computations after little graphical preprocessing.

According to the complexity analysis at the end of Section 6.3, the overhead takes $O(k^2(n+e))$ time. Our believe is that in many case, this overhead may help us cut down the time $O(l\lambda^q)$ for numerical computations, which is usually of higher order than $O(k^2(n+e))$.

As a final note, let us point out the previous algorithms for evaluating influence diagrams can possibly be modified to evaluate SDDN's.

Chapter 7

Removable arcs and independence for decision nodes

Given a decision network, there are often nodes and arcs that can be harmlessly removed, in the sense that their removal does not affect the optimal expected value of the network. It is a good idea to prune such nodes and arcs at the preprocessing stage because doing so simplifies the network. It is well known that barren (random and decision) nodes are removable (Shachter 1986). This chapter addresses the issue of removable arcs in the setting of SDDN's.

We begin by establishing the equivalence between removable arcs and independencies for decision nodes (Section 7.1), which is of fundamental importance to this chapter. Section 7.2 introduces lonely arcs — a class of removable arcs that can be graphically identified. In Section 7.3, we show that deleting lonely arcs from an SDDN does not destroy its stepwise-decomposability. Section 7.4 introduces the concepts of potential lonely arcs and of potential barren nodes to deal with the interaction between lonely arcs and barren nodes. Finally, a pruning algorithm is given in Section 7.5. In the next chapter, we shall show that this algorithm prunes all the removable arcs that are graphically identifiable.

Before starting the exposition, let us point out that the issue of removable arcs cannot be addressed in influence diagrams, since deleting arcs from an influence diagram may lead to the violation of the no-forgetting constraint.

7.1 Removable arcs and conditional independencies for decision nodes

In a decision network, an arc into a decision node is *removable* if its removal does not affect the optimal expected value of the network. In a decision network skeleton $\mathcal{K}$, an arc into a decision node is *removable* if it is removable in every decision network over $\mathcal{K}$.

A decision table is a decision function represented in the form of a table. In a decision table of a decision node d, there is one dimension in correspondence to each parent of d. One particular dimension b is *irrelevant* to d if fixing the values of all the other dimensions, no matter which value b takes, the value for d remain the same.

In a decision network, a decision node d is *independent of one particular parent b given all the other parents of d* if there exists an optimal decision table of d in which the b-dimension is irrelevant. When it is the case, we shall write $I_d(d, b|\pi'_d)$, where π'_d stands for $\pi_d - \{b\}$. In a decision network skeleton $\mathcal{K}$, a decision node d is *independent of one particular parent b given all the other parents of d* if it is so in every decision network over $\mathcal{K}$.

The following proposition reveals the relationship between removable arcs and conditional independencies for decision nodes, which is the corner stone of this chapter.

Proposition 7.1 *Let $\mathcal{N}$ be a decision network, d be a decision node and b a parent of d. Then the arc $b \rightarrow d$ is removable if and only if d is independent of b given all the other parents of d.*

Proof: Let d_1, d_2, $\ldots$, d_k be all the decision nodes in $\mathcal{N}$. Suppose d is d_i. We shall write π'_{d_i} for $\pi_{d_i} - \{b\}$.

We first show that if $b \rightarrow d_i$ is removable, then $I_d(d_i, b|\pi'_b)$. Let $\mathcal{N}'$ be the decision network obtained from $\mathcal{N}$ by removing the arc $b \rightarrow d_i$. Since $b \rightarrow d_i$ is removable, $E[\mathcal{N}] = E[\mathcal{N}']$.

Let $\delta'=(\delta'_1, \ldots, \delta'_k)$ be a policy for $\mathcal{N}'$. Let δ'_i be the decision function of d_i in δ'. Then δ'_i is a mapping $\delta'_i : \Omega_{\pi'_{d_i}} \rightarrow \Omega_{d_i}$. Construct a policy δ for $\mathcal{N}$ from δ' by extending δ'_i to be the mapping $\delta_i : \Omega_{\pi_{d_i}} \rightarrow \Omega_{d_i}$ such that

$$\delta_i(\pi_{d_i}) = \delta'_i(\pi'_{d_i}).$$

Then we have

$$E_\delta[\mathcal{N}] = E_{\delta'}[\mathcal{N}'].$$

Now letting δ' be an optimal policy of $\mathcal{N}'$, we get

$$E_\delta[\mathcal{N}] = E_{\delta'}[\mathcal{N}'] = E[\mathcal{N}'] = E[\mathcal{N}].$$

Therefore δ is an optimal policy for $\mathcal{N}$. Noticing that δ_i is independent of b, we get $I_d(d_i, b|\pi'_{d_i})$.

We now show that if $I_d(d_i, b|\pi'_{d_i})$, then $b \rightarrow d_i$ is removable. Since $I_d(d_i, b|\pi'_{d_i})$, there exists an optimal policy δ for $\mathcal{N}$ such that the decision function δ_i of d_i is independent of b. Construct a policy δ' for $\mathcal{N}'$ as follows: let the decision functions of all decision nodes other than d_i be the same as in δ; and let the decision function $\delta'_i : \Omega_{\pi'_{d_i}} \rightarrow \Omega_d$ of d_i be such that

$$\delta'_i(\pi'_{d_i}) = \delta_i(\pi_{d_i}).$$

This definition is valid because $\delta_i(\pi_{d_i})$ is independent of b. It follows from the definition of δ' that $E_{\delta'}[\mathcal{N}'] = E_\delta[\mathcal{N}]$. Consequently,

$$E[\mathcal{N}'] \geq E_{\delta'}[\mathcal{N}'] = E_\delta[\mathcal{N}] = E[\mathcal{N}].$$

On the other hand, we have shown in the first part of the proof that for any policy δ' for $\mathcal{N}'$, there is a policy δ for $\mathcal{N}$ such that $E_\delta[\mathcal{N}] = E_{\delta'}[\mathcal{N}']$. Hence $E[\mathcal{N}'] \leq E[\mathcal{N}]$. Therefore $E[\mathcal{N}'] = E[\mathcal{N}]$. Consequently, the arc $b \rightarrow d_i$ is removable. $\square$

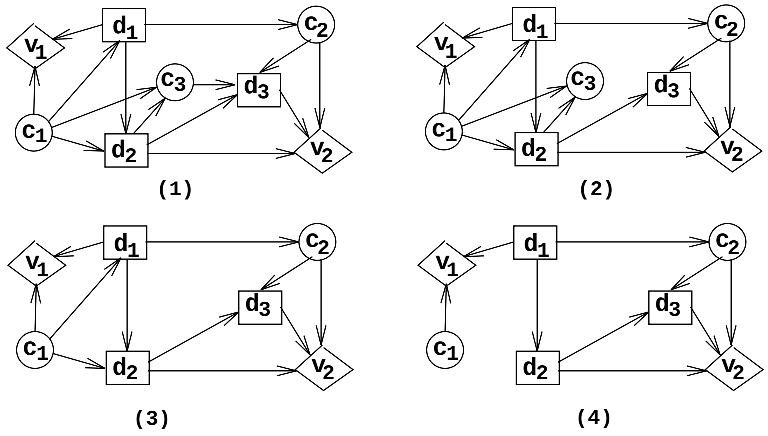

Figure 7.24: Removable arcs and removable nodes.

7.2 Lonely arcs

This section introduces lonely arcs — a class of removable arcs that can be graphically identified.

Suppose $\mathcal{K}$ is a decision network skeleton. Let d be a decision node in $\mathcal{K}$, and let b be a parent of d. The arc $b\rightarrow d$ is said to be *accompanied* if there exist at least one edge in the moral graph $m(\mathcal{K})$ of $\mathcal{K}$ that connects b and some nodes in the downstream set $Y_{II}(d, \mathcal{K})$. When it is the case, we say that such edges accompany the arc $b\rightarrow d$. The arc $b\rightarrow d$ is *lonely* if it is not accompanied. In a decision network $\mathcal{N}$, an arc $b\rightarrow d$ into a decision d is *lonely* if it is lonely in the underlying skeleton.

For example, in the decision network skeleton shown in Figure 7.24 (1), the downstream set $Y_{II}(d_3, \mathcal{K})$ is the singleton $\{v_2\}$. Since the arc $c_2\rightarrow v_2$ is in $\mathcal{K}$, there is the edge (c_2, v_2) in $m(\mathcal{K})$, which accompanies the arc $c_2\rightarrow d_3$. However, the arc $c_3\rightarrow d_3$ is lonely.

The following two lemmas exhibit some basic properties of lonely arcs.

Lemma 7.1 *Suppose $\mathcal{K}$ is a decision network skeleton, and d is an SD candidate node in $\mathcal{K}$. An arc $b\rightarrow d$ is accompanied if and only if b is*

- *a parent to a random node in the downstream set* $Y_{II}(d, \mathcal{K})$*, or*
- *a parent to a value node in the downstream set* $Y_{II}(d, \mathcal{K})$*, or*
- *a disturbance recipient in* π_d*, or*
- *a parent to a disturbance recipient in* π_d*.* □

Lemma 7.2 *Suppose* $\mathcal{K}$ *is a decision network skeleton. Let* d *and* d' *be two different decision nodes in* $\mathcal{K}$*. Suppose* d' *is an SD candidate node. Then an arc* $b \rightarrow d$ *is a lonely arc in* $\mathcal{K}$ *if and only if it is a lonely arc in the body* $\mathcal{K}_I(d', \mathcal{K})$*.* □

Theorem 7.1 *Suppose* $\mathcal{K}$ *is an SDDN skeleton. If an arc* $b \rightarrow d$ *into a decision node* d *is lonely, then* d *is independent of* b*. Consequently, the arc* $b \rightarrow d$ *is removable.*

Proof: Let $\mathcal{N}$ be a decision network over $\mathcal{K}$. We need to show that d is independent of b in $\mathcal{N}$.

By Lemma 7.2, we can assume, without losing generality, that d is an SD candidate node. Let $\mathcal{N}_{II}$ be the tail of $\mathcal{N}$ w.r.t d. Let P_0 denote the joint potential over the random and decision nodes in $\mathcal{N}_{II}$, i.e the product of the conditional probabilities of all the random nodes in the downstream set $Y_{II}(d, \mathcal{N})$ and of the disturbance recipients in π_d.

Since the arc $b \rightarrow d$ is lonely, by Lemma 7.1 b can be neither a disturbance recipient, nor a parent to a disturbance recipient in π_d, nor a parent to random node in $Y_{II}(d, \mathcal{N})$. Thus, P_0 is independent of b.

Again because $b \rightarrow d$ is lonely, by Lemma 7.1 b cannot be a parent to any value nodes in $Y_{II}(d, \mathcal{N})$. Hence, all the value functions in $\mathcal{N}_{II}$ are independent of b.

Putting those two points together, we get that the evaluation functional $e(d, \pi_d)$ (see equation (6.47)) is independent of b. According to equation (6.55), the optimal decision function of d is independent of b. The theorem is proved. □

In Figure 7.24 (1), the arc $c_3 \rightarrow d_3$ is lonely, hence removable. The removal of $c_3 \rightarrow d_3$ gives us the decision network skeleton in Figure 7.24 (2).

The following corollary is obtained from the proof of Theorem 7.1.

Corollary 7.1 *Suppose $b \rightarrow d$ is a lonely arc in an SDDN $\mathcal{N}$. Let $\mathcal{N}'$ be the decision network obtained from $\mathcal{N}$ by removing the arc $b \rightarrow d$. Then, $\mathcal{N}'$ and $\mathcal{N}$ have the same optimal decision tables for all the decision nodes other than d. Furthermore, the optimal decision tables for d in $\mathcal{N}'$ can be obtained from the optimal decision tables for d in $\mathcal{N}$ by shrinking the irrelevant b-dimension.* $\Box$

7.3 Pruning lonely arcs and stepwise-solvability

In order to repeatedly apply Theorem 7.1, we need the following theorem.

Theorem 7.2 *The decision network skeleton resulted from pruning a lonely arc from an SDDN skeleton is again stepwise-decomposable.*

Proof: Let $\mathcal{K}$ be an SDDN skeleton and $b \rightarrow d$ be a lonely arc. Let $\mathcal{K}'$ be the resulting skeleton after removing $b \rightarrow d$ from $\mathcal{K}$. We prove that $\mathcal{K}'$ is stepwise-decomposable by induction on the number k of decision nodes in $\mathcal{K}$. When k=1, $\mathcal{K}'$ also contains only one decision node; hence is stepwise-decomposable.

Suppose $\mathcal{K}'$ is stepwise-decomposable if $k=m-1$. Now consider the case of $k=m$. Let d' be a candidate node of $\mathcal{K}$. There are two cases depending on whether $d'=d$. Let us first consider the case when $d' \neq d$. According to Lemma 7.2, $b \rightarrow d$ is also a lonely arc in the body $\mathcal{K}_I(d', \mathcal{K})$. Let $\mathcal{K}'_I$ be the resulting decision network skeleton after removing $b \rightarrow d$ from $\mathcal{K}_I$. By the induction hypothesis, $\mathcal{K}'_I$ is stepwise-decomposable. It is easy to see that $\mathcal{K}'_I$ is the body of $\mathcal{K}'_I(d', \mathcal{K}')$. By Lemma 5.2, $\mathcal{K}'$ is also stepwise-decomposable.

Now consider the case when $d'=d$. Since there are no edges in $m(\mathcal{K})$ that connect b and nodes in the downstream set $Y_{II}(d,\mathcal{K})$, the set $Y_{II}(d,\mathcal{K}')$ is the same as $Y_{II}(d,\mathcal{K})$. So, d is also a candidate decision node of $\mathcal{K}'$.

The body $\mathcal{K}'_I(d,\mathcal{K}')$ is different from the body $\mathcal{K}_I(d,\mathcal{K})$ only in that in $\mathcal{K}'_I$ there is no arc from b to the tail-value node u, while there is in $\mathcal{K}_I$. Since $\mathcal{K}_I$ is stepwise-decomposable, so must be $\mathcal{K}'_I$. By Lemma 5.2, $\mathcal{K}'$ is also stepwise-decomposable. □

7.4 Potential lonely arcs and barren nodes

In a decision network skeleton, a *barren node* is a random or decision node that does not have any children. In the following, we shall distinguish decision barren nodes and random barren nodes. The node c_3 in Figure 7.24 (2) is a random barren node.

Proposition 7.2 *(Shachter 1986) A barren node may be simply removed from a decision network. If it is a decision node, then any decision function is optimal.*

Now we recursively define potential lonely arcs and potential barren nodes. A *potential lonely arc* is a lonely arc or an arc that becomes lonely after the removal of some barren and potential barren nodes, and the removal of some lonely and potential lonely arcs. A *potential barren node* is a barren node or a node that becomes barren after the removal of some lonely and potential lonely arcs, and the removal of some barren and potential barren nodes.

Going back to our example, after the removal of $c_3 \rightarrow d_3$, the the node c_3 becomes barren, and hence can be removed. After the removal of c_3, $c_1 \rightarrow d_2$ becomes lonely. After the removal of $c_1 \rightarrow d_2$, $c_1 \rightarrow d_1$ becomes lonely.

Here is a corollary of Theorem 7.1.

Corollary 7.2 *Suppose $\mathcal{K}$ is an SDDN skeleton. If an arc $b \rightarrow d$ into a decision node d is a potential lonely arc, then d is independent of b. Consequently, the arc $b \rightarrow d$ is removable.*

7.5 An algorithm

This section presents the algorithm PRUNE-REMOVABLES, which prunes all the potential lonely arcs and potential barren nodes in an SDDN skeleton.

Procedure PRUNE-REMOVABLES($\mathcal{K}$)

- Input: $\mathcal{K}$ — an SDDN skeleton.
- Outputs: An SDDN skeleton that does not contain potential arcs and potential barren nodes.

1. **If** there is no decision node in $\mathcal{K}$, output $\mathcal{K}$ and stop.
2. **Else**
   - Find and remove all the barren nodes;
   - Find a candidate node d of $\mathcal{K}$, and compute the downstream set $Y_{II}(d, \mathcal{K})$ of π_d.
   - Find and remove all the lonely arcs among the arcs from π_d to d. Let $\mathcal{K}'$ be the resulting skeleton.
   - In $\mathcal{K}'$, treat d as a random node[1] (thus $\mathcal{K}'$ contains one less decision node than $\mathcal{K}$) and recursively call PRUNE-REMOVABLES($\mathcal{K}'$).

As an example, consider the SDDN skeleton in Figure 7.24 (1). There are no barren nodes at the beginning; and d_3 is the only candidate node. The downstream set $\mathcal{K}_{II}(d_3, \mathcal{K})$

[1]The node d is treated as a random node only within the scope of PRUNE-REMOVABLES.

is the singleton $\{v_2\}$. One can see that $c_3{\rightarrow}d_3$ is the only lonely arc. After the removal of $c_3{\rightarrow}d_3$, the skeleton becomes as shown in Figure 7.24 (2), where c_3 is a barren node. After the removal of c_3, we get the skeleton in Figure 7.24 (3). Since d_3 is now treated as a random node, d_2 becomes a candidate node. The arc $c_1{\rightarrow}d_2$ is lonely, and hence is removed. Thereafter, d_2 is also treated as a random node, rendering d_1 a candidate node. The arc $c_1{\rightarrow}d_1$ is lonely and hence removed. The final skeleton is shown in Figure 7.24 (4).

Let $\mathcal{K}'$ be the output decision network skeleton of of PRUNE-REMOVABLES($\mathcal{K}$). How is $\mathcal{K}'$ related to $\mathcal{K}$ in terms of decision tables? Let $\mathcal{N}$ and $\mathcal{N}'$ be decision networks over $\mathcal{K}$ and $\mathcal{K}'$ respectively such that in both $\mathcal{N}$ and $\mathcal{N}'$ each variable has the same frame, each random variable has the same conditional probability, and each value node has the same value functions. By repeatedly applying Corollary 7.1, we can conclude that the optimal decision tables for $\mathcal{N}'$ can be obtained from those for $\mathcal{N}$ by deleting irrelevant dimensions.

Finally, let us consider the complexity of PRUNE-REMOVABLES. Let n be the total number of nodes in $\mathcal{K}$, k be the number of decision nodes, a be the number of arcs, e be the number of edges in the moral graph of $\mathcal{K}$, and p be the maximum number of parents of a decision node. Finding all the barren nodes takes $O(a)$ time. According to the complexity analysis of TEST-STEPWISE-DECOMPOSABILITY, finding an SD candidate node and computing its downstream set takes $O(k(n+e))$ time. To find lonely arcs, we need to check, for each node x in π_d, if x is connected to at least one node in the downstream set $Y_{II}(d, \mathcal{K})$, which can be done in $O(pn)$ time. So, the total complexity of PRUNE-REMOVABLE is $O(k(k(e+n)+pn)) = O(k^2e + k^2n + kpn)$.

Chapter 8

Stepwise-solvability and stepwise-decomposability

We have shown in Section 5.2 that if a smooth decision network is stepwise-decomposable, then it stepwise-solvable. In this chapter, we go further to prove that if a decision network skeleton, smooth or non-smooth, is stepwise-decomposable, then if it is stepwise-solvable. More importantly, we show that under "normal" circumstances if a decision network skeleton is stepwise-solvable, then it is stepwise-decomposable (Section 8.8). Thus, stepwise-decomposability is the weakest graphical criterion that guarantees stepwise-solvability.

According to Corollary 7.2, potential lonely arcs are removable. In this chapter, we also show that potential lonely arcs are all the removable arcs that can be graphically identified (Section 8.7).

The proof technique is induction on the number of random nodes and on the number of decision nodes. In order to do induction on the number of random nodes, we need three operations on decision network skeletons, namely short-cutting (Section 8.2), root random node removal (Section 8.3), and arc reversal (Section 8.4). Section 8.5 shows how those three operations fit together. An induction apparatus on the number of decision node is given in Section 8.6. Let us begin with the concept of normal decision network skeletons.

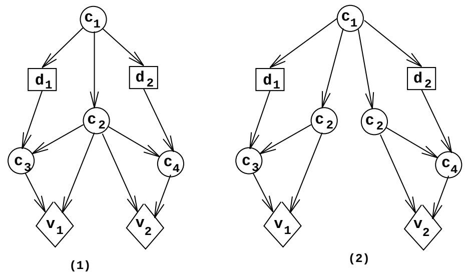

Figure 8.25: An abnormal decision network skeleton (1), and an normal equivalent skeleton (2).

8.1 Normal decision network skeletons

A decision network skeleton $\mathcal{K}$ is *normal* if for any decision node d, there is a directed path from d to each value node in the downstream set $Y_{II}(d, \mathcal{K})$ of π_d. A decision network is *normal* if its underlying skeleton is.

As an example, let $\mathcal{K}$ be the decision network skeleton in Figure 8.25 (1). $Y_{II}(d_1, \mathcal{K})$ contains all the nodes except c_1. In particular, $v_2 \in Y_{II}$. But there is no directed path from d_1 to v_2. So $\mathcal{K}$ is abnormal. On the other hand, the decision network skeleton in Figure 8.25 (2) is normal.

What is the intuition behind this concept of normality? Consider a decision node d and a value node v in a decision network $\mathcal{N}$. Given any policy δ of $\mathcal{N}$, let P_δ be the joint probability δ induces over all the random and decision nodes of $\mathcal{N}$. The expected value $E_\delta[v]$ of v is given by

$$E_\delta[v] = \sum_{\pi_v} P_\delta(\pi_v)\mu_v(\pi_v),$$

where μ_v stands for the value function of v. According Proposition 3.1, if there is no directed path from d to v, then d is irrelevant to $P_\delta(\pi_v)$ and hence to $E_\delta[v]$. In other words, d can influence v only when there exists a directed path from d to v.

Intuitively a normal decision network is one where each decision node d can influence all the value nodes that are not m-separated from d by the parents of d. In other words, all those value nodes that d can not influence are m-separated from d by π_d.

An abnormal decision network skeleton can be stepwise-solvable even when it is not stepwise-decomposable. For example, the decision network skeleton in Figure 8.25 (1) is not stepwise-decomposable. However it is stepwise-solvable. To see this, let $\mathcal{N}$ be an arbitrary decision network over the skeleton. Construct a decision network $\mathcal{N}'$ over the skeleton in Figure 8.25 (2) such that c_2' has the same frame and conditional probability as c_2. For any policy δ, let P_δ be the joint probability δ induces over all the random and decision nodes in $\mathcal{N}$, and P_δ' be the joint probability δ induces over all the random and decision nodes in $\mathcal{N}'$. By Proposition 3.1, we have that

$$P_\delta(\pi_{v_1}) = P_\delta(c_2, c_3) = P_\delta'(c_2, c_3) = P_\delta'(\pi_{v_1}').$$

Thus the expected value of v_1 under δ in $\mathcal{N}$ is the same as that in $\mathcal{N}'$. By the same line of reasoning, we can show that the expected value of v_2 under δ in $\mathcal{N}$ is the same as that in $\mathcal{N}'$. Therefore $\mathcal{N}$ and $\mathcal{N}'$ are equivalent. Since $\mathcal{N}'$ is stepwise-decomposable, it is stepwise-solvable. Therefore $\mathcal{N}$ is also stepwise-solvable.

The main goal of this chapter is to show that a normal decision skeleton with no barren nodes and no lonely arcs is stepwise-solvable only if it is stepwise-decomposable. We also show that a normal SDDN skeleton with no barren nodes contains removable arcs only if it contains potential lonely arcs.

The reader may ask: what about abnormal decision network skeletons? We conjecture that abnormal decision network skeletons can always to transformed into "equivalent" normal skeletons. For example, the decision network skeleton in Figure 8.25 (1) can be transformed into the one in Figure 8.25 (2). However, we have not been able to precisely formulate and prove the conjecture.

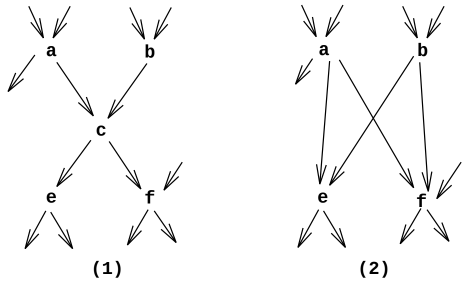

Figure 8.26: Short-cutting. The random node c in (1) is short-cut, resulting in the decision network skeleton in (2).

8.2 Short-cutting

This sections introduces the operation of shorting cutting random nodes from a decision network skeleton. The properties of the operation with regard to induction are explored. Short-cutting is the first of the three operations that are needed to facilitate induction on the number of random nodes.

Before getting started, however, let us make a note about notation usage in this chapter. Applying the operation of short-cutting, or any other operation, on a decision network skeleton $\mathcal{K}$ results in another decision network skeleton $\mathcal{K}'$. We shall let π_x and π'_x to denote the set of parents of x in $\mathcal{K}$ and in $\mathcal{K}'$ respectively. Let $\mathcal{N}$ and $\mathcal{N}'$ be decision networks over $\mathcal{K}$ and $\mathcal{K}'$. We shall denote the conditional probability in $\mathcal{N}$ of a random node c by $P(c|\pi_c)$ and the value function in $\mathcal{N}$ of a value node v by $\mu_v(\pi_v)$. Similarly, we use $P'(c|\pi'_c)$ and $\mu'_v(\pi'_v)$ to denote the conditional probability of c and the value function of v in $\mathcal{N}'$ respectively.

Let $\mathcal{K}$ be a decision network skeleton. Let c be random node in $\mathcal{K}$ such that c has at least one parent. To *short-cut* c is to delete c from $\mathcal{K}$, and draw an arc from every parent of c to each child of c. Figure 8.26 illustrates this concept. We see that after the short-cuting, every child of c inherit all the parents of c.

The main task of this section is to prove Propositions 8.1 and 8.2, which are constructing blocks of our induction mechanism. We first present two lemmas.

Lemma 8.1 *Let $\mathcal{K}$ be a decision network skeleton and c be a random node in $\mathcal{K}$ that has at least one parent. Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by short-cutting c. If c is not a barren node, then for any two nodes x and y in $\mathcal{K}'$,*

1. *There is a directed path PATH1 from x to y in $\mathcal{K}'$ if and only if there is a directed path from x to y in $\mathcal{K}$ that consists of the same nodes as PATH1 with the possible addition of the node c.*
2. *There is a path PATH2 between x and y in the moral graph $m(\mathcal{K}')$ if and only if there is a path between x and y in the moral graph $m(\mathcal{K})$ that consists of the same nodes as PATH2 with the possible addition of the node c.*

Proof: The lemma follows directly from the definition of short-cutting. □

Lemma 8.2 *Let $\mathcal{K}$ be a decision network skeleton and c be a random node in $\mathcal{K}$ that has at least one parent. Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by short-cutting c. Let d be a decision node in $\mathcal{K}$ (or equivalently in $\mathcal{K}'$).*

1. *If c is in the upstream set $Y_I(d,\mathcal{K})$, then $\pi'_d=\pi_d$, and $Y_{II}(d,\mathcal{K}')=Y_{II}(d,\mathcal{K})$.*
2. *If c is in the downstream set $Y_{II}(d,\mathcal{K})$, then $\pi'_d=\pi_d$, and $Y_{II}(d,\mathcal{K}')=Y_{II}(d,\mathcal{K})-\{c\}$.*
3. *If $c\in\pi_d$, then $\pi'_d=(\pi_d-\{c\})\cup\pi_c$. Furthermore if none of the parents of c are $Y_{II}(d,\mathcal{K})$, then $Y_{II}(d,\mathcal{K}')=Y_{II}(d,\mathcal{K})$.*

Proof: The lemma follows directly from Lemma 8.1 and the fact the the downstream set $Y_{II}(d,\mathcal{K})$ consists of all the nodes in $\mathcal{K}$ that are not m-separated from d by π_d. □

Proposition 8.1 *Let $\mathcal{K}$ be a decision network skeleton, and let c be a random node which has at least one parent. Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by short-cutting c.*

1. *If $\mathcal{K}$ does not contain any barren nodes, neither does $\mathcal{K}'$.*
2. *If $\mathcal{K}$ normal, so is $\mathcal{K}'$.*
3. *If $\mathcal{K}$ stepwise-decomposable, so is $\mathcal{K}'$.*
4. *Suppose $\mathcal{K}$ is stepwise-decomposable and contains no barren nodes, and suppose that if $c \in \pi_d$ for some decision node d, then none of the parents of c are in the downstream set $Y_{II}(d, \mathcal{K})$. Then when $\mathcal{K}$ does not contain any lonely arcs, neither does $\mathcal{K}'$.*

Proof: To show item 1, suppose a decision or random node x is not barren in $\mathcal{K}$. Then it has at least one child y. If $y = c$, then the children of c in $\mathcal{K}$ become the parents of x. Otherwise, y remains a child of x in $\mathcal{K}'$. In either case, x has at least one child; hence $\mathcal{K}'$ contains no barren nodes.

By Lemma 8.2, we have that for any decision node d,

$$Y_{II}(d, \mathcal{K}') \subseteq Y_{II}(d, \mathcal{K}). \tag{8.63}$$

Together with Lemma 8.1, this proves item 2.

To show item 3, we notice that because of equation (8.63), if a decision node d is an SD candidate decision node of $\mathcal{K}$, then it is also an SD candidate decision node of $\mathcal{K}'$.

First consider the case when $c \in Y_{II}(d, \mathcal{K})$. In this case the body $\mathcal{K}'_I(d, \mathcal{K}')$ of $\mathcal{K}'$ w.r.t d is the same as the body $\mathcal{K}_I(d, \mathcal{K})$ of $\mathcal{K}$ w.r.t d. If $\mathcal{K}$ is stepwise-decomposable, then so is $\mathcal{K}_I$, and hence $\mathcal{K}'_I$. By Lemma 5.2, $\mathcal{K}'$ is also stepwise-decomposable.

On the other hand, when $c \notin Y_{II}(d, \mathcal{K})$, then $\mathcal{K}'_I$ is the same as the resulting decision network skeleton after short-cutting c from $\mathcal{K}_I$. If $\mathcal{K}$ is stepwise-decomposable, so is

$\mathcal{K}_I$. Consequently we can assume, as an induction hypothesis, that $\mathcal{K}'_I$ is stepwise-decomposable. By Lemma 5.2, $\mathcal{K}'$ is also stepwise-decomposable. Item 3 is therefore proved.

To show item 4, let d be an arbitrary decision node. We need to show that the arcs from nodes in π'_d to d are accompanied in $\mathcal{K}'$. There are three cases:

Case 1). If $c \in Y_I(d, \mathcal{K})$, by Lemma 8.2 we have $\pi'_d = \pi_d$ and $Y_{II}(d, \mathcal{K}') = Y_{II}(d, \mathcal{K})$. Thus, the arcs from nodes in π'_d to d are accompanied in $\mathcal{K}'$ by the same edges as in $\mathcal{K}$.

Case 2). If $c \in Y_{II}(d, \mathcal{K})$, by Lemma 8.2 we have $\pi'_d = \pi_d$ and $Y_{II}(d, \mathcal{K}') = Y_{II}(d, \mathcal{K}) - \{c\}$. Since the arcs from nodes in π'_d to d are accompanied in $\mathcal{K}$, and since $\mathcal{K}$ contains no barren nodes, by Lemma 8.1 the arcs from π'_d to d remain accompanied in $\mathcal{K}'$.

Case 3). If $c \in \pi_d$, $\pi'_d = (\pi_d - \{c\}) \cup \pi_c$. The arcs from node in $\pi_d - \{c\}$ to d are accompanied in $\mathcal{K}$ and remain accompanied in $\mathcal{K}'$. We need only show that an arc from a node $y \in \pi_c$ to d are not lonely in $\mathcal{K}'$.

Since $\mathcal{K}$ contains no lonely arcs and none of the parents of c are in the downstream set $Y_{II}(d, \mathcal{K})$, either there is an arc $c \rightarrow x$ to a node x in $Y_{II}(d, \mathcal{K})$, or there exists a random node $b \in \pi_d$ and a node $x \in Y_{II}(d, \mathcal{K})$ such that the arcs $c \rightarrow b$ and $x \rightarrow b$ appear in $\mathcal{K}$.

In the first case, the arc $y \rightarrow x$ appears in $\mathcal{K}'$. Hence the edge (y, x) appears in $m(\mathcal{K}')$ which accompanies the arc $y \rightarrow d$. In the second case, $y \rightarrow b$ appears in $\mathcal{K}'$, and hence the edge (y, x) appears in $m(\mathcal{K}')$, which accompanies the arc $y \rightarrow d$. This proves that $\mathcal{K}'$ contains no lonely arcs. The proof is complete. □

Proposition 8.2 *Let $\mathcal{K}$ be a decision network skeleton, let c be a random node which has at least one parent. Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by short-cutting c. Then for any decision network $\mathcal{N}'$ over $\mathcal{K}$, there is a decision network $\mathcal{N}$ over $\mathcal{K}$ that is equivalent to $\mathcal{N}'$.*

Proof: Given $\mathcal{N}'$, construct $\mathcal{N}$ as follows. Let all the nodes in $\mathcal{N}$, excluding c, have the same frame as the corresponding nodes in $\mathcal{N}'$. Let c be a compound variable consisting of a copy of each node in π_c. We set the conditional probability $P(c|\pi_c)$ of c to be

$$P(c|\pi_c) = \begin{cases} 1 & \text{if } c = \pi_c \\ 0 & \text{otherwise} \end{cases} \tag{8.64}$$

For any child of y of c, $\pi'_y = (\pi_y - \{c\}) \cup \pi_c$. Noticing $\Omega_{\pi_y} = \Omega_{\pi'_y}$, we set

$$P(y|\pi_y) = P'(y|\pi_y - \{c\}, \pi_c) = P'(y|\pi'_y).$$

The conditional probabilities of all other random nodes in $\mathcal{N}$ are the same as in $\mathcal{N}'$.

For any value node v, there are two cases depending on whether $c \in \pi_v$. When $c \notin \pi_v$, we have $\pi_v = \pi'_v$. In this case, we set

$$\mu_v(\pi_v) = \mu'_v(\pi'_v).$$

On the other hand, when $c \in \pi_v$, we can assume that $\pi_v \cap \pi_c = \emptyset$, i.e v has no parents in π_c. Because if v has parents in π_c, we can always set the value function of v to be independent of those nodes. Consequently we have $\pi'_v = (\pi_v - \{c\}) \cup \pi_c$. Noticing $\Omega_{\pi_v} = \Omega_{\pi'_v}$, we set

$$\mu_v(\pi_v) = \mu'_v(\pi_v - \{c\}, \pi_c) = \mu'_v(\pi'_v).$$

To show that $\mathcal{N}$ and $\mathcal{N}'$ are equivalent, we first notice that they do have the same policy space. Let δ be a policy, and let P_δ be the joint probability δ induces over all the random and decision nodes of $\mathcal{N}$, and let P'_δ be the joint probability δ induces over all the random and decision nodes of $\mathcal{N}'$.

Let B be a set of random and decision nodes of $\mathcal{N}$. It follows from the definition of the conditional probabilities of $\mathcal{N}$ that

$$P_\delta(B) = \begin{cases} P'_\delta(B) & \text{if } c \notin B \\ P'_\delta(B - \{c\}, \pi_c) & \text{if } c \in B \text{ and } B \cap \pi_c = \emptyset \end{cases} \tag{8.65}$$

For any value node v such that $c \notin \pi_v$, we have

$$\sum_{\pi_v} P_\delta(\pi_v)\mu_v(\pi_v) = \sum_{\pi'_v} P'_\delta(\pi'_v)\mu'_v(\pi'_v).$$

On the other hand, for any value node v such that $c \in \pi_v$ we have

$$\begin{aligned}\sum_{\pi_v} P_\delta(\pi_v)\mu_v(\pi_v) &= \sum_{\pi_v - \{c\}, \pi_c} P'_\delta(\pi_v - \{c\}, \pi_c)\mu'_v(\pi_v - \{c\}, \pi_c) \\ &= \sum_{\pi'_v} P'_\delta(\pi'_v)\mu_v(\pi'_v).\end{aligned}$$

Therefore

$$E_\delta[\mathcal{N}] = E_\delta[\mathcal{N}'].$$

That is $\mathcal{N}$ and $\mathcal{N}'$ are equivalent. □

8.3 Root random node removal

This section investigates the operation of removing root random nodes from decision network skeletons. The properties of the operation with regard to induction are of particular interest. Root random node removal is the second of the three operations that are needed to facilitate induction on the number of random nodes.

Proposition 8.3 *Let $\mathcal{K}$ be decision network skeleton, and let c be a root random node, i.e a random node without parents. Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by removing c and the arcs originating from c.*

1. *If $\mathcal{K}$ contains no barren nodes, neither does $\mathcal{K}'$.*
2. *If $\mathcal{K}$ is normal, then so is $\mathcal{K}'$.*
3. *If $\mathcal{K}$ is stepwise-decomposable, then so is $\mathcal{K}'$.*

4. *Suppose $\mathcal{K}$ is normal and stepwise-decomposable, and contains no barren nodes. Suppose that for any decision node d such that $c \in Y_{II}(d, \mathcal{K})$, none of the children of c are in π_d. Then when $\mathcal{K}$ does not contain any lonely arcs, neither does $\mathcal{K}'$.*

Proof: The first item is straightforward.

To prove second item, we notice that for any decision node d,

$$Y_{II}(d, \mathcal{K}') \subseteq Y_{II}(d, \mathcal{K}). \tag{8.66}$$

Together with the fact that c is a root random node, this equation entails item 2.

Because of equation (8.66), an SD candidate decision node d of $\mathcal{K}$ is also an SD candidate decision node for $\mathcal{K}'$. Moreover, when c is not in the downstream set $Y_{II}(d, \mathcal{K})$, then the body $\mathcal{K}'_I(d, \mathcal{K}')$ of $\mathcal{K}'$ w.r.t d can be obtained from the body $\mathcal{K}_I(d, \mathcal{K})$ of $\mathcal{K}$ w.r.t d by removing c. Hence item 3 can be proved in the same way as the third item of Proposition 8.1.

To show item 4, let d be an arbitrary decision node. We need to show that the arcs from nodes in π'_d to d are accompanied in $\mathcal{K}'$. There are three cases:

Case 1). If $c \in Y_I(d, \mathcal{K})$, then $\pi'_d = \pi_d$ and $Y_{II}(d, \mathcal{K}') = Y_{II}(d, \mathcal{K})$. In this case, the arcs from nodes in π'_d to d are accompanied in $\mathcal{K}'$ by the same edges as in $\mathcal{K}$.

Case 2). If $c \in Y_{II}(d, \mathcal{K})$, then $\pi'_d = \pi_d$ and $Y_{II}(d, \mathcal{K}') = Y_{II}(d, \mathcal{K}) - \{c\}$. For any $x \in \pi'_d$, the arc $x \rightarrow d$ is accompanied in $\mathcal{K}$ by, say, the edge (x, y) in $m(\mathcal{K})$. There are two subcases depending on whether or not $y=c$.

Case 2.1) $y=c$. Since c is a root, there must exist another node z such that the arcs $x \rightarrow z$ and $c \rightarrow z$ appear in $\mathcal{K}$. Since none of the children of c are in π_d, $z \in Y_{II}(d, \mathcal{K})$. There are three subsubcases depending on whether z is a random node, a decision node, or a value node.

Case 2.1.1). z is a random node. Since $\mathcal{K}$ contain no barren nodes, there exists a value node v such that there is directed path from z to v. Since $\mathcal{K}$ is normal, there must be a directed path from d to v. Hence $z \in Y_{II}(d, \mathcal{K}')$. Therefore in $\mathcal{K}'$ the arc $x \rightarrow d$ is accompanied by the edge (x, z) of $m(\mathcal{K}')$.

Case 2.1.2). z is a a value node. Since $\mathcal{K}$ is normal, there exists a directed path from d to v. Hence, z must be in the downstream set $Y_{II}(d, \mathcal{K}')$. Therefore in $\mathcal{K}'$ the arc $x \rightarrow d$ is accompanied by the edge (x, z).

Case 2.1.3). z is a decision node. In this case, there must be at least one value node in the downstream set $Y_{II}(z, \mathcal{K})$, because $\mathcal{K}$ contains no barren nodes. Since $\mathcal{K}$ is normal, there exists a direct path, say PATH, from d to v. Since $\mathcal{K}$ is stepwise-decomposable, π_z m-separated v from d. Therefore z must be in PATH. Consequently $z \in Y_{II}(d, \mathcal{K}')$. Therefore in $\mathcal{K}'$ the arc $x \rightarrow d$ is accompanied by the edge (x, z).

Case 2.2) $y \neq c$. There are again three subsubcases depending on whether y is a random node, a decision node, or a value node. The proof for this subcase can be carried out in the same way as in case 2.1), except with z replaced by y.

Case 3). If $c \in \pi_d$, then $\pi'_d = \pi_d - \{c\}$. For any node $x \in \pi'_d$, the arc $x \rightarrow d$ is accompanied in $\mathcal{K}$. Hence either there is an arc connecting x and a node z in the downstream set $Y_{II}(d, \mathcal{K})$, or there exists another node $y \in \pi_d$ and a node $z \in Y_{II}(d, \mathcal{K})$ such that the arcs $x \rightarrow y$ and $z \rightarrow y$ appear in $\mathcal{K}$.

In the first case, z cannot be c. Hence the arc that connects x and z in $\mathcal{K}$ is also in $\mathcal{K}'$, hence $x \rightarrow d$ is accompanied in $\mathcal{K}'$. In the second case, y cannot be c because c is a root random node; and z cannot be c either because $c \in \pi_d$. Hence the arcs $x \rightarrow y$ and $z \rightarrow y$ also appear in $\mathcal{K}'$. Consequently, the arc $x \rightarrow d$ are also accompanied in $\mathcal{K}'$. The proof is complete. □

Proposition 8.4 *Let $\mathcal{K}$ be decision network skeleton, and let c be a root random node. Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by removing c and the arcs originating from c. For any decision network $\mathcal{N}'$ over $\mathcal{K}'$, there is a decision network $\mathcal{N}$*

over $\mathcal{K}$ that is equivalent to $\mathcal{N}'$.

Proof: Given $\mathcal{N}'$, construct $\mathcal{N}$ as follows. Let all the nodes in $\mathcal{N}$, excluding c, have the same frames as in $\mathcal{N}'$. Let c take only one value, say 1. For a random node r such that $c \notin \pi_r$, set the conditional probability of r the same as in $\mathcal{N}'$. If a random node r is such that $c \in \pi_r$, then $\pi_r = \pi'_r \cup \{c\}$. In this case, set its conditional probability $P(r|\pi_d)$ as follows:

$$P(r|\pi_r) = P'(r|\pi'_r),$$

where $P'(r|\pi'_r)$ stands for the conditional probability of r in $\mathcal{N}'$. This definition is valid because c takes only one value.

For any value node v such that $c \notin \pi_v$, set the value function of v to be the same as in $\mathcal{N}'$. If a value node v is such that $c \in \pi_v$, then $\pi_v = \pi'_v \cup \{c\}$. In this case, set

$$\mu_v(\pi_v) = \mu'_v(\pi'_v),$$

where $\mu'(v|\pi'_v)$ stands for the value function of v in $\mathcal{N}'$. This definition is valid because c takes only one value.

We now show that $\mathcal{N}$ is equivalent to $\mathcal{N}'$. For any decision node d such that $c \notin \pi_d$, then $\pi'_d = \pi_d$; d has the same decision function space in both $\mathcal{N}$ and $\mathcal{N}'$. If a decision node is such that $c \in \pi_d$, then $\pi'_d = \pi_d - \{c\}$. Since c only take one value, d still has the same decision function space in both $\mathcal{N}$ and $\mathcal{N}'$. So, $\mathcal{N}$ has the same policy space as $\mathcal{N}'$.

It is evident that given a policy δ, $E_\delta[\mathcal{N}] = E_\delta[\mathcal{N}']$. Therefore, $\mathcal{N}$ and $\mathcal{N}'$ are equivalent. The proposition is proved. $\Box$

8.4 Arc reversal

This section revisits the operation of reversing arcs in decision network skeleton, with an eye on its induction properties. Arc reversal is the third of the three operations that are needed to facilitate induction on the number of random nodes.

Proposition 8.5 *Suppose $\mathcal{K}$ is a decision network skeleton. Let b and c be two random nodes such that the arc $c \rightarrow b$ appears in $\mathcal{K}$ and is reversible. Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by reversing the arc $c \rightarrow b$.*

1. *Suppose c has at least two children. Then if $\mathcal{K}$ does not contains any barren nodes, neither does $\mathcal{K}'$.*
2. *Suppose c is a root. Then if $\mathcal{K}$ is normal, so is $\mathcal{K}'$.*
3. *Suppose c is a root. Then if $\mathcal{K}$ is stepwise-decomposable, then so is $\mathcal{K}'$.*
4. *Suppose c is a root. Then if $\mathcal{K}$ does not contains any lonely arcs, neither does $\mathcal{K}'$.*

Proof: Item 1 is straightforward.

When c is a root, the moral graph $m(\mathcal{K}')$ of $\mathcal{K}'$ is the same as the moral graph $m(\mathcal{K})$ of $\mathcal{K}$. Hence, items 3 and 4 follow.

To show item 2, let d be an arbitrary decision node. It follows from $m(\mathcal{K}') = m(\mathcal{K})$ that $Y_{II}(d, \mathcal{K}') = Y_{II}(d, \mathcal{K})$. Let v be a value node in the downstream set $Y_{II}(d, \mathcal{K}') = Y_{II}(d, \mathcal{K})$. Since $\mathcal{K}$ is normal, there must be a directed path, say PATH1, in $\mathcal{K}$ from d to v. Since c is a root, the arc $c \rightarrow b$ cannot be in PATH1. Thus PATH1 is also a path in $\mathcal{K}'$. Therefore $\mathcal{K}'$ is also normal. The proposition is proved. □

Proposition 8.6 *Let $\mathcal{K}$ be a decision network skeleton, let b and c be two random nodes such that the arc $c \rightarrow b$ appears in $\mathcal{K}$ and is reversible. Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by reversing the arc $c \rightarrow b$. If c is a root, then*

$$\pi'_b = \pi_b - \{c\} \text{ and } \pi'_c = \{b\} \cup \pi'_b.$$

Furthermore for any decision network $\mathcal{N}'$ over $\mathcal{K}'$ such that

1. *c is a compound variable consisting a copy of each node in π'_c,*

2. *the conditional probability $P'(c|\pi'_c)$ is given by*

$$P'(c|\pi'_c) = \begin{cases} 1 & \text{if } c = \pi'_c \\ 0 & \text{otherwise} \end{cases}, \tag{8.67}$$

3. *and if a value node is a descendent of c, then it is also a descendent of b,*

there exists an decision network $\mathcal{N}$ over $\mathcal{K}$ that is equivalent to $\mathcal{N}'$.

Proof: Given $\mathcal{N}'$, construct $\mathcal{N}$ as follows. Let all the variables have the same frames as in $\mathcal{N}'$. Noticing that c is the compound variable consisting a copy of each node in π'_c, we set

- the conditional probability $P(b|\pi_b) = P(b|c, \pi'_b)$ of b to be

$$P(b|c, \pi'_b) = \begin{cases} P'(b|\pi'_b) & \text{if } c{=}\pi'_c \\ 0 & \text{otherwise} \end{cases}, \tag{8.68}$$

- and the prior probability of c to be the the uniform distributions, i.e $P(c) = \frac{1}{|c|}$, where $|c|$ stands for the number of possible values of c.

The conditional probabilities of all other random nodes are set to be the same as in $\mathcal{N}'$.

For any value node v, $\pi_v{=}\pi'_v$. If v is not a descendent of c, we set

$$\mu_v(\pi_v) = \mu'_v(\pi'_v),$$

and if v is a descendent of c, we set

$$\mu_v(\pi_v) = |c|\mu'_v(\pi'_v).$$

To show that $\mathcal{N}$ and $\mathcal{N}'$ are equivalent, we first notice that for any decision node d, $\pi_d = \pi'_d$; hence $\mathcal{K}$ and $\mathcal{K}'$ have the same policy space. Let δ be a policy, and let P_δ be the joint probability δ induces over all the random and decision nodes of $\mathcal{N}$, and let P'_δ be the joint probability δ induces over all the random and decision nodes of $\mathcal{N}'$. It suffices to show that for any value node v,

$$\sum_{\pi_v} P_\delta(\pi_v)\mu_v(\pi_v) = \sum_{\pi_v} P'_\delta(\pi_v)\mu'_v(\pi_v). \tag{8.69}$$

If v is not a descendent of c, then it cannot be a descendent of b. By Proposition 3.1, both c and b are irrelevant to $P_\delta(\pi_v)$, as well as to $P'_\delta(\pi_d)$. Hence $P_\delta(\pi_d) = P'_\delta(\pi_d)$. Consequently equation 8.69 is true.

Now if v is a descendent of c, then it is also a descendent of b. Consider $P_\delta(b, c|\pi'_b)$ and $P'_\delta(b, c|\pi'_b)$. Noticing that $\pi'_c=\{b\}\cup\pi'_b$, we have

$$\begin{aligned} P_\delta(b, c|\pi'_b) &= P(c)P(b|c, \pi'_b) \\ &= \begin{cases} \frac{1}{|c|}P'(b|\pi'_b) & \text{if } c = (b, \pi'_b) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

and

$$\begin{aligned} P'_\delta(b, c|\pi'_b) &= P'(b|\pi'_b)P'(c|b, \pi'_b) \\ &= \begin{cases} P'(b|\pi'_b) & \text{if } c = (b, \pi'_b) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Hence we get

$$P_\delta(b, c|\pi'_b) = \frac{1}{|c|}P'_\delta(b, c|\pi'_b).$$

Since v is a descendent of both b and c, we have

$$P_\delta(\pi_v) = \frac{1}{|c|}P'_\delta(\pi_v).$$

Therefore

$$\sum_{\pi_v} P_\delta(\pi_v)\mu_v(\pi_v) = \sum_{\pi_v} \frac{1}{|c|} P'_\delta(\pi_v)|c|\mu'_v(\pi_v) = \sum_{\pi_v} P'_\delta(\pi_v)\mu'_v(\pi_v). \tag{8.70}$$

The proof is completed. □

8.5 Induction on the number of random nodes

is a decision node

This section shows how the three operations discussed in the last three sections fit together to form an induction strategy on the number of random nodes. This induction strategy allows us to, in a certain sense, get rid of all the random nodes in the downstream set $Y_{II}(d, \mathcal{K})$ for any d, as shown by the following proposition.

Proposition 8.7 *Let $\mathcal{K}$ be a normal SDDN skeleton without barren nodes and without lonely arcs. Let d_r be a decision node in $\mathcal{K}$. Then there exists another decision network skeleton $\mathcal{K}'$ such that*

- **COND1**: *$\mathcal{K}'$ is normal and stepwise-decomposable, and contains no barren nodes and no lonely arcs;*
- **COND2($\mathcal{K}$):** *The upstream component $K_I(d_r, \mathcal{K}')$ of $\mathcal{K}'$ w.r.t d_r is the same as $K_I(d_r, \mathcal{K})$;*
- **COND3**: *In $\mathcal{K}'$, there are no random nodes in the downstream set $Y_{II}(d_r, \mathcal{K}')$ of π_{d_r}; and*
- **COND4($\mathcal{K}$):** *For any decision network $\mathcal{N}'$ over $\mathcal{K}'$, there exits a decision network $\mathcal{N}$ over $\mathcal{K}$ that is equivalent to $\mathcal{N}'$.*

Proof: We prove this lemma by induction on the number k of random nodes in the downstream set $Y_{II}(d_r, \mathcal{K})$ of π_{d_r}. When $k = 0$, the lemma is trivially true. Suppose the lemma is true for the case of $k = m{-}1$. Now consider the case of $k = m$.

Let d be a decision node in $Y_{II}(d_r, \mathcal{K})$ such that there are random nodes in $Y_{II}(d, \mathcal{K})$, and that either there are no decision nodes in $Y_{II}(d, \mathcal{K})$ or for any decision node $d' \in Y_{II}(d, \mathcal{K})$ there are no random nodes in $Y_{II}(d', \mathcal{K})$.

Since $\mathcal{K}$ contains no barren nodes, there can only be three cases:

1. There exists a random node c in the $Y_{II}(d, \mathcal{K})$ that has at least one parent; or else
2. There exists a root random node c in $Y_{II}(d, \mathcal{K})$ whose children are either value nodes or decision nodes in $Y_{II}(d, \mathcal{K})$; or else
3. Every a random node in $Y_{II}(d, \mathcal{K})$ is a root and has at least one child in π_d.

Case 1): In this case, we short-cut c from $\mathcal{K}$, resulting in $\mathcal{K}^*$. According to Propositions 8.1, $\mathcal{K}^*$ is also a normal SDDN without barren nodes and lonely arcs.

Since there are only $m{-}1$ random node in $Y_{II}(d, \mathcal{K}^*)$, there exits, by the induction hypothesis, a decision network skeleton $\mathcal{K}'$ that satisfies COND1, COND2($\mathcal{K}^*$), COND3, and COND4($\mathcal{K}^*$).

It is easy to see that $\mathcal{K}^*$ satisfies COND2($\mathcal{K}$). By transitivity, $\mathcal{K}'$ also satisfies COND2($\mathcal{K}$).

To see that $\mathcal{K}'$ satisfies COND4($\mathcal{K}$), let $\mathcal{N}'$ be a decision network over $\mathcal{K}'$. Since $\mathcal{K}'$ satisfies COND4($\mathcal{K}^*$), there exist an decision network $\mathcal{N}^*$ over $\mathcal{K}^*$ that is equivalent to $\mathcal{N}'$. Because of Proposition 8.2, there must be a decision network $\mathcal{N}$ over $\mathcal{K}$ that is equivalent to $\mathcal{N}^*$. By transitivity, $\mathcal{N}$ is also equivalent $\mathcal{N}'$. So, $\mathcal{K}'$ also satisfies COND4($\mathcal{K}$). Therefore, the lemma is correct in this case.

Case 2): In this case, we simply remove c from $\mathcal{K}$, resulting in $\mathcal{K}^*$. One can show that the lemma is also true in this case by using Propositions 8.3 and 8.4 and by following the same line of reasoning as in Case 1.

Case 3): In this case, let c be a random node in the downstream set $Y_{II}(d, \mathcal{K})$. Then c is a root and has at least one child in π_d. Let $b \in \pi_d$ be a child of c such that (COND5:) there is no other $b' \in \pi_d$ that is a child of c and a parent of b. Since $\mathcal{K}$ is stepwise-decomposable, b has to be a random node. Because of COND5, the arc $c \rightarrow b$ is reversible. Reverse the arc $c \rightarrow b$ in $\mathcal{K}$, resulting in $\mathcal{K}^*$. By Propositions 8.5, $\mathcal{K}^*$ is normal and stepwise-decomposable, and it contains no barren nodes and no lonely arcs.

There are also m random nodes in $Y_{II}(d, \mathcal{K}^*)$. However in $Y_{II}(d, \mathcal{K}^*)$ there is a random node, namely c, that has at least one parent b. According to Case 1), there must be a decision network skeleton $\mathcal{K}'$ that satisfies COND1, COND2($\mathcal{K}^*$), COND3, and COND4($\mathcal{K}^*$).

Since $\mathcal{K}^*$ satisfies COND2($\mathcal{K}$), so does $\mathcal{K}'$.

To see that $\mathcal{K}'$ satisfies COND4($\mathcal{K}$), let $\mathcal{N}'$ be a decision network over $\mathcal{K}'$. Since $\mathcal{K}'$ satisfies COND4($\mathcal{K}^*$), there exist an decision network $\mathcal{N}^*$ over $\mathcal{K}^*$ that is equivalent to $\mathcal{N}'$.

From the proof of Proposition 8.2, we can have $\mathcal{N}^*$ such that

1. c is a compound variable consisting of a copy of each node in π_c^*, where π_c^* is the set of parents of c in $\mathcal{K}^*$. Since c is a root in $\mathcal{K}$, $\pi_c^* = \{b\} \cup (\pi_b - \{c\}) = \{b\} \cup \pi_b^*$.

2. The conditional probability $P^*(c|\pi_c^*) = P^*(c|b, \pi_b^*)$ is given by

$$P^*(c|b, \pi_b^*) = \begin{cases} 1 & \text{if } c = \pi_c^* = (b, \pi_b^*) \\ 0 & \text{otherwise} \end{cases} . \tag{8.71}$$

Moreover since $c \in Y_{II}(d, \mathcal{K})$ and $\mathcal{K}$ is normal, if a value node is a descendent of c, then it must be a descendent of d, and hence b. So Proposition 8.6 applies and gives us that there is a decision network $\mathcal{N}$ over $\mathcal{K}$ that is equivalent to $\mathcal{N}^*$, and hence to $\mathcal{N}'$. Therefore $\mathcal{K}'$ also COND4($\mathcal{K}$). Thus the lemma is true in this case also. The proof is complete. □

8.6 Induction on the number of decision nodes

This section shows how to carry out induction on the number of decision nodes. First, let us define two properties of value functions that we will do induction with.

Let $\mathcal{N}$ be a decision network whose random and decision variables (nodes) are all binary. Let A be a subset of nodes of $\mathcal{N}$. For any value node v of $\mathcal{N}$, its value function $\mu_v(\pi_v)$ is said to *have property* $Q(A)$ if

- $\mu_v(\pi_v)$ is independent of nodes in $\pi_v \cap A$, and
- $\mu_v(\pi_v) = q_v$ (some real number) when all the nodes in $\pi_v - A$ take the same value, regardless what this value is. When the nodes in $\pi_v - A$ do not all take the same value, $\mu_v(\pi_v)$ is strictly smaller than q_v.

The value function is said to *have property* $Q_1(A)$

- $\mu_v(\pi_v)$ is independent of nodes in $\pi_v \cap A$, and
- $\mu_v(\pi_v) = q_v$ (some real number) when all the nodes in $\pi_v - A$ take the value 1. When there is at least one node in $\pi_v - A$ that does not take the value 1, $\mu_v(\pi_v)$ is strictly smaller than q_v.

Proposition 8.8 *Suppose $\mathcal{N}$ is a normal SDDN with no barren nodes and no lonely arcs. Suppose all the random and decision variables (nodes) of $\mathcal{N}$ are binary. Let d be an SD candidate decision node of $\mathcal{N}$, and let A be a set of nodes in $\mathcal{N}$. Suppose there are* *no random nodes in the downstream set $Y_{II}(d, \mathcal{N})$ of π_d. Then if all the value functions in $\mathcal{N}$ have property $Q(A)$ (or $Q_1(A)$), so do all the value functions of the body $\mathcal{N}_I(d, \mathcal{N})$.*

Proof: Let u be the tail-value node in $\mathcal{N}_I$. It suffices to show that $\mu_u(\pi_d)$ has property $Q(A)$ (or $Q_1(A)$).

First of all, since there are no barren nodes, there must be at least one value node in $Y_{II}(d, \mathcal{N})$. Let $v_1, \ldots, v_m$ be all the value nodes in $Y_{II}(d, \mathcal{N})$.

Let $\mathcal{N}_{II}$ be the tail of $\mathcal{N}$ w.r.t d. Since there are no random nodes in $Y_{II}(d, \mathcal{N})$, $Y_{II}(d, \mathcal{N})$ consists of only value nodes. So we have

$$\mu_u(\pi_d) = E[\mathcal{N}_{II}|\pi_d] \tag{8.72}$$

$$= max_d \sum_{i=1}^{m} \mu_{v_i}(\pi_{v_i}). \tag{8.73}$$

Since all the $\mu_{v_i}(\pi_{v_i})$'s are independent of nodes in $A \cap \pi_d$, so must be $\mu_u(\pi_d)$.

Suppose all the value functions in $\mathcal{N}$ have property $Q(A)$. Since $\mathcal{N}$ is normal, d is a parent of every v_i. Thus when all the nodes in $\pi_d - A$ take the same value, say α, the value of $\mu_u(\pi_d)$ is $\sum_{i=1}^{m} q_{v_i}$, which is achieved when $d = \alpha$.

Now consider the case when there are two nodes x and y in $\pi_d - A$ such that x take the value 0 and y takes 1. Since $\mathcal{N}$ contains no lonely arcs, and there are no random nodes in $Y_{II}(d, \mathcal{N})$, there must be at least one value node, say v_i, which is a child of x and another value node, say v_j (may be the same as v_i), which is a child of y. Because the value functions μ_{v_i} and μ_{v_j} have property $Q(A)$, we have that if $d = 0$, $\mu_{v_j}(\pi_{v_j}) < q_{v_j}$, and if $d = 1$, $\mu_{v_i}(\pi_{v_i}) < q_{v_i}$. Therefore, $\mu_u(\pi_d) < \sum_{i=1}^{m} q_{v_i}$. This shows that μ_u has property $Q(A)$.

To prove the proposition for $Q_1(A)$, suppose all the value functions in $\mathcal{N}$ has property $Q_1(A)$. When all the nodes in $\pi_d - A$ take the value 1, $\mu_u(\pi_d) = \sum_{i=1}^{m} q_{v_i}$, which is achieved when $d = 1$.

Now consider the case when there is one node $x \in \pi_d$, whose value is 0 instead of 1. There is a value node v_j in $Y_{II}(d, \mathcal{N})$ that is a child of x. Because μ_{v_i} has property $Q_1(A)$, $\mu_j(\pi_{v_j}) < q_{v_j}$. Hence, $\mu_u(\pi_d) < \sum_{i=1}^{m} q_{v_i}$. This shows that μ_u has property $Q_1(A)$. The proposition is proved. □

8.6.1 An extension and a corollary

Let d be a decision node in an SDDN $\mathcal{N}$. We can extend the concept of a downstream component from the case when $\mathcal{N}$ is smooth at d to the case when $\mathcal{N}$ is not smooth at d in the same way as we did for the concept of as tail in Section 6.2. Let $\mathcal{N}_{II}(d, \mathcal{N})$ stand for the downstream component of $\mathcal{N}$ w.r.t d. As in Section 3.6, we can define the conditional expected value $E[\mathcal{N}_{II}(d_r, \mathcal{N})|\pi_{d_r}]$.

Proposition 8.9 *Suppose $\mathcal{N}$ is a normal SDDN with no barren nodes and no lonely arcs. Suppose all the random and decision variables (nodes) of $\mathcal{N}$ are binary. Let d_r be a decision node $\mathcal{N}$, and let A be a set of nodes in $\mathcal{N}$. Suppose there are no random nodes in the downstream set $Y_{II}(d_r, \mathcal{N})$. Then if all the value functions in $\mathcal{N}$ have property $Q(A)$ (or $Q_1(A)$), so does the conditional expected value $E[\mathcal{N}_{II}(d_r, \mathcal{N})|\pi_{d_r}]$.*

Proof: This proposition can proved by repeatedly use Proposition 8.8. □

Combining Proposition 8.9 and Proposition 8.7, we get the following corollary.

Corollary 8.1 *Let $\mathcal{K}$ be a normal SDDN skeleton without barren nodes and lonely arcs. Let d_r be a decision node in $\mathcal{K}$. For any subset $A \subseteq \pi_{d_r}$, there exists a decision network $\mathcal{N}$ over $\mathcal{K}$ such that $E[\mathcal{N}_{II}(d_r, \mathcal{N})|\pi_{d_r}]$ has property $Q(A)$ (or $Q_1(A)$).*

Proof: Let $\mathcal{K}'$ be an SDDN skeleton as in Proposition 8.7. There are no random nodes in $Y_{II}(d_r, \mathcal{K}')$. Construct a decision network $\mathcal{N}'$ over $\mathcal{K}'$ as follows. Let all the random and decision variables be binary. For any value node v, set

$$\mu_v(\pi_v) = \begin{cases} 1 & \text{if all the variable in } \pi_v - A \text{ take the same value} \\ 0 & \text{otherwise} \end{cases} . \tag{8.74}$$

Then all the value function in $\mathcal{N}'$ have property $Q(A)$. By Proposition 8.9, $E[\mathcal{N}_{II}(d_r, \mathcal{N}')|\pi'_{d_r}]$ also has property $Q(A)$. According to Proposition 8.7, there is a decision network $\mathcal{N}$ over $\mathcal{K}$ that is equivalent to $\mathcal{N}'$. Since the upstream Component $K_I(d_r, \mathcal{K})$ of $\mathcal{K}$ is the same as $K_I(d_r, \mathcal{K}')$, $\pi_{d_r} = \pi'_{d_r}$. Thus we have $E[\mathcal{N}_{II}(d_r, \mathcal{N})|\pi_{d_r}] = E[\mathcal{N}_{II}(d_r\mathcal{N}')|\pi'_{d_r}]$. Therefore $E[\mathcal{N}_{II}(d_r, \mathcal{N})|\pi_{d_r}]$ has property $Q(A)$.

The $Q_1(A)$ part can be proved in the same way. □

8.7 Lonely arcs and removable arcs

We are now ready to prove a theorem about the relationship between removable arcs and lonely arcs.

Theorem 8.1 *Let $\mathcal{K}$ be a normal SDDN skeleton without barren nodes. If $\mathcal{K}$ contains no lonely arcs, then it contains no removable arcs.*

Before proving this theorem, let us point out an important implication.

Corollary 8.2 *In a normal SDDN skeleton, an arc into a decision node is removable if and only if it is a potential lonely arc.* □

To put the corollary in another way, in a normal SDDN, potential lonely arcs are all the removable arcs that can be graphically identified without resorting to numerical computations.

Proof of Theorem 8.1: Let d_r be a decision node of $\mathcal{K}$. Let c be an arbitrary node in π_{d_r}. We need to show that the arc $c \rightarrow d_r$ is not removable.

Because of Proposition 8.7, we can assume that there are no random nodes in the downstream set $Y_{II}(d_r, \mathcal{K})$.

Let $\mathcal{N}$ be an SDDN over $\mathcal{K}$. Assume all the random and decision nodes are binary. Let $A = \pi_{d_r} - \{c\}$. For any value node v in $\mathcal{N}$, set μ_v to be

$$\mu_v(\pi_v) = \begin{cases} 1 & \text{when all the variables in } \pi_v - A \text{ take the same value} \\ 0 & \text{otherwise} \end{cases} \tag{8.75}$$

Then the value functions have property $Q(A)$.

We find an SD candidate decision node d, computes its body $\mathcal{N}_I(d, \mathcal{N})$ w.r.t d. It is easy to verify that $\mathcal{N}_I(d, \mathcal{N})$ is also stepwise-decomposable and normal, and it contains no barren nodes and no lonely arcs. According to the Proposition 8.8, all the value functions $\mathcal{N}_I$ have property $Q(A)$.

We then find an SD candidate decision node of $\mathcal{N}_I(d, \mathcal{N})$, computes its body, and so on so forth. Eventually, we will obtain a normal SDDN, denoted by $\mathcal{N}_r$, in which d_r is an SD candidate, and which contains no barren nodes and no lonely arcs. Furthermore, all the the value functions in $\mathcal{N}_r$ have property $Q(A)$.

Since $\mathcal{N}_r$ contains no lonely arcs, and there are no random nodes in the downstream set $Y_{II}(d_r, \mathcal{N})$, there must be at least one value node $v \in Y_{II}(d_r, \mathcal{N})$ that is a child of c. In the mean time, $\mathcal{N}_r$ is also normal, so this value node v is also a child of d_r. All the value functions of $\mathcal{N}_r$ have the $Q(A)$ property. Since $A = \pi_{d_r} - \{c\}$, all the value functions in the tail $\mathcal{N}_{II}(d_r, \mathcal{N}_r)$ depend only on d_r or v or both. Therefore when c=0, the optimal decision for d_r is 0, and when c=1, the optimal decision for d_r is 1. Thus d_r depends on c and hence the arc $c \rightarrow d_r$ is not removable. The theorem is proved. □

8.8 Stepwise-solvability and stepwise-decomposability

This section proves the following theorem about the relationship between stepwise-decomposability and stepwise-solvability.

Theorem 8.2 *Suppose $\mathcal{K}$ is a normal decision network skeleton with no barren nodes and no lonely arcs. Then $\mathcal{K}$ is stepwise-solvable if and only if it is stepwise-decomposable.*

A decision network skeleton in Figure 3.15 is not stepwise-decomposable, hence it is not stepwise-solvable. Consequently, as we predicted in Section 3.4, with appropriate probabilities and value functions, optimal policies can be found only by considering the two decisions simultaneously.

The remainder of this section is to prove Theorem 8.2. In a decision network, a *decision root node* is a root node that is also a decision node.

Lemma 8.3 *Suppose $\mathcal{K}$ is a normal decision network without barren nodes. Suppose d is a decision node in $\mathcal{K}$. If there are decision root nodes in the downstream set $Y_{II}(d, \mathcal{K})$, then d cannot be an SS candidate node.*

Proof: Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ by replacing with deterministic nodes those decision nodes that are different from d and have at least one parent. It suffices to show that (Statement1:) d is not an SS candidate node in $\mathcal{K}'$.

Let π'_x be the set of parents of a node x in $\mathcal{K}'$. We show Statement1 by induction on the number k of random nodes, including deterministic nodes, in $Y_{II}(d, \mathcal{K}')$. When $k = 0$, all the nodes in $Y_{II}(d, \mathcal{K}')$ are either decision root nodes or value nodes; and there exists at least one decision root node. By the definition of $Y_{II}(d, \mathcal{K}')$, there must be one decision root node $d' \in Y_{II}(d, \mathcal{K}')$ such that d' has a value node child v. Since $\mathcal{K}$ is normal, so is $\mathcal{K}'$. Hence, there exits a directed path from d to v. Because all the nodes in $Y_{II}(d, \mathcal{K}')$ are either decision nodes or value nodes, d must be a parent of v.

Construct a decision network $\mathcal{N}'$ over $\mathcal{K}'$ as follows. Let all the random and decision variables be binary; let the value functions of the value nodes other than v all be zero; and set

$$\mu_v(\pi_v) = \begin{cases} 1 & \text{if } d{=}d' \\ 0 & \text{otherwise} \end{cases} \tag{8.76}$$

We see that if $d' = 1$, the optimal decision for d is $d = 1$; and if $d' = 0$, the optimal decision for d is $d = 0$. Therefore the optimal policy for d depends on the policy for d' [1]. Consequently d is not an SS candidate node. So, Statement1 is true in the case $k = 0$.

Assume Statement1 is true for the case of $k = m{-}1$. Consider the case of $k = m$. There are three subcases.

Subcase 1). There is a random node $c{\in}Y_{II}(d, \mathcal{K}')$ that has at least one parent. In this case, we can short-cut c from $\mathcal{K}'$, resulting in $\mathcal{K}^*$. According to Proposition 8.1, $\mathcal{K}^*$ is also normal and contains no barren nodes. Since there are only $m{-}1$ random nodes in $Y_{II}(d, \mathcal{K}^*)$, by the induction hypothesis, d is not an SS candidate node in $\mathcal{K}^*$. Through Proposition 8.2, this implies that d is not an SS candidate node in $\mathcal{K}'$.

Subcase 2). There is random node $c{\in}Y_{II}(d, \mathcal{K}')$ whose children all are value nodes. In this case, we can simply remove c from $\mathcal{K}'$, resulting in $\mathcal{K}^*$. Using Propositions 8.3 and 8.4 and following the same line of reasoning as in Subcase 1), we can show that Statement1 is true in this subcase.

Subcase 3). Every random node $c{\in}Y_{II}(d, \mathcal{K}')$ is a root, and it has at least one child in π'_d. Let $b{\in}\pi'_d$ a a child of c such that there is no other $b'{\in}\pi'_d$ that is a child of c and a parent of b. By the definition of $\mathcal{K}'$, b is a random (maybe deterministic) node. By the choice of b, the arc $c{\rightarrow}b$ is reversible. We reverse the arc $c{\rightarrow}b$ in $\mathcal{K}'$ to get $\mathcal{K}''$.

[1] For later convenience, let us remark that this conclusion follows for any value function μ_v of v that has property $Q(\{d, d'\})$.

By Proposition 8.5, $\mathcal{K}''$ is also normal and contains no barren nodes. There are m random nodes in $Y_{II}(d, \mathcal{K}'')$, one of which, namely c, has parents. As in Subcase 1), we short-cut c from $\mathcal{K}''$, resulting $\mathcal{K}^*$. By the induction hypothesis, there is a decision network $\mathcal{N}^*$ over $\mathcal{K}^*$ in which the optimal decision function of d depends on the decision policy of some other decision node d'.

By Proposition 8.2, there exists a decision network $\mathcal{N}''$ over $\mathcal{K}''$ that is equivalent to $\mathcal{N}^*$; and by the proof of Proposition 8.2, we conclude that $\mathcal{N}''$ can be such that

1. c is a compound variable consisting of a copy of each node in π''_c, where π''_c is the set of parents of c in $\mathcal{K}''$. Since c is a root in $\mathcal{K}'$, $\pi''_c = \{b\} \cup (\pi'_b - \{c\}) = \{b\} \cup \pi''_b$.

2. The conditional probability $P''(c|\pi''_c) = P''(c|b, \pi''_b)$ is given by

$$P''(c|b, \pi''_b) = \begin{cases} 1 & \text{if } c = \pi''_c = (b, \pi''_b) \\ 0 & \text{otherwise} \end{cases} . \tag{8.77}$$

Moreover, since $c \in Y_{II}(d, \mathcal{K}')$ and $\mathcal{K}'$ is normal, if a value node is a descendent of c, then it must be a descendent of d, and hence b. So, Proposition 8.6 applies, and gives us that there is a decision network $\mathcal{N}'$ over $\mathcal{K}$ that is equivalent to $\mathcal{N}''$, and hence to $\mathcal{N}^*$. Therefore in $\mathcal{N}'$, the optimal decision function of d depends on the decision function of some other decision node. Consequently, d is not an SS candidate node in $\mathcal{K}'$. The proof is complete. □

Lemma 8.4 *Let $\mathcal{K}$ be a normal decision network skeleton with no barren nodes. Suppose d is a decision node in $\mathcal{K}$, and suppose there are no decision root nodes in the downstream set $Y_{II}(d, \mathcal{K})$. If there exists at least one decision node, other than d, in $Y_{II}(d, \mathcal{K})$, then d is not an SS candidate node.*

Proof: Let $d' \neq d$ be a decision node in $Y_{II}(d, \mathcal{K})$. Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ be replacing all the decision nodes other than d and d' by deterministic nodes. It suffices to show that (Statement1:) d is not an SS candidate node in $\mathcal{K}'$.

We prove Statement1 by induction on the number k of random nodes, including deterministic nodes, in $Y_{II}(d, \mathcal{K}')$. When $k = 0$, $Y_{II}(d, \mathcal{K})$ consists of d, d', and value nodes. By the definition of $Y_{II}(d, \mathcal{K})$, there must exist a value node v that is a child of d'. Since $\mathcal{K}$ is normal, so is $\mathcal{K}'$. Hence, d there is a directed path from d to v. There are two cases: either d is a parent of v, or d is a parent of d'.

It has been shown in the proof of Lemma 8.3 that when both d and d' are parents to v, d is not an SS candidate node. Now consider the case when d is a parent of d'. Construct a decision network $\mathcal{N}'$ over $\mathcal{K}'$ as follows. Let all the random and decision variables be binary; let the value functions of all the value nodes other than v be zero; and let

$$\mu_v(\pi_v) = \begin{cases} 1 & \text{if } d'{=}1 \\ 0 & \text{otherwise} \end{cases} \tag{8.78}$$

Noticing $d \in \pi'_{d'}$, we have that when the decision function δ' of d' is such that $\delta'(\pi'_{d'}) = d$, the optimal decision for d is $d = 1$; and when the decision function δ' of d' is such that $\delta'(\pi'_{d'}) = 1 - d$, the optimal decision for d is $d = 0$. Therefore, the optimal decision function for d depends on the decision function of d' [2]. Thus, d is not an SS candidate node, and Statement1 is true for the case of $k = 0$.

Assume Statement1 is true for the case of $k = m - 1$. We can prove that Statement1 is true for the case of $k = m$ by following the same line of reasoning as in the proof of Lemma 8.3. There is only one issue that demands special attention. In Subcase 3), we

[2] For later convenience, let us remark that this conclusion follows for any value function μ_v of v that has property $Q_1(\{d'\})$.

need to reverse the arc $c \rightarrow b$. This can only be done when c is a random node and not a deterministic node. Since there are no root decision nodes in $Y_{II}(d, \mathcal{K})$, there are no deterministic root nodes in $Y_{II}(d, \mathcal{K}')$. Thus, c can not be a deterministic node. The lemma is proved. □

Lemma 8.5 *Suppose $\mathcal{K}$ is a normal decision network skeleton with no barren nodes. Let d be a decision node in $\mathcal{K}$. Suppose there are no decision nodes in the downstream set $Y_{II}(d, \mathcal{K})$. If there is a decision node $d' \in \pi_d$ such that at least one of the parents of d' are in $Y_{II}(d, \mathcal{K})$, then d is not an SS candidate node.*

Proof: Let $\mathcal{K}'$ be the decision network skeleton obtained from $\mathcal{K}$ be replacing all the decision nodes other than d and d' by deterministic nodes. It suffices to show that (Statement1:) d is not an SS candidate node in $\mathcal{K}'$.

We prove Statement1 by induction on the number k of random nodes, including deterministic nodes, in $Y_{II}(d, \mathcal{K}') - \pi'_{d'}$. When k=0, $Y_{II}(d, \mathcal{K}')$ consists of the parents of d', and value nodes. By the definition of $Y_{II}(d, \mathcal{K}')$, there must be at least one parent c of d' that has a value node child v. Since $\mathcal{K}$ is normal, so is $\mathcal{K}'$. Thus, d must also be a parent of v.

Construct a decision network $\mathcal{N}'$ over $\mathcal{K}'$ as follows. Let all the random and decision variables be binary; let the value functions of the value nodes other than v all be zero; and set

$$\mu_v(\pi_v) = \begin{cases} 1 & \text{if } d{=}c \\ 0 & \text{otherwise} \end{cases} \tag{8.79}$$

Noticing $c \in \pi'_{d'}$ and $d' \in \pi'_d$, we have that if the decision function δ' for d' is such that $\delta'(\pi'_{d'}) = c$, then the optimal decision function δ^o of d is such that $\delta^o(\pi'_d) = d'$; and if the decision function δ' for d' is such that $\delta'(\pi'_{d'}) = 1{-}c$, then the optimal decision function

δ^o of d is such that $\delta^o(\pi'_d) = 1{-}d'$. That is, the optimal decision function of d depends on the decision function of d' [3]. Consequently d is not an SS candidate node in $\mathcal{K}'$; Statement1 is true for the case of $k = 0$.

Assume Statement1 is true in the case of $k = m{-}1$. We can prove that Statement1 is true in the case of $k = m$ in the same way as in the proof of Lemma 8.3. The lemma is thus proved. □

Proposition 8.10 *Suppose $\mathcal{K}$ is a normal decision network skeleton with no barren nodes. Let d be a decision node in $\mathcal{K}$. If d is an SS candidate node, then it is also an SD candidate node.*

Proof: Since d is an SS candidate node, by Lemmas 8.3 and 8.4, there cannot be decision nodes in the downstream set $Y_{II}(d, \mathcal{K})$. By Lemma 8.5, there cannot be decision nodes which have parents in $Y_{II}(d, \mathcal{K})$. Therefore, π_d m-separates d from all other decision nodes and their parents; i.e d is an SD candidate node. □

Corollary 8.3 *Let $\mathcal{K}$ be a normal decision network skeleton with no barren nodes. Let d be a decision node in $\mathcal{K}$. Suppose the downstream component $\mathcal{K}_{II}(d, \mathcal{K})$ is stepwise-decomposable and contains no lonely arcs. Then in the upstream component $\mathcal{K}_I(d, \mathcal{K})$, a decision node is an SD candidate node if it is an SS candidate node.*

Proof: One can prove this corollary in the same way as we prove Proposition 8.10. The only issue that demands special attention is that in $\mathcal{K}_I$, there is a downstream-value node u. We may not be able to arbitrarily set the value function μ_v of u; it has to be the optimal conditional expected value $\mu_v{=}E[\mathcal{N}_{II}|\pi_d]$ of a decision network $\mathcal{N}_{II}$ over $\mathcal{K}_{II}$. As we mentioned in Footnotes 1, 2, and 3, we need only to be able to set μ_v such that it has

[3] For later convenience, let us remark that this conclusion follows for any value function μ_v of v that has property $Q(\{d', c\})$.

property $Q(\{x, y\})$ for some $x, y \in \pi_d$, or has property $Q_1(\{x\})$ for some $x \in \pi_d$. Since $\mathcal{K}_{II}$ is normal and stepwise-decomposable, and contains no barren nodes and lonely arcs, this is possible according to Corollary 8.1. □

In a decision network skeleton $\mathcal{K}$, a decision node d is a *potential SD candidate node* if either it is an SD candidate node, or there exists an SD candidate node $d'(\neq d)$ such that d is a potential SD candidate node in the body $\mathcal{K}_I(d', \mathcal{K})$. It is easy to see that a decision network skeleton is stepwise-decomposable if and only if every decision node is a potential SD candidate node.

A potential SD candidate node d is the *oldest*, if there is no other potential SD candidate node that is an ancestor of d.

Proof of Theorem 8.2: Let us first show stepwise-decomposability implies stepwise-solvability. Suppose $\mathcal{K}$ is stepwise-decomposable. Let $\mathcal{N}$ be an arbitrary decision network over $\mathcal{K}$. We need to show that $\mathcal{N}$ is stepwise-solvable. Let $\mathcal{N}'$ be the output network of SMOOTHING($\mathcal{N}$). Then, $\mathcal{N}'$ is a smooth SDDN. According to Theorem 5.1, $\mathcal{N}'$ is stepwise-solvable. Since $\mathcal{N}'$ and $\mathcal{N}$ is equivalent, $\mathcal{N}$ is also stepwise-solvable.

To prove that stepwise-solvability also implies stepwise-decomposability, it suffices to show that if there exist decision nodes in $\mathcal{K}$ that are not potential SD candidate nodes, then $\mathcal{K}$ is not stepwise-solvable.

For simplicity, let us assume that there is only one oldest potential candidate node d^o. Then none of the decision nodes in the upstream component $\mathcal{K}_I(d^o, \mathcal{K})$ are SD candidate nodes. By Corollary 8.3, none decision nodes in $\mathcal{K}_I$ can be SS candidate nodes. Therefore $\mathcal{K}$ is not stepwise-solvable. The theorem is proved. □

Chapter 9

SDDN's and Markov decision processes

In the introduction, we have shown how finite stage Markov decision processes (MDP's)[1] can be represented as SDDN's. This chapter shows that an SDDN can be condensed into an equivalent MDP.

This practice is interesting for two reasons. Conceptually, it reveals the close relationships between SDDN's and MDP's: MDP's are special SDDN's and SDDN's can be condensed into MDP's.

Computationally, the concept of condensation opens up the possibility of parallel computation in evaluating SDDN's (Section 9.2); it enables us to exploit the asymmetric nature of decision problems (Section 9.5); and it also leads to an incremental approach for computing the values of information (Zhang *et al* 1993b).

The organization of this chapter is as follows. The concept of sections in smooth regular SDDN's is introduced in Section 9.1. Section 9.2 gives the definition of condensation of smooth regular SDDN's, and points out the possibility of parallel computation. Section 9.3 shows that a smooth regular SDDN is equivalent to its condensation. Non-smooth regular and irregular SDDN's are treated in Section 9.4. Section 9.5 exploits the asymmetric nature of decision problems to minimize the number of states of the variables in condensations. On the basis of condensation, Section 9.6 proposes a two stage approach for evaluating SDDN's, which is compared to the approaches by Howard and Matheson (1984) and Cooper (1989) in Section 9.6.1.

[1] In this chapter, when talking about Markov decision processes we always mean finite stage Markov decision processes.

9.1 Sections in smooth regular SDDN's

The first step toward the concept of condensation is to introduce the concept of sections for smooth regular SDDN's. Let $\mathcal{N}$ be a smooth regular SDDN. Let d_1, d_2, $\ldots$, d_k be all the decision nodes. Since $\mathcal{N}$ is regular, there is a total ordering among the decision nodes. Let the total ordering be as indicated by the subscriptions. More explicitly, let us assume that d_i *directly precedes* d_{i+1} in the sense that there is no other decision node d_j such that d_i precedes d_j and d_j precedes d_{i+1}. In this case, we also say that d_{i+1} *directly succeeds* d_i.

For any $i \in \{1, 2, \ldots, k-1\}$, the *section of* $\mathcal{N}$ *from* π_{d_i} *to* $\pi_{d_{1+1}}$, denoted by $\mathcal{N}(d_i, d_{i+1})$, is the subnetwork of $\mathcal{N}$ that consists of the following nodes:

1. the nodes in $\pi_{d_i} \cup \pi_{d_{i+1}}$,
2. the nodes that are in both the downstream set $Y_{II}(d_i, \mathcal{N})$ of π_{d_i} and the upstream set $Y_I(d_{i+1}, \mathcal{N})$ of $\pi_{d_{i+1}}$.

The graphical connections among the nodes remain the same as in the $\mathcal{N}$ except that all the arcs among the nodes in $\pi_{d_i} \cup \{d_i\}$ are removed. The nodes outside $\pi_{d_i} \cup \{d_i\}$ are either random nodes or value nodes; their conditional probabilities or value functions remain the same as in $\mathcal{N}$. The nodes in $\pi_{d_i} \cup \{d_i\}$ are either decision nodes or random nodes. There are no conditional probabilities associated with random nodes in $\pi_{d_i} \cup \{d_i\}$.

The *initial section* $\mathcal{N}(-, d_1)$ consists of the nodes in π_{d_1} and the nodes in the upstream set $Y_I(d_1, \mathcal{N})$ of π_{d_1}. It consists of only random and value nodes, whose conditional probabilities or value functions remain the same as in $\mathcal{N}$.

Value nodes in the initial section do not affect the optimal policies, even though they do contribute to the optimal expected value. From now on, we shall assume there are no value nodes in the initial section, with the understanding that they, if any, are taken

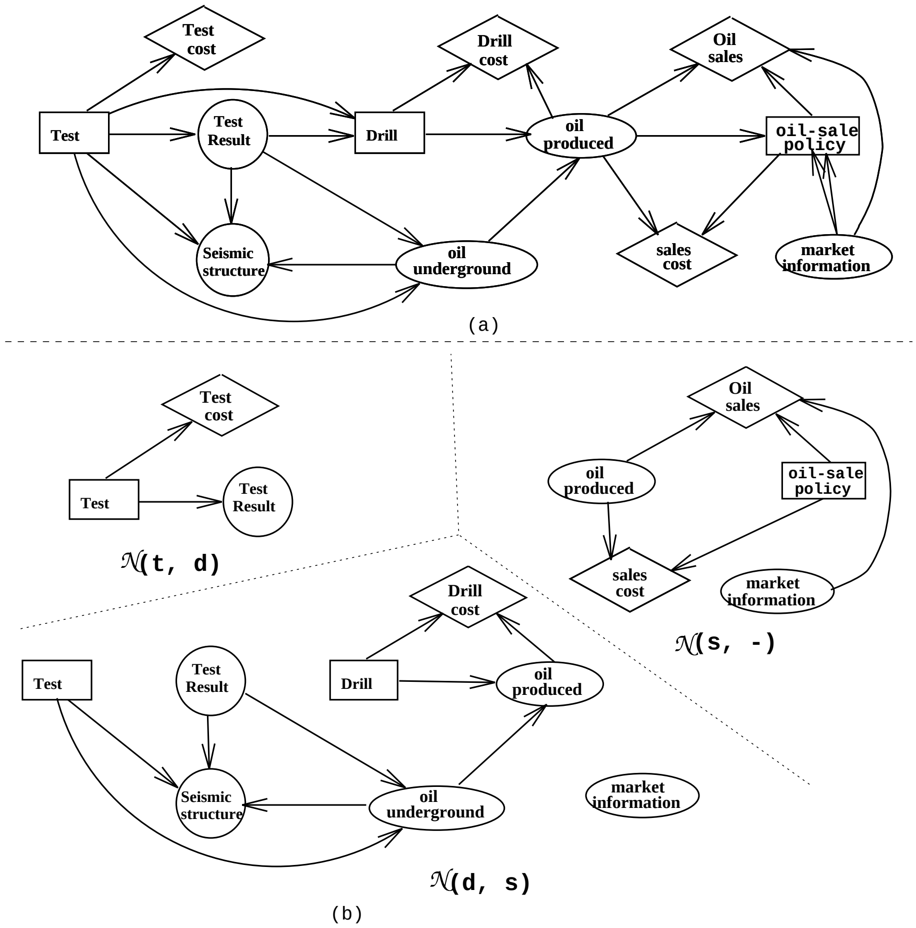

Figure 9.27: A regular SDDN and its sections: `t` stands for `test`, `d` stands for `drill`, and `s` stands for `oil-sale-police`.

care of by some preprocessing measure.

The *terminal section* $\mathcal{N}(d_k, -)$ consists of the nodes in the π_{d_k} and the nodes in the downstream set $Y_{II}(d_k, \mathcal{N})$ of π_{d_k}. The graphical connections, the conditional probabilities, and the value functions in the terminal section are specified in the same way as in the case of ordinary sections.

As an example, consider the decision network in Figure 9.27 (a), which is a reproduction of Figure 6.22 (b). The network is smooth, regular and stepwise-decomposable.

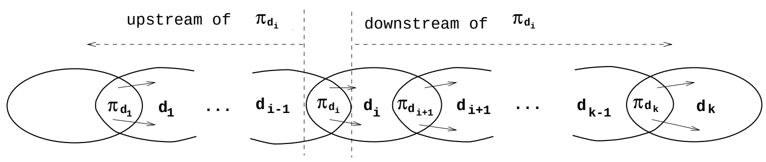

Figure 9.28: An abstract view of a smooth regular SDDN. Smoothness is indicated by the fact that all the arrows from the $\pi'_{d_i}s$ are pointing downstream.

Let us denote this SDDN by $\mathcal{N}$. Let us also denote the variable `test` by `t`, `drill` by `d`, `oil-sale-policy` by `s`, `drill-cost` by `dc`, `test-result` by `tr`, `oil-produced` by `op`, and `market-information` by `mi`.

There are four sections in $\mathcal{N}$: $\mathcal{N}(-, \mathtt{t})$, $\mathcal{N}(\mathtt{t}, \mathtt{d})$, $\mathcal{N}(\mathtt{d}, \mathtt{s})$, and $\mathcal{N}(\mathtt{s}, -)$. The initial section $\mathcal{N}(-, \mathtt{t})$ contains no nodes. All the other sections are shown in Figure 9.27 (b).

At this point, we wish to emphasize that in each section $\mathcal{N}(d_i, d_{i+1})$, all the decision nodes are in $\pi_{d_i} \cup \{d_i\}$. Since $\mathcal{N}$ is smooth, in $\mathcal{N}(d_i, d_{i+1})$ there are no arcs pointing toward those decision nodes. Thus, they can be regarded random nodes with no prior probabilities. Consequently, $\mathcal{N}(d_i, d_{i+1})$ can be viewed as a semi-Bayesian network with value nodes.

9.1.1 An abstract view of a regular smooth SDDN

The concept of sections provides us with a proper perspective for viewing smooth regular SDDN's. A regular smooth SDDN $\mathcal{N}$ can be thought of as consisting of a chain sections

$$\mathcal{N}(-, d_1), \mathcal{N}(d_1, d_2), \ldots, \mathcal{N}(d_{k-1}, d_k), \mathcal{N}(d_k, -).$$

Two neighboring sections $\mathcal{N}(d_{i-1}, d_i)$ and $\mathcal{N}(d_i, d_{i+1})$ share the nodes in π_{d_i}, which m-separate the other nodes in $\mathcal{N}(d_{i-1}, d_i)$ from all the other nodes $\mathcal{N}(d_i, d_{i+1})$. Figure 9.28 shows this abstract view of a regular SDDN.

9.1.2 Conditional probabilities in a section

In each section $\mathcal{N}(d_i, d_{i+1})$, one can compute $P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$ — the conditional probability of the $\pi_{d_{i+1}}$ given π_{d_i} and d_i in $\mathcal{N}(d_i, d_{i+1})$. In the initial section $\mathcal{N}(-, d_1)$, one can compute $P_{\mathcal{N}(-,d_1)}(\pi_{d_1})$ — the marginal probability π_1 in $\mathcal{N}(-, d_1)$.

Lemma 9.1 *Let $\mathcal{N}$ be a smooth regular SDDN and d_i and d_{i+1} be two consecutive decision nodes. For any policy δ for $\mathcal{N}$, let P_δ denote the joint probability determined by δ over all the decision and random nodes. Then we have*

$$P_\delta(\pi_{d_{i+1}}|\pi_{d_i}, d_i) = P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{d_{i+1}}|\pi_{d_i}, d_i), \tag{9.80}$$

and

$$P_\delta(\pi_{d_1}) = P_{\mathcal{N}(-,d_1)}(\pi_{d_1}). \tag{9.81}$$

Proof: According to Proposition 3.1, all the nodes in the downstream set of $\pi_{d_{i+1}}$ are irrelevant to $P_\delta(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$. Hence, they can be harmlessly pruned from $\mathcal{N}$. According to Proposition 3.2, all the nodes in the upstream set of π_{d_i} are also irrelevant to $P_\delta(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$. Hence, they can also be harmlessly pruned from $\mathcal{N}$. After pruning the nodes in the downstream set of $\pi_{d_{i+1}}$ and those in the upstream set of π_{d_i}, what is left of $\mathcal{N}$ is exactly $\mathcal{N}(d_i, d_{i+1})$. The lemma is therefore proved. □

In words, this lemma says that the conditional of probability of $\pi_{d_{i+1}}$ given π_{d_i} and d_i is independent of the policy δ and can be computed locally in the section $\mathcal{N}(d_i, d_{i+1})$.

9.1.3 The local value function of a section

We now turn to value functions. For a value node v_{ij} in $\mathcal{N}(d_i, d_{i+1})$, one can compute the conditional probability $P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{v_{ij}}|\pi_{d_i}, d_i)$.

Lemma 9.2 *Let $\mathcal{N}$ be a smooth regular SDDN and d_i and d_{i+1} be two consecutive decision nodes. For any policy δ for $\mathcal{N}$, let P_δ denote the joint probability determined by δ over all the decision and random nodes. For any value node v_{ij} in the section $\mathcal{N}(d_i, d_{i+1})$, we have*

$$P_\delta(\pi_{v_{ij}}|\pi_{d_i}, d_i) = P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{v_{ij}}|\pi_{d_i}, d_i). \tag{9.82}$$

Proof: The same as the proof of Lemma 9.1. □

Define a function $f'_{v_{ij}} : \Omega_{\pi_{d_i}} \times \Omega_{d_i} \rightarrow R^1$ by

$$f'_{v_{ij}}(\pi_{d_i}, d_i) = \sum_{\pi_{v_{ij}}-(\pi_{d_i}\cup\{d_i\})} P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{v_{ij}}|\pi_{d_i}, d_i) f_{v_{ij}}(\pi_{v_{ij}}) \tag{9.83}$$

where $f_{v_{ij}}$ is the value function of v_{ij} in $\mathcal{N}$.

Let $v_{i1}, \ldots, v_{im_i}$ be all the value nodes in the section $\mathcal{N}(d_i, d_{i+1})$. The *local value function* $f_i : \Omega_{\pi_{d_i}} \times \Omega_{d_i} \rightarrow R^1$ of the section $\mathcal{N}(d_i, d_{i+1})$ is defined by

$$f_i(\pi_{d_i}, d_i) = \sum_{j=1}^{m_i} f'_{v_{ij}}(\pi_{d_i}, d_i). \tag{9.84}$$

Note that if there are no value node in the section, then f_i is the constant 0. In particular, the local value function of the initial section is zero since it is assumed to contain no value nodes.

9.1.4 Comparison with decision windows

The concept of decision windows is introduced by Shachter (1990) as a way to understand information arcs—arcs into decision nodes[2]. The window W_i consists of those random nodes that are observed for the first time between the decision maker's choice for d_{i-1} and her/his choice for d_i.

[2]Recall that in influence diagrams arcs into decision nodes are interpreted indication of information availability.

The major difference between sections and windows is that sections correspond to graphical separation, while windows do not. A section $\mathcal{N}(d_{i-1}, d_i)$ can be compared to an interval on the real line: the "left end point" is $\pi_{d_{i-1}}$, and the "right end point" is π_{d_i}. Two neighboring sections are separated by their common "end point"; two sections that are not neighbors are separated by the sections in between.

On the other hand, a node in a window W_i can be connected to a node in any other window W_j; and there is no concept of "end points". Consequently, with windows we do not have the the two lemmas given in the previous two subsections.

9.2 Condensing smooth regular SDDN's

Intuitively, condensing a smooth regular SDDN $\mathcal{N}$ means doing the following in each section $\mathcal{N}(d_i, d_{i+1})$ of $\mathcal{N}$: (1) delete all the random nodes that are neither in the π_{d_i} nor in $\pi_{d_{i+1}}$, (2) combine all the value nodes into one single value node v_i^c, and (3) group the nodes in π_{d_i} into one compound variable x_i. What results is a Markov decision process (MDP). Now the formal definition.

The *condensation* of $\mathcal{N}$, denoted by $\mathcal{N}^c$, is an MDP given as follows:

1. It consists of the following nodes:

    - Random nodes x_i $(1 \leq i \leq k)$, where x_i is the compound variable consists of all the nodes in π_{d_i} when $\pi_{d_i} \neq \emptyset$. When $\pi_{d_i} = \emptyset$, x_i is a variable that has only one possible value;
    - The same decision nodes d_i $(1 \leq i \leq k)$ as in $\mathcal{N}$; and
    - Value nodes v_i^c $(1 \leq i \leq k)$.

2. The graphical connections among the nodes are as follows:

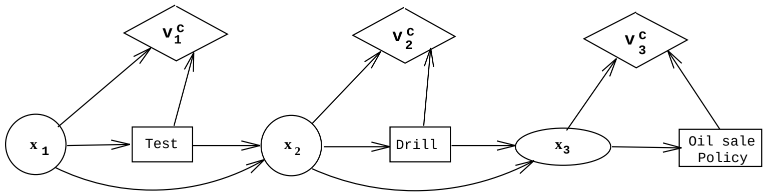

Figure 9.29: The skeleton of the condensation of the SDDN in Figure 9.27 (a) .

- For any $i \in \{1, 2, \ldots, k\}$; there is an arc from x_i to d_i, an arc from x_i to v_i^c, and an arc from d_i to v_i^c.
- For any $i \in \{1, 2, \ldots, k-1\}$, there is an arc from x_i to x_{i+1} and an arc from d_i to x_{i+1}.

3. The conditional probabilities and value functions are as follows:

   - The conditional probability $P^c(x_{i+1}|x_i, d_i)$ ($i \in \{1, \ldots, k-1\}$) is defined to be $P_{\mathcal{N}(d_{i+1}, d_i)}(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$.
   - The prior probability $P^c(x_1)$ is defined to be $P_{\mathcal{N}(-, d_1)}(\pi_{d_1})$.
   - The value function $f_{v_i^c}$ for v_i^c ($i \in \{1, \ldots, k\}$) is defined to be the local value function f_i.

Since the condensation $\mathcal{N}^c$ is an MDP, we shall sometimes refer to $P^c(x_{i+1}|x_i, d)$ as the *transition probability* from x_i to x_{i+1}.

Figure 9.29 depicts the skeleton of the condensation of the SDDN in 9.27 (a). Since `test` has no parent, x_1 is a degenerate variable with only one value. The variable x_2 stands for the compound variable consisting of `test` and `test-result`, and x_3 stands for the compound variable consisting of `oil-produced` and `oil-market`.

The prior probability of x_1 is trivially defined, the transition probability $P^c(x_2|x_1, \mathtt{t})$

is set to be $P_{\mathcal{N}(\mathtt{t,d})}(\mathtt{t,\ tr|t})$, and the transition probability $P^c(x_3|x_2, \mathtt{d})$ is set to be $P_{\mathcal{N}(\mathtt{d,s})}(\mathtt{op,\ mi|t,\ tr,\ d})$.

The value function $f_{v_1^c}$ for the value node v_1^c is a representation of `test-cost`, $f_{v_2^c}$ is a representation of `drill-cost`, and $f_{v_3^c}$ is a representation of the summation `oil-produced` and `sale-cost`.

The SDDN in Figure 9.27 (a) is not in the form of an MDP. However, its condensation, as shown in Figure 9.29, is a Markov decision process. In particular, each random node separates the network into two parts.

The condensation of a smooth regular SDDN is usually not a homogeneous MDP (Denardo 1982). The frames of the x_i's are different from one another.

9.2.1 Parallel computations

In the process of condensation, the following numerical computations are carried out in each section $\mathcal{N}(d_i, d_{i+1})$:

- The calculation of the conditional probabilities $P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$, and $P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{v_{ij}}|\pi_{d_i}, d_i)$ (for each value node v_{ij}), and
- The summations as indicated by equations (9.83, 9.84).

Those conditional probabilities can be obtained from the marginals $P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{d_i}, d_i)$, $P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{d_{i+1}}, \pi_{d_i}, d_i)$, and $P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{v_{ij}}, \pi_{d_i}, d_i)$ (for each value node v_{ij}); and the marginals can in turn be computed by using clique tree propagation, so that all the marginals can be computed by visiting each clique at most twice.

An important observation is that the numerical computations in different sections can be carried out in parallel.

9.3 Equivalence between SDDN's and their condensations

This section shows the following theorem.

Theorem 9.1 *A smooth regular SDDN is equivalent to its condensation.*

Proof: Let $\mathcal{N}$ be a smooth regular SDDN and $\mathcal{N}^c$ be its condensation. Let π_{d_i} denote the set of parents of d_i in $\mathcal{N}$. Let $\pi^c_{d_i}$ denote the set of parents of d_i in $\mathcal{N}^c$. We know that $\pi^c_{d_i} = \{x_i\}$. Since

$$\Omega_{\pi_{d_i}} = \Omega_{\pi^c_{d_i}}, \tag{9.85}$$

we will sometimes abuse symbols to use π_{d_i}, π_{d_i}, and x_i interchangeably.

Because of equation (9.85), a policy for $\mathcal{N}$ is also a policy for $\mathcal{N}^c$, and vice versa. In other words, $\mathcal{N}$ and $\mathcal{N}^c$ have the same policy space.

So, what remains to be proved is that for any policy δ

$$E_\delta[\mathcal{N}] = E_\delta[\mathcal{N}^c]. \tag{9.86}$$

Let P_δ and P^c_δ denote the joint probability induced by δ over the set of random and decision nodes of $\mathcal{N}$ and $\mathcal{N}^c$ respectively. This following lemma will be proved shortly.

Lemma 9.3 *For any decision node d_i,*

$$P_\delta(\pi_{d_i}) = P^c_\delta(\pi^c_{d_i}) \text{ and } P_\delta(\pi_{d_i}, d_i) = P^c_\delta(\pi^c_{d_i}, d_i).$$

Suppose $d_1, \ldots, d_k$ are all the decision nodes in $\mathcal{N}$. For each i, let $v_{i1}, \ldots, v_{im_i}$ be all the decision nodes in the section $\mathcal{N}(d_i, d_{i+1})$. We have

$$\begin{aligned} E_\delta[v_{ij}] &= \sum_{\pi_{v_{ij}}} P_\delta(\pi_{v_{ij}}) f_{v_{ij}}(\pi_{v_{ij}}) && \text{(By definition)} \\ &= \sum_{\pi_{v_{ij}}} \sum_{\pi_{d_i}, d_i} P_\delta(\pi_{d_i}, d_i) P_\delta(\pi_{v_{ij}} | \pi_{d_i}, d_i) f_{v_{ij}}(\pi_{v_{ij}}) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{\pi_{v_{ij}}} \sum_{\pi_{d_i}, d_i} P^c_\delta(\pi_{d_i}, d_i) P_{\mathcal{N}(d_i, d_{i+1})}(\pi_{v_{ij}} | \pi_{d_i}, d_i) f_{v_{ij}}(\pi_{v_{ij}}) \quad \text{(By Lemmas 9.3 and 9.1)} \\
&= \sum_{\pi_{d_i}, d_i} P^c_\delta(\pi_{d_i}, d_i) \sum_{\pi_{v_{ij}}} P_{\mathcal{N}(d_i, d_{i+1})}(\pi_{v_{ij}} | \pi_{d_i}, d_i) f_{v_{ij}}(\pi_{v_{ij}}) \qquad (9.87)
\end{aligned}
$$

Consequently, we have

$$
\begin{aligned}
E_\delta[\mathcal{N}] &= \sum_{i=1}^{k} \sum_{j=1}^{m_i} E_\delta[v_{ij}] \\
&= \sum_{i=1}^{k} \sum_{j=1}^{m_i} \sum_{\pi_{d_i}, d_i} P^c_\delta(\pi_{d_i}, d_i) \sum_{\pi_{v_{ij}}} P_{\mathcal{N}(d_i, d_{i+1})}(\pi_{v_{ij}} | \pi_{d_i}, d_i) f_{v_{ij}}(\pi_{v_{ij}}) \quad \text{(By equation (9.87))} \\
&= \sum_{i=1}^{k} \sum_{\pi_{d_i}, d_i} P^c_\delta(\pi_{d_i}, d_i) \sum_{j=1}^{m_i} \sum_{\pi_{v_{ij}}} P_{\mathcal{N}(d_i, d_{i+1})}(\pi_{v_{ij}} | \pi_{d_i}, d_i) f_{v_{ij}}(\pi_{v_{ij}}) \\
&= \sum_{i=1}^{k} \sum_{\pi_{d_i}, d_i} P^c_\delta(\pi_{d_i}, d_i) f_i(\pi_{d_i}, d_i) \quad \text{(By equations (9.83) and (9.84))} \\
&= \sum_{i=1}^{k} \sum_{x_i, d_i} P^c_\delta(x_i, d_i) f_i(x_i, d_i) \quad \text{(x_i and π_{d_i} are interchangeable.)} \\
&= E_\delta[\mathcal{N}^c]. \quad \text{(By definition)}
\end{aligned}
$$

The theorem is therefore proved. □

Proof of Lemma 9.3:

We now prove this lemma by induction on i. By Lemma 9.1, this lemma is true for the case when i=1. Suppose the lemma is true for the case of i. Consider the case of $i+1$:

$$
\begin{aligned}
P_\delta(\pi_{d_{i+1}}) &= \sum_{\pi_{d_i} \cup \{d_i\} - \pi_{d_{i+1}}} P_\delta(\pi_{d_i}, d_i) P_\delta(\pi_{d_{i+1}} | \pi_{d_i}, d_i) \\
&= \sum_{\pi_{d_i} \cup \{d_i\} - \pi_{d_{i+1}}} P^c_\delta(\pi_{d_i}, d_i) P_{\mathcal{N}(d_i, d_{i+1})}(\pi_{d_{i+1}} | \pi_{d_i}, d_i) \quad \text{(Lemma 9.1)} \\
&= \sum_{\pi_{d_i} \cup \{d_i\} - \pi_{d_{i+1}}} P^c_\delta(\pi_{d_i}, d_i) P^c_\delta(\pi_{d_{i+1}} | \pi_{d_i}, d_i) \\
&= P^c_\delta(\pi^c_{d_{i+1}}).
\end{aligned}
$$

Moreover, since $P_\delta(d_i|\pi_{d_i})$ and $P^c_\delta(d_i|\pi_{d_i})$ are completely determined by δ, they are equal. Consequently, we have

$$\begin{aligned} P_\delta(\pi_{d_{i+1}}, d_{i+1}) &= P_\delta(\pi_{d_{i+1}})P_\delta(d_{i+1}|\pi_{d_{i+1}}) \\ &= P^c_\delta(\pi_{d_{i+1}})P^c_\delta(d_{i+1}|\pi_{d_{i+1}}) \\ &= P^c_\delta(\pi_{d_{i+1}}, d_{i+1}). \end{aligned}$$

The lemma is therefore proved. □

9.4 Condensing SDDN's in general

This section extends the concept of condensation to cover all SDDN's. Let us first consider regular SDDN's in general. Irregular SDDN will be dealt with in Subsection 9.4.3.

9.4.1 Sections

Let $\mathcal{N}$ be a regular SDDN, smooth or non-smooth. Let d_1, d_2, $\ldots$, d_k be all the decision nodes in $\mathcal{N}$. For any $i \in \{1, 2, \ldots, k-1\}$, the *section of* $\mathcal{N}$ *from* π_{d_i} *to* $\pi_{d_{i+1}}$, denoted by $\mathcal{N}(d_i, d_{i+1})$, is the subnetwork of $\mathcal{N}$ that consists of the following nodes:

1. the nodes in $\pi_{d_i} \cup \pi_{d_{i+1}}$,
2. the nodes that are in the downstream set $Y_{II}(d_i, \mathcal{N})$ of π_{d_i} and in the upstream set $Y_I(d_{i+1}, \mathcal{N})$ of $\pi_{d_{i+1}}$, and
3. the disturbance nodes of d_{i+1}, (which are in the downstream set $Y_{II}(d_{i+1}, \mathcal{N})$ of $\pi_{d_{i+1}}$).

The graphical connections among the nodes in the section $\mathcal{N}(d_i, d_{i+1})$, remain the same as in $\mathcal{N}$, except that all the arcs pointing to nodes in $\pi_{d_i} \cup \{d_i\}$ that are not disturbance recipients of d_i are removed.

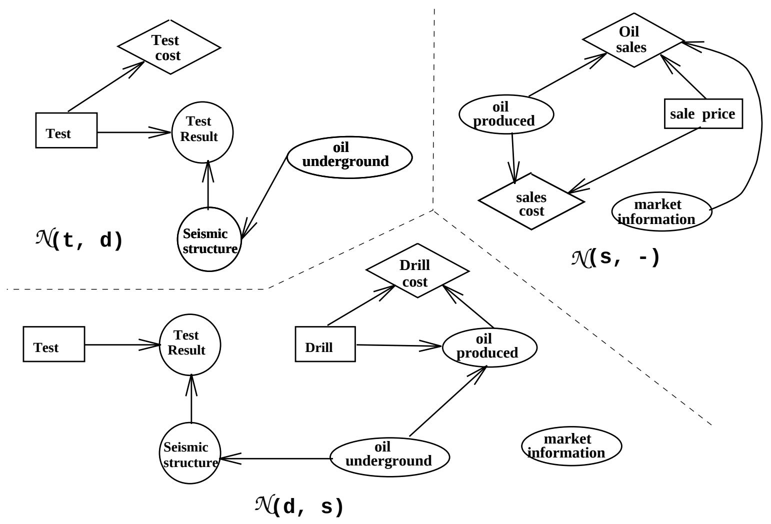

Figure 9.30: Sections in a non-smooth regular SDDN.

The nodes outside $\pi_{d_i}\cup\{d_i\}$ are either random nodes or value nodes. The value functions of all the value nodes remain the same as in $\mathcal{N}$.

The conditional probabilities of the random nodes outside $\pi_{d_i}\cup\{d_i\}$ remain the same as in $\mathcal{N}$. If a random node $c\in\pi_{d_i}\cup\{d_i\}$ is a disturbance recipients of d_i, then its conditional probability is the same as in $\mathcal{N}$. The decision nodes in $\pi_{d_i}\cup\{d_i\}$ and the random nodes in $\pi_{d_i}\cup\{d_i\}$ that are not disturbance recipient of d_i are viewed as root random nodes without prior probabilities. Compare this definition with the definition of tail for the non-smooth case (Section 6.2).

The *initial section* $\mathcal{N}(-, d_1)$ consists of the nodes in π_{d_1}, the nodes in the upstream set of π_{d_1}, and the disturbance nodes of d_1 (which are in the downstream set of π_{d_1}). This section consists of only random and value nodes, whose conditional probabilities or value functions of those nodes remain the same as in $\mathcal{N}$.

The *terminal section* $\mathcal{N}(d_k, -)$ is defined in the same way as in the smooth case.

Figure 9.30 shows the sections of the non-smooth regular SDDN in Figure 1.7. Unlike the case of smooth regular SDDN's (Figure 9.27), the sections $\mathcal{N}(\mathtt{t},\mathtt{d})$ and $\mathcal{N}(\mathtt{d},\mathtt{s})$ not only share the node in $\pi_{\mathtt{drill}}$, but also two other nodes — `oil-underground` and `seismic-structure`.

The reader is encouraged to verify that if a regular SDDN is smooth, then its sections as defined here are the same as those defined in Section 9.1. To do so, one only need to keeps in mind that in a smooth SDDN, there are no disturbance recipients.

9.4.2 Condensation

As in the smooth case, in each section $\mathcal{N}(\pi_{d_i},\pi_{d_{i+1}})$ we can compute the conditional probability $P_{\mathcal{N}(\pi_{d_i},\pi_{d_{i+1}})}(\pi_{d_{i+1}}|\pi_{d_i},d_i)$. We can also compute $P_{\mathcal{N}(\pi_{d_i},\pi_{d_{i+1}})}(\pi_{v_{ij}}|\pi_{d_i},d_i)$ for each value node v_{ij} in $\mathcal{N}(\pi_{d_i},\pi_{d_{i+1}})$, and hence the local value function f_i.

The*condensation* of $\mathcal{N}$ is defined from the $P_{\mathcal{N}(\pi_{d_i},\pi_{d_{i+1}})}(\pi_{d_{i+1}}|\pi_{d_i},d_i)$'s and the f_i's in the same way as in Section 9.2.

Theorem 9.2 *A regular SDDN, smooth or non-smooth, is equivalent to its condensation.*

Proof: Let $\mathcal{N}$ be a regular SDDN, let $\mathcal{N}^*$ be the output network of SMOOTHING($\mathcal{N}$). Then $\mathcal{N}^*$ is smooth and equivalent to $\mathcal{N}$. According to Proposition 9.1, which comes up next, the condensation of $\mathcal{N}$ is the same as the condensation of $\mathcal{N}^*$. Thus $\mathcal{N}^*$ is equivalent to the condensation $\mathcal{N}^c$ of $\mathcal{N}$ or equivalently of $\mathcal{N}^*$. Consequently, $\mathcal{N}$ is equivalent to $\mathcal{N}^c$. □

Proposition 9.1 *Let $\mathcal{N}$ be a non-smooth regular SDDN, and let $\mathcal{N}^*$ be the output network of SMOOTHING($\mathcal{N}$). Let d_i and d_{i+1} be two consecutive decision nodes. Then*

$$P_{\mathcal{N}^*(d_i,d_{i+1})}(\pi_{d_{i+1}}|\pi_{d_i},d_i) = P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{d_{i+1}}|\pi_{d_i},d_i), \tag{9.88}$$

and

$$P_{\mathcal{N}^*(-,d_1)}(\pi_{d_1}) = P_{\mathcal{N}(-,d_1)}(\pi_{d_1}). \tag{9.89}$$

Furthermore, for any value node v_{ij} in $\mathcal{N}(d_i, d_{i+1})$,

$$P_{\mathcal{N}^*(d_i,d_{i+1})}(\pi_{v_{ij}}|\pi_{d_i}, d_i) = P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{v_{ij}}|\pi_{d_i}, d_i). \tag{9.90}$$

Proof: Given any policy δ of $\mathcal{N}$, let P_δ be the joint probability δ induces over all the random and decision nodes of $\mathcal{N}$. Then δ is also a policy of $\mathcal{N}^*$. Let P^*_δ be the joint probability δ induces over all the random and decision nodes of $\mathcal{N}^*$. Since arc reversals do not change the joint probability, we have

$$P^*_\delta = P_\delta. \tag{9.91}$$

In particular, we have

$$P^*_\delta(\pi_{d_1}) = P_\delta(\pi_{d_1}),$$

$$P^*_\delta(\pi_{d_{i+1}}|\pi_{d_i}, d_i) = P_\delta(\pi_{d_{i+1}}|\pi_{d_i}, d_i),$$

$$P^*_\delta(\pi_{v_{ij}}|\pi_{d_i}, d_i) = P_\delta(\pi_{v_{ij}}|\pi_{d_i}, d_i).$$

Consequently, we have

$$P_{\mathcal{N}^*(-,d_1)}(\pi_{d_1}) = P^*_\delta(\pi_{d_1}) = P_\delta(\pi_{d_1}) = P_{\mathcal{N}(-,d_1)}(\pi_{d_1}),$$

$$\begin{aligned} P_{\mathcal{N}^*(d_i,d_{i+1})}(\pi_{d_{i+1}}|\pi_{d_i}, d_i) &= P^*_\delta(\pi_{d_{i+1}}|\pi_{d_i}, d_i) \\ &= P_\delta(\pi_{d_{i+1}}|\pi_{d_i}, d_i) \\ &= P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{d_{i+1}}|\pi_{d_i}, d_i), \end{aligned}$$

and

$$\begin{aligned} P_{\mathcal{N}^*(d_i,d_{i+1})}(\pi_{v_{ij}}|\pi_{d_i}, d_i) &= P^*_\delta(\pi_{v_{ij}}|\pi_{d_i}, d_i) \\ &= P_\delta(\pi_{v_{ij}}|\pi_{d_i}, d_i) \\ &= P_{\mathcal{N}(d_i,d_{i+1})}(\pi_{v_{ij}}|\pi_{d_i}, d_i). \end{aligned}$$

The proposition is therefore proved. □

9.4.3 Irregular SDDN's

The generalization of concept of condensation to irregular SDDN's is straightforward. The only issue that demands special attention is that there may be more than one decision node that directly succeeds a given decision node. Thus, one will not be able to speak of the section from d_i to d_{i+1}; one can only speak of the section starting at d_i.

The condensation of an irregular SDDN is not a (linear) MDP since the decision nodes are not totally ordered. It is an MDP over a rooted tree.

9.5 Asymmetries and unreachable states

This section investigates how to make use of the asymmetric nature of decision problems to cut down the number of states for nodes in condensations. The basic idea is due to Qi (1993).

9.5.1 Asymmetries in decision problems

In the oil wildcatter examples, if the `test` is not performed, there is no `test-result`. The meaningfulness of the variable `test-result` depends on the decision made about `test`. This phenomenon is called *asymmetry* .

Asymmetries are readily represented in decision trees. In Figure 1.2, we see that the decision to `test` leads the decision maker to the upper branch of the tree, where s/he will observe the `test-result` upon which to make the `drill` decision. One the other hand, the decision not to `test` leads the decision maker to the lower branch of the tree, where there is no `test-result`. S/he has to make the `drill` decision without knowing anything about the `seismic-structure`. The lower branch is much smaller than the upper branch because of asymmetries.

Influence diagrams (and decision networks) are appreciated for compactness, intuitiveness and the ability to reveal independencies. However they cannot represent asymmetries. Artificial states have to be introduced to render the decision problems "symmetric". The oil wildcatter problems are made "symmetric" by introducing the artificial value `no-result` for the variable `test-result`.

Artificial states may bring about unnecessary computations. In the oil wildcatter example, both `test` and `test-result` are parents of `drill`. Thus, a decision for `drill` has to be computed for all possible combinations of values of `test` and `test-result`, even though some of those combinations, for instance `test`=`no` and `test-result`=`open-structure`, are impossible. Such unnecessary computations lead some to doubt the efficiency of influence diagrams (Lawrence 1990, Shafer 1993).

Fung and Shachter (1990), Covaliu and Oliver (1992), Smith *et al* (1993) , and Shenoy (1993) deal with the asymmetric nature of decision problems by generalize influence diagrams to explicitly represent asymmetries. This section shows that in SDDN's, asymmetries can uncovered even when they are not explicitly represented. We still need to introduce artificial states; but we are able to eliminate the unnecessary computations brought about by those artificial states.

9.5.2 Eliminating unreachable states in condensations

In the condensation shown in Figure 9.29, x_2 is a compound variable consisting of `test` and `test-result`, which are respectively shorthanded as `t` and `tr`. The variable `t` has two possible values — `yes` or `no`, while `tr` has four — `no-structure` (`ns`), `open-structure` (`os`), `closed-structure` (`cs`) and `no-result` (`nr`). Thus, x_2 has eight possible values.

Since x_2 is a variable in a condensation, which is a MDP, we shall refer to its possible values as states.

Due to the asymmetric nature of the oil wildcatter problem, four of the eight states of x_2, namely (`t`=`no`, `tr`=`ns`), (`t`=`no`, `tr`=`os`), (`t`=`no`, `tr`=`cs`), and (`t`=`yes`, `tr`=`nr`), are unreachable, in the sense that their probabilities are zero. By pruning those states of x_2, we avoid the unnecessary computation due to the introduction of the artificial state `no-result`.

We now define the concept of unreachable state more rigorously. Suppose $\mathcal{N}$ is a regular SDDN and $\mathcal{N}^c$ is its condensation. Let d_i be a decision node and let x_i be its unique parent in $\mathcal{N}^c$. As before, we shall use x_i, π_{d_i}, and $\pi^c_{d_i}$ interchangeably.

Any policy δ of $\mathcal{N}^c$ induces a joint probability P^c_δ over all the random and decision nodes of $\mathcal{N}^c$. A state β of x_i is *unreachable under* δ if $P^c_\delta(x_i{=}\beta) = 0$. A state of x_i is *unreachable* if it is unreachable under all the policies of $\mathcal{N}^c$.

One interesting question is: How can one identify unreachable states?

For any state β of x_1, $P^c_\delta(x_1{=}\beta)$ is independent of δ and equals $P_{\mathcal{N}(-,d_1)}(\pi_1{=}\beta)$. So, it is unreachable if and only if

$$P_{\mathcal{N}(-,d_1)}(\pi_1{=}\beta) = 0.$$

When $i > 1$, the state $\{x_i{=}\beta\}$ is unreachable if and only if the following is true: for any policy δ and for any $\alpha{\in}\Omega_{d_{i-1}}$ and any $\gamma{\in}\Omega_{x_{i-1}}$, either

1. $P^c_\delta(x_{i-1}{=}\gamma) = 0$, or
2. $P^c_\delta(x_i{=}\beta|x_{i-1}{=}\gamma, d_{i-1}{=}\alpha) = 0$.

By Lemma 9.1, we have:

$$P^c_\delta(x_i{=}\beta|x_{i-1}{=}\gamma, d_{i-1}{=}\alpha) = P_{\mathcal{N}(d_{i-1},d_i)}(\pi_{d_i}{=}\beta|\pi_{d_{i-1}}{=}\gamma, d_{i-1}{=}\alpha).$$

Therefore, we have

Theorem 9.3 *Suppose $\mathcal{N}$ is a regular SDDN.*

1. *A state $\{x_1 = \beta\}$ is unreachable if and only if*

$$P_{\mathcal{N}(-,d_1)}(\pi_{d_1}{=}\beta) = 0.$$

2. *When $i > 1$, the state $\{x_i{=}\beta\}$ is unreachable if and only if for any $\alpha \in \Omega_{d_{i-1}}$ and any $\gamma \in \Omega_{\pi_{d_{i-1}}}$, either*

   (a) $\{\pi_{d_{i-1}}{=}\gamma\}$ is unreachable, or

   (b) $P_{\mathcal{N}(d_{i-1},d_i)}(\pi_{d_i}{=}\beta|\pi_{d_{i-1}}{=}\gamma, d_{i-1}{=}\alpha) = 0.$

This theorem suggests an obvious top-down procedure of constructing the condensation of regular SDDN's that eliminates unreachable states along the way. One begins by computing $P_{\mathcal{N}(-,d_1)}(\pi_{d_1}{=}\beta)$ for each state β of x_1, and deletes those states with zero probability. One then computes $P_{\mathcal{N}(d_1,d_2)}(\pi_{d_2}{=}\beta|\pi_{d_1}{=}\gamma, d_1{=}\alpha)$ for each state β of x_2 given each reachable state γ of x_1 and each value α of d_1, and deletes all the states β whose conditional probabilities are always zero. And so on and so forth.

So far in this section, our exposition has been in terms only regular SDDN's. However, the idea can be extended to irregular SDDN's in a straightforward way.

This approach constructs condensations that do not contain any unreachable states. However, it also excludes the possibility of the parallel computations we mentioned in Subsection 9.2.1.

9.6 A two-stage algorithm for evaluating SDDN's

The concept of condensation leads to the following two-stage approach for evaluating regular SDDN's.

Procedure EVALUATE2($\mathcal{N}$)

- Input: $\mathcal{N}$ — an SDDN.
- Output: The optimal policy and the optimal expected value of $\mathcal{N}$.

1. Compute the condensation $\mathcal{N}^c$ of $\mathcal{N}$.
2. Evaluate $\mathcal{N}^c$ by the folding back strategy.

The procedure EVALUATE2 can be viewed as a modification of EVALUATE1, where all the Bayesian network inferences (BNI's) are grouped into one stage, i.e the stage of condensation. EVALUATE2 is better than EVALUATE1 in terms of modularity. As a matter of fact, EVALUATE2 can be implemented in three modules: one for managing sections of SDDN's, one for doing BNI's, and one for evaluating MDP's. MDP's have been studied in Dynamic programming for a long time, BNI's have also been under extensive investigation during the last decade. Good algorithms exist for both MDP's and BNI's. This leaves us with only the module for managing sections of SDDN's.

Besides modularity, a more important of advantage of EVALUATE2 over EVALUATE1 is, as pointed out in Section 9.5, that EVALUATE2 enables us to exploit the asymmetric nature of decision problems. Also EVALUATE2 facilitates parallel processing (Section 9.2.1). As we pointed out Section 9.5, one can only have one of those two advantages; they do not co-exist.

Zhang *et al* (1993b) presents yet another advantage of EVALUATE2, namely EVALUATE2 also leads an incremental method of computing the value of perfect information.

9.6.1 Comparison with other approaches

Two two-stage algorithms for evaluating influence diagrams have been proposed by Howard and Matheson (1984) and Cooper (1989).

To evaluate an influence diagram, Howard and Matheson (1984) first transforms the diagram into a decision tree, and then evaluate the decision tree. Our approach transform an SDDN into an MDP, instead of a decision tree. In Howard and Matheson's transformation, every random node in the influence diagram corresponds to one level in the decision tree, while in our approach all the random nodes, except for those that are observed at some point, are summed out in the condensation process. Also, condensing an SDDN into an MDP does not lose independencies for decision nodes, while transforming an SDDN into a decision tree would.

To understand the approach by Cooper (1989), consider an influence diagram $\mathcal{N}$. Let v be the only one value node and let the decision nodes be d_1, d_2, ..., d_k. Given a policy δ, let P_δ be the joint probability δ induces over all the random and decision nodes. It is implicit in Cooper (1989) that $P_\delta(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$ does not depend on δ; and hence can be written as $P(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$.

Recursively define a series of functions g_i ($1 \leq i \leq k$) by

$$g_k(\pi_{d_k}) =_{def} max_{d_k} \sum_{\pi_{d_k}-\pi_v} f_v(\pi_v) P(\pi_v|\pi_{d_k}, d_k), \tag{9.92}$$

f_v is the value function for v; and for any $i<k$

$$g_i(\pi_{d_i}) =_{def} max_{d_i} \sum_{\pi_{d_{i+1}}-\pi_{d_i}} g_{i+1}(\pi_{d_{i+1}}) P(\pi_{d_{i+1}}|\pi_{d_i}, d_i). \tag{9.93}$$

The following proposition is given by Cooper (1989).

Proposition 9.2 *For any $i \in \{1, 2, \ldots, k\}$, the optimal policy δ^o for d_i is given by*

$$\delta^o(\pi_{d_i}) = argmax_{d_i} \sum_{\pi_{d_{i+1}}-\pi_{d_i}} g_{i+1}(\pi_{d_{i+1}}) P(\pi_{d_{i+1}}|\pi_{d_i}, d_i).$$

The optimal expected value $E[\mathcal{N}]$ is given by

$$E[\mathcal{N}] = \sum_{\pi_{d_1}} g_1(\pi_{d_1}).$$

Proof: Consider the condensation $\mathcal{N}^c$ of $\mathcal{N}$. The probability $P(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$ is the same as $P^c(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$. The function g_k is the value function of v_k^c, and the value functions of all the other value node in $\mathcal{N}^c$ are zero. The proposition thus follows from the fact that $\mathcal{N}^c$ is an MDP. □

To make a comparison, it has been pointed out in this thesis that the conditional probability $P(\pi_{d_{i+1}}|\pi_{d_i}, d_i)$ can be computed locally in the section $\mathcal{N}(d_i, d_{i+1})$. More importantly, we have generalized Proposition 9.2 from influence diagrams to SDDN's (Theorems 9.1 and 9.2).

Chapter 10

Conclusion

10.1 Summary

This thesis has been about how to use Bayesian decision theory to solve decision problems that involve multiple decisions and multiple variables. Here is a summary of our contributions.

First of all, the concept of a decision network has been developed from a general way for stating decision problems and a standard Bayesian decision theory setup. A decision network is a representation of a decision problem together with knowledge (belief) and utilities necessary for its solution. We have argued that decision networks are the most general representation framework for the purpose of solving what we call multiple-decision problems in Bayesian decision theory. In particular, decision networks are more general than influence diagrams.

The evaluation of a decision network requires, in general, an exhaustive search through the whole policy space. A concept of decomposability has been introduced for decision networks and it has been shown that this decomposability leads to a divide and conquer strategy.

From a computational point of view, it is most desirable if a decision network is stepwise-solvable, i.e if it can be evaluated by considering one decision node at a time. We have introduced stepwise-decomposable decision networks (SDDN's) and have shown they can be evaluated not only by considering one decision node at a time, but also by considering only one portion, usually a small portion, of the network at time.

We have also shown that stepwise-decomposability is the weakest graphical criterion that guarantees stepwise-solvability.

The problem of evaluating SDDN's has been studied in detail. A number of important concepts, such as simple semi-decision networks, body, tail, and smoothness have been identified. A procedure named EVALUATE1 has been proposed. The advantages of this procedure include the adoption of the divide and conquer strategy, a clear separation of Bayesian network inferences, and minimal numerical divisions.

We have introduced the concept of decision nodes being conditionally independent of part of available information, and have shown its equivalence to the concept of removable arcs. An algorithm has been designed that is able to find and prune *all* the removable arcs in an SDDN that can be identified graphically without resorting to numerical computations.

Finally, we have investigated the relationship between SDDN's and Markov decision processes. Finite stage Markov decision processes are special SDDN's, and SDDN's can be condensed into Markov decision processes. This relationship leads to a two-stage approach for evaluating SDDN's. Besides providing an even cleaner interface between decision network evaluation and Bayesian network inferences than EVALUATE1, this approach is able to make use of the asymmetric nature of decision problems, facilitates parallel computation, and gives rise to an incremental way of computing the value of perfect information.

10.2 Future work

10.2.1 Application to planning and control under uncertainty

High on our list of future research is to apply the theory of decision networks to the area of planning and control under uncertainty, and thereby further develop the theory.

The history of influence diagrams is short (Matheson and Howard 1984, Miller *et al* 1976); and using influence diagrams to represent dynamic systems for the purpose of planning and control is a development over the last three or four years (Dean and Wellman 1991). Many issues remains to be addressed.

To get a feeling about some of the issues, let us consider the mobile target localization problem taken from Dean *et al* (1990) with some minor changes. As shown in Figure 10.31, there is a mobile target, and there is a robot that tracks the target. The target location as observed by the robot may be different from the actual target location. At each time point, the robot makes a decision as to what location to report based on the observed target location. There is a payoff depending on how accurate the report is; the overall payoff is the sum of the payoffs of all the time points. The robot also makes a tracking decision according to its own location and the observed target location. The location of the robot at the next time point depends on its location and the tracking decision made at the current time point.

In this example, the decisions at time point t2, for instance, depend on the observed target location at time point t1, because it helps the robot to better estimate the actual target location at time point t2. The decisions at time point t20 depend on the observed target locations at all the previous time points, as indicated by the dotted arcs. So, the decision nodes at time point t20 have 21 parents, and their decision table have n^{21} entries, where n stands for the number of possible locations. A decision table of such a size can neither be computed nor be stored.

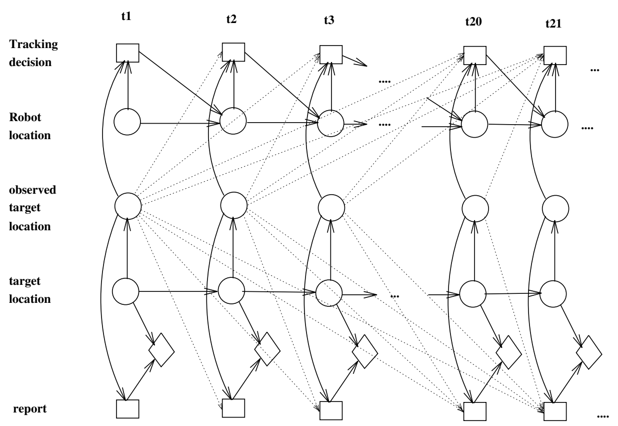

Figure 10.31: Mobile target localization.

As we mentioned in the introduction, a reasonable thing to do here is to assume decision at any time point depends on only, for instance, two previous time points, while being independent of all the other time points. This kind of independence assumptions for decision node can be make in decision networks because they are able to represent independencies for decision nodes. One important issue in applying the theory of decision network to planning and control is how can one guarantee that those independence assumptions yield decisions with acceptable bounds from the optimal?

When evaluating decision alternatives at a certain time point, one looks into the future to see what states of affairs each of the alternative may result in and with what certainties. Another issue in applying the theory of decision network to planning and control is how many time points should one looks into the future? How should one discount the importance of future time points that are far from the present?

The above two issues are about reducing the complexities of planning and control problems in the time dimension. Another dimension in which one can explore the opportunities for reducing complexities is the dimension of problem structure. The graph-theoretical language of decision networks allows us to capture, at the level of relation, the dependence and independence relationships among nodes. A step further is to explore the so-called inter-causal indepedencies (e.g. Heckerman 1993), which basically says that several caused independently contribute to an effect. Inter-causal independence is at the numerical level.

Let k be the number of parents of a random node c. Suppose all variables are binary. Then, the conditional probability of c requires $2^{k+1}-1$ numbers to specify. Assessing those numbers and using them in inferences is hard when k is large. However, if the conditional probability of c satisfies the so-called noisy OR-gate model — an inter-causal independence model — then we need only k numbers to specify the conditional probability (Pearl 1988). There is a growing research interest in making use of inter-causal models to reduce the complexities of knowledge acquisition and inference in Bayesian networks and Influence diagrams (e.g. Heckerman 1993). Much work remain to be done in this direction.

One specific idea we have is to treat the noisy OR-gate model in Dempster-Shafer theory. Dempster-Shafer theory is a theory about combining evidence from independence sources, and is thus closely related to the concept of inter-causal independence.

10.2.2 Implementation and experimentation

Another thing to do in the future is implementation. The procedure EVALUATE1 can probably be implemented on top of IDEAL, a software package for analysis of influence diagrams developed by Srinivas and Breese (1990). For the sake of TEST-STEPWISE-DECOMPOSABILITY and PRUNE-REMOVABLES, it is a good idea to keep the skeleton of a decision network separate from its numerical components, namely its conditional probabilities and value functions. The implementation of EVALUATE2 may be more involved if the functionalities of parallel processing, exploitation of asymmetries and incremental computation of value of information are all to be materialized.

In the thesis, we have argued the that EVALUATE1 and EVALUATE2 are advantageous over all the previous algorithms for evaluating influence diagrams. Experiments need to be performed to substantiate this claim.

10.2.3 Extension to the continuous case

This thesis has only dealt with discrete variables. It would be interesting to extend our theory so that it can also handle continuous variables. To achieve this extension, the following three issues need to be addressed.

The first issue is the existence of an optimal policy. This is not an issue in the discrete case. Since there are only finitely many possible policies, one of them must be optimal. In the continuous case, however, there may be infinitely many possible policies; the existence of an optimal policy is not obvious.

The second issue is integration. In the discrete case, we sum out variables. In the continuous case, we need to integrate out variables. While summation is provided in most programming languages, integration is not.

The third issue is the operation of finding the maxima of a function. This can be done by exhaustive enumeration in the discrete case. In the continuous case, more advanced techniques need to be used.

10.2.4 Multiple agents

We have said that decision network is able to represent multiple cooperative agents with a common goal. We have also given two examples in this regard. However, much foundational work remains to be done to ensure that the optimal decision policies as defined in this thesis is indeed optimal in particular applications.

One can also consider game theory situations where agents are adversaries of each other. One may come out with some kind of game network based on game trees in a way similar to how decision networks grew out of decision trees.

Bibliography

[1] M. Baker and T. E. Boult (1990), Pruning Bayesian networks for efficient computation, in *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, July, Cambridge, Mass. , pp. 257 - 264.

[2] K. Basye, T. Dean, and J. S. Vitter (1989), Coping with uncertainty in map learning, in *Proceedings IJCAI 11*, pp. 663-668.

[3] D. E. Bell, H. Raiffa, and A. Tversky (1988), *Decision Making: Descriptive, Normative, and prescriptive interactions*, Cambriage University Press.

[4] U. Bertelè and F. Brioschi (1972), *Nonserial dynamic programming*, Mathematics in Science and Engineering, Vol. 91, Academic Press.

[5] D. P. Bertsekas (1976), *Dynamic Programming and Stochastic Control*, Mathematics in Science and Engineering, Vol. 125, Academic Press.

[6] J. Breese (1991). Construction of belief and decision networks, Technical Report, Rockwell International Science Center.

[7] K. Chang and R. Fung (1990). Refinement and Coarsening of Bayesian networks, in *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, July, Cambridge, Mass. , pp. 475-482.

[8] J. R. Clarke and G. M. Provan (1992), A dynamic decision-theoretic approach to the management of GVHD, in *Proc. AAAI Spring Symposium on AI and Medicine*.

[9] G. F. Cooper (1989), A method for using belief networks as influence diagrams, in *Proceedings of the fifth Conference on Uncertainty in Artificial Intelligence*, pp. 55-63.

[10] G. F. Cooper (1990) The computational complexity of probabilistic inference using Bayesian belief networks, *Artificial Intelligence*, **42**, pp. 393-405.

[11] G. F. Cooper (1990), Bayesian belief-network inference using recursive decomposition, Report No. KSL 90-05, Knowledge Systems Laboratory, Medical Computer Science, Stanford University.

[12] Z. Covaliu and R. M. Oliver (1992), Formulation and solution of decision problems using decision diagrams, Working paper, University of California at Berkeley.

[13] T. L. Dean and K. Kanazawa (1989), A model for reasoning about persistence and causation, *Computational Intelligence*, **5**(3), pp. 142-150.

[14] T. L. Dean and M. P. Wellman (1991), *Planning and Control*, Morgan Kaufmann.

[15] E. V. Denardo (1982), *Dynamic Programming: Models and Applications* Prentice-Hall.

[16] P. Dagum and R. M. Chavez (1993), Approximating probabilistic inference in Bayesian belief networks, *Pattern Analysis and Machinr Intelligence*, Vol. 15, 3, pp. 246-255.

[17] J. W. Egar and M. A. Musen (1993), Graph-grammar assistance for Automated generation of influence diagrams, in *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pp. 235-243.

[18] K. J. Ezawa (1986), Efficient evaluation of influence diagrams, Ph.D. Thesis, Dept. of Engineering-Economics Systems, Stanford University.

[19] K. J. Ezawa (1992) Technology planning for advanced telecommunication services: a computer-aided approach, *Telematics and Informatics*, **19**, No. 2.

[20] P. C. Fishburn (1988), Normative theories of decision making under risk and under uncertainty, in Bell *et al* 1988.

[21] K. W. Fertig and J. Breese (1993), Probability intervals over influence diagrams, *Pattern Analysis and Machinr Intelligence*, Vol. 15, 3, pp. 280-287.

[22] R. M. Fung and R. D. Shachter (1990), Contingent Influence Diagrams, Advanced Decision Systems, 1500 Plymouth St., Mountain View, CA 94043, USA.

[23] P. Gördenfors and N. Sahlin (1988a), *Decision, Probability, and Utility: Selected Readings*, Cambridge University Press.

[24] P. Gördenfors and N. Sahlin (1988b), Introduction: Bayesian decision theory – foundations and problems, in P. Gördenfors and N. Sahlin (1988)a.

[25] D. Geiger, T. Verma, and J. Pearl (1990), *d*-separation: From theorems to algorithms, in *Uncertainty in Artificial Intelligence* **5**, pp. 139-148.

[26] R. P. Goldman and E. Cherniak (1993), A language for construction belief networks, *Pattern Analysis and Machinr Intelligence*, Vol. 15, 3, pp. 196-208.

[27] M. Goldzsmith, P. Morris, and J. Pearl (1993), A maximum entropy approach to nonmonotonic reasoning, *Pattern Analysis and Machinr Intelligence*, Vol. 15, 3, pp. 220-232.

[28] I. Good (1961), A causal calculus (I). *British Journal of Philosophy of Science*, 11, pp. 305-318

[29] N. A. J. Hastings and J. M. C. Mello (1977), *Decision networks*, John Wiley & Sons.

[30] D. Heckerman (1993), Caual indepedence for knowledge acquisition and inference, in *Proceedings of the Nitth Conference on Uncertainty in Artificial Intelligence*, pp. 122-127.

[31] D. Heckerman and E. Horvitz (1990), Problem formulation as the reduction of a decision model, in *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, July, Cambridge, Mass. , pp. 82-89.

[32] M. Henrion (1987), Some practical issues in constructing belief networks, in L. Kanal, T. Levitt, and J. Lemmer (eds.) *Uncertainty in Artificial Intelligence*, 3, pp. 161-174, North-Holland.

[33] M. Horsch and D. Poole (1991), A dynamic approach to probabilistic inference using Bayesian networks, in *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, July, Cambridge, Mass. , pp. 155-161.

[34] J. Hosseini (1968), Decision analysis and its application in the choice between two wildcat oil ventures, *Interfaces*, 16, pp. 75-85.

[35] R. A. Howard, and J. E. Matheson (1984), Influence Diagrams, in *The principles and Applications of Decision Analysis*, Vol. II, R. A. Howard and J. E. Matheson (eds.). Strategic Decisions Group, Menlo Park, California, USA.

[36] R. A. Howard (1990), From influence to relevance to knowledge, in [50].

[37] F. V. Jensen, K. G. Olesen, and K. Anderson (1990), An algebra of Bayesian belief universes for knowledge-based systems, *Networks*, 20, pp. 637 - 659.

[38] Kanazawa (1991), A logic and time nets for probabilistic inference, In *Proceedings AAAI-91.*

[39] J. Kim and J. Pearl (1983), A computational model for causal and diagnostic reasoning in inference engines, in *Proceedings of the Eigth International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, pp. 190-193.

[40] U. Kjærulff (1990), Triangulation of Graphs - Algorithms giving small total state space, R 90-09, Institute for Electronic Systems, Department of Mathematics and Computer Science, Strandvejen, DK 9000 Aalborg, Denmark.

[41] P. Klein, A. Agrawal, A. Ravi, and S. Rao (1990), Approximation through multicommodity flow, in Proceedings of 31st Symposium on Foundations of Computer Science, pp. 726-737.

[42] S. L. Lauritzen and D. J. Spiegehalter (1988), Local computations with probabilities on graphical structures and their applications to expert systems, *Journal of Royal Statistical Society B*, 50: 2, pp. 157 - 224.

[43] S. L. Lauritzen, A. P. Dawid, B. N. Larsen, and H. G. Leimer (1990), Independence Properties of Directed Markov Fields, *Networks*, 20, pp. 491-506.

[44] T. S. Levitt, T. O. Binford, G. J. Ettinger, and P. Gelband (1988), Utility-based control for computer vision, *Proceedings of the Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 245-256.

[45] T. S. Levitt, J. M. Agosta, and T. O. Binford (1989), Model-Based influence diagrams for machine vision, *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 233-244.

[46] A. C. Miller, M. W. Merkhofer, R. A. Howard, J. E. Matheson, and T. R. Rice (1976), Development of automated computer aids for decision analysis, Technical Report 3309, SRI International, Menlo Park, Calif.

[47] J. E. Matheson (1990), Unsing influence diagrams to value information and control, in [50].

[48] W. W. North (1968), A tutorial introduction to decision theory, reprinted in [75].

[49] P. Ndilikilikesha (1991), Potential Influence Diagrams, Working Paper No. 235, Business School, University of Kansas.

[50] R. M. Oliver and J. Q. Smith eds. (1990), *Influence Diagrams, Belief Nets and Decision Analysis*, John Wiley and Sons.

[51] J. Pearl (1988), *Probabilistic Reasoning in Intelligence Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, Los Altos, CA.

[52] L. D. Phillips (1990), Discussion of From Influence to Relevance to Knowledge by R. A. Howard, in [50].

[53] D. Poole and E. Neufeld (1991), Sound probabilistic inference in Prolog: An executable specification of Bayesian networks, Department of Computer Science, University of British Columbia, Vancouver, B. C., V6T 1Z2, Canada.

[54] D. Poole (1992), Search for Computing posterior probabilities in Bayesian networks, Technical Report 92-24, Department of Computer Science, University of British Columbia, Vancouver, Canada.

[55] G. M. Provan (1991), Dynamic network updating techniques for diagnostic reasoning, in *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence.*

[56] G. M. Provan and D. Poole (1991), The utility of consistency-based diagnostic techniques, in *The Proc. of the 2ns Conference on the principles of Knowledge representation.*

[57] G. M. Provan and J. R. Clarke (1993), Dynamic network construction and updating techniques for the diagnosis of acute abdominal pain, *Pattern Analysis and Machine Intelligence* **15**, pp. 299-307.

[58] M. L. Puterman (1990), Markov decision processes, in D. P. Heyman and M. J. Sobel (eds.), *Handbooks in OR & MS.*, Vol. 2, pp. 331-434, Elsevier Science Publishers.

[59] R. Qi and D. Poole (1992a), Two algorithms for decision tree search, PRICAI'92, Seoul, Korea, pp. 121-127.

[60] R. Qi (1993), Decision graphs: algorithms and applications, Ph.D Thesis, Department of Computer Science, University of British Columbia, under preparation.

[61] H. Raiffa, (1968), *Decision Analysis*, Addison-Wesley, Reading, Mass.

[62] D. J. Rose (1970), Triangulated graphs and the elimination process, *Journal of Mathematical Analysis and Applications*, **32**, pp 597-609.

[63] L. J. Savage (1954), *The foundations of statistics*, Wiley, New York.

[64] R. O. Schlaifer (1967), *Analysis of decisions under uncertainty (preliminary edition)*, McGraw-Hill.

[65] R. Shachter (1986), Evaluating Influence Diagrams, *Operations Research*, 34, pp. 871-882.

[66] R. Shachter (1988), Probabilistic Inference and Influence Diagrams, *Operations Research*, 36, pp. 589-605.

[67] R. Shachter (1990), An ordered examination of influence diagrams, *Networks*, Vol. 20, 5, pp. 535-564.

[68] R. D Shachter, S. K. Andersen, and P. Szolovits (1992), The equivalence of exact methods for probabilistic inference in belief networks, Department of Engineering-Economic Systems, Stanford University.

[69] R. D. Shachter, B. D'Ambrosio, and B. A. Del Favero (1990), Symbolic Probabilistic Inference in Belief Networks, in *AAAI-90*, pp. 126-131.

[70] Shachter and Peot (1992), Decision making using probabilistic inference methods, in *Proc. of 8th Conference on Uncertainty in Artificial Intelligence*, July 17-19, Stanford University, pp. 276-283.

[71] G. Shafer (1988), Savage revisited, in Bell *et al* 1988.

[72] G. Shafer and P. Shenoy (1988), Local computation in hypertrees, Working Paper No. 201, Business School, University of Kansas.

[73] G. Shafer (1990), Decision making: introduction to Chapter 3 of [75].

[74] G. Shafer (1993), *Probabilistic Expert Systems*, to be published by SIAM.

[75] G. Shafer and J. Pearl (1990), *Reading in uncertainty reasoning*, Morgan Kaufmann Publishers, San Mateo, California.

[76] P. P. Shenoy, (1990), Valuation-Based Systems for Bayesian Decision Analysis, Working Paper No. 220, Business School, University of Kansas.

[77] P. P. Shenoy, (1992), Valuation-Based Systems for Bayesian Decision Analysis, *Operations research*, 40, No. 3, pp. 463-484.

[78] P. P. Shenoy, (1993), Valuation network representation and solution of asymmetric decision problems, work paper No. 246, Business School, University of Kansas.

[79] J. E. Smith, S. Holtzman, and J. E. Matheson (1993), Structuring conditional relationships in influence diagrams, *Operations Research*, 41, No. 2, pp. 280-297.

[80] J. Q. Smith (1988), *Decision Analysis: a Bayesian Approach*, Chapman and Hall.

[81] J. Q. Smith (1989), Influence Diagrams for Statistical Modeling, *Ann. Stat.*, 17, pp. 654-672.

[82] T. P. Speed (1991), Complexity, calibration and causality in influence diagrams, in [50].

[83] S. Srinivas and J. Breese (1990), IDEAL: A software package for analysis of influence diagrams, in *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, July, Cambridge, Mass. , pp. 212-219.

[84] J. A. Tatman and R. Shachter (1990), Dynamic programming and influence diagrams, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 20, pp. 265-279.

[85] J. von Neumann and O. Morgenstein (1944), *Theory of games and economic behaviours*, Princeton University Press. 2nd eed., 1947, 3rd ed., 1953.

[86] M. Wellman (1990), *Formulation of Tradeoffs in Planning under Uncertainty*, Pitman, London.

[87] M. Wellman, J. Breese, and R. Goldman (1992), From knowledge bases to decision models, *Knowledge Engineering Review*, **7**(1).

[88] L. Zhang (1993), Studies on hypergraphs (I): Hyperforests, *Discrete Applied Mathematics*, 42, pp. 95-112.

[89] L. Zhang and D. Poole (1992) Sidestepping the triangulation problem in Bayesian net computations, in *Proc. of 8th Conference on Uncertainty in Artificial Intelligence*, July 17-19, Stanford University, pp. 360-367.

[90] L. Zhang and D. Poole (1992), Stepwise-Decomposable Influence Diagrams, in *The Proc. of the 3rd Conference on the Principles of Knowledge Representation*, Cambridge, Mass. USA, October 26-29, 1992.

[91] L. Zhang, Runping Qi and D. Poole (1993a), Minimizing Decision Tables in Influence Diagrams, in *The Proc. of the Fourth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, Florida, January 3-6, 1993.

[92] L. Zhang, Runping Qi and D. Poole (1993b), Incremental computation of the value of perfect information in stepwise-decomposable influence diagrams, in *Proc. of 9th Conference on Uncertainty in Artificial Intelligence*, pp. 400-409.

[93] L. Zhang, Runping Qi and D. Poole (1993c), A computational theory of decision networks, accepted for publication on *International Journal of Approximate Reasoning*.

Index