Stochastic Simulation in Dynamic Probabilistic Networks Using Compact Representation

by

Adrian Y. W. Cheuk

B. Sc., The Chinese University of Hong Kong, 1994

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 26, 1996

Department of COMPUTER SCIENCE

The University of British Columbia
Vancouver, Canada

Date 96-08-30

Abstract

In recent years, researchers in the AI domain have used Bayesian belief networks to build models of expert opinion. Though computationally expensive deterministic algorithms have been devised, it has been shown that exact probabilistic inference in belief networks, especially multiply connected ones, is intractable. In view of this, various approximation methods based on *stochastic simulation* appeared in an attempt to perform efficient approximate inference in large and richly interconnected models. However, due to convergence problems, approximation in *dynamic probabilistic networks* has seemed unpromising. Reversing arcs into evidence nodes can improve convergence performance in simulation, but the resulting exponential increase in network complexity and, in particular, the size of the conditional probability tables (CPTs) can often render this *evidence reversal* method computationally inefficient.

In this thesis, we describe a structured simulation algorithm that uses the evidence reversal technique based on a *tree-structured representation* for CPTs. Most real systems exhibit a large amount of local structure. The tree can reduce network complexity by exploiting this structure to keep CPTs in a compact way even after arcs have been reversed. The tree also has a major impact on improving computational efficiency by capturing *context-specific independence* during simulation. Experimental results show that in general our *structured evidence reversal* algorithm improves convergence performance significantly while being both spatially and computationally much more efficient than its unstructured counterpart.

Contents

List of Tables

List of Figures

Acknowledgments

"I am the Alpha and the Omega, the First and the Last, the Beginning and the End."

— Revelation 22:13

In the beginning, the author, who was then more affiliated to the field of graphics, was less than ready to explore deeply into the field of (AI) simulation with his superficial knowledge of the field. During the research process of this thesis, there have been many struggles for the unseasoned author to try to understand various fundamental concepts. It is truly God's grace that the author could strive for his goal and in the end bring this thesis to its completion. The author hereby dedicates this thesis to the Lord, his God, acknowledging His unstinting help from beginning to end and bestowing the highest praise on Him. May His name be exalted and glorified, henceforth and forever!

The author would also like to express his heartfelt thanks towards Professor Craig Boutilier, the supervisor for this thesis, for his precious advice and unfailing support throughout the research. Professor Boutilier has been providing the author with a handful of inspirational ideas that led to the development of this thesis to its full extent. He has also been very nice, (which, in the author's opinion, is quite remarkable for a professor and is highly appreciated), complying with his motto, "Be nice to people." May his benevolence continue to benefit students to come, and may his professorial expertise further his career as well as research in the AI area.

Special thanks also go to Professor David Poole. Although the author only knows him by name, Professor Poole kindly promised to be the second reader of this thesis. Without his generous help and professional comments this thesis would certainly not have been complete.

One last note of thanks is reserved for the author's (high-school) alma mater, *Wah Yan College, Hong Kong*, where he learnt "how to treasure and keep for a lifetime all that's noble and true." Should chances arise that a *Wahyanite* will come across this thesis, he is reminded the following excerpt from the School song:

"In all that we do whether duty or pleasure,

We count not the cost, but unselfishly strive;

What's mean or unmanly we shun with displeasure;

Come praise or come blame, we hold our heads high."

With this "Wah Yan Spirit" in his heart, the author finally came through all difficulties and fulfilled his degree requirements. In the days to come, this Spirit will continue to live in his heart, and he will continue "to serve the Lord God in the love of our neighbour." May glory be with God, both now and forevermore. Amen!

1. Introduction

A central issue in AI has been to devise algorithms for an agent to make decisions under situations in the real world. In view of the presence of uncertainty in the decision process, a *probabilistic reasoning* approach has evolved to represent uncertainty in reasoning systems.

1.1 Probabilistic Reasoning and Belief Networks

The main advantage of probabilistic reasoning over traditional logical reasoning is in allowing the agent to reach rational decisions even when there is not enough information to prove that any action will "work" with certainty. Furthermore, the probabilistic approach has a clear theoretical basis combined with a theory of rational decision-making, an operational definition, a language for expressing uncertain dependence, and an ability to integrate diagnostic and predictive reasoning [HENR88]. The most promising current candidates for a coherent probabilistic representation appear to be the (Bayesian) **belief network** [PEAR88] and the closely related **influence diagram** [SHAC86]. In this thesis, we will use the belief network (BN) as a form of uncertainty representation. The interested reader may refer to Section 2.2.1 for an introduction to BNs.

1.2 Stochastic Simulation and Dynamic Probabilistic Networks

The basic task for any probabilistic inference system is to compute the posterior probability distribution for a set of **query variables**, given exact values for some **evidence variables**. In general, an agent gets values for evidence variables from its percepts (or from other reasoning), and asks about the possible values of other variables so that it can decide what action to take [RUNO95]. In order to answer such queries in a BN, various algorithms have been proposed and employed in practice. These inference algorithms, classified into three categories, either give an exact solution or an approximation of such. Since exact inference in BNs is known to be NP-hard [COOP90], for very large networks, and especially for **dynamic probabilistic networks** (DPNs), approximation using **stochastic simulation** (SS) is currently the method of choice. In Section 2.3 we will study the class of SS algorithms including *Logic Sampling* [HENR88], *Markov Simulation* [PEAR87], and *Likelihood Weighting* [SHPE90]. In Chapter 3 we will examine the application of SS in DPNs.

1.3 Problems and Solutions

In view of the dynamic nature of decision processes we would like to use SS in DPNs so that queries can be answered in a timely fashion. In so doing we would also like to solve the following problems that are brought forth.

1.3.1 Accuracy

The nature of simulation implies there is always an "error", or a deviation between simulation and the true behavior of the real-world process being simulated. While the DPN, being an extension of the BN, captures on the one hand the temporal aspect of real-world processes, it also magnifies the inherent errors of SS in BNs on the other, since "errors" in one time slice will be propagated to the following time slice and accumulated. Sequential error propagation will result in a significant deviation from the actual behavior of the process. We are interested in reducing this error to a minimum, thus bringing simulation "back on track". We will put our focus on the *Evidence Reversal* (ER) algorithm [KAKR95] in Section 3.2.

1.3.2 Compactness

Associated with each node in a BN or DPN is a **conditional probability table** (CPT). In standard SS algorithms, a CPT is encoded using a tabular representation. This simple representation fails to capture qualitative regularities in a CPT. In particular, there are independencies that hold only when specific values are assigned to certain variables. We are interested in capturing this kind of context-based irrelevance and the advantages of so doing. In Chapter 4 we will focus on one particular qualitative representation scheme — tree-structured CPTs [BFGK96].

1.3.3 Efficiency

In order to perform probabilistic inference in BNs and DPNs, SS methods sample the values of variables concerned. (Which variables are actually concerned, or *relevant*, will be discussed in Sections 5.1 and 5.2.) This requires the assignment of values to variables according to their probability distributions. As we would like to make inference in BNs and DPNs as efficient as possible, for any query we would like any SS method concerned to produce as accurate an answer using as few variable assignments, or *instantiations*, as possible. In Chapter 5 we propose several ways to achieve this goal.

1.4 Overview

Chapter 2 covers the background knowledge for BNs and inference in BNs, and surveys several inference algorithms that have major significance. Readers familiar with BNs may proceed directly to

Chapter 3, which focuses on DPNs and gives in detail the ER algorithm and the related *Arc Reversal* (AR) algorithm [SHAC86]. Chapter 4 introduces the *Structured Arc Reversal* (SAR) algorithm, detailing the construction of a CPT using a tree representation and the application of AR under such a representation. In Chapter 5 we present several techniques that, when combined with our *Structured Evidence Reversal* (SER) algorithm, will increase the efficiency of simulation and inference in DPNs. Chapter 6 features details of experiments conducted to examine the benefits that the tree-structured representation can bring to simulation and inference in DPNs. Conclusions and future work are presented in Chapter 7.

2. Background Knowledge

The real world is complex and dynamic in nature. In order to model changes in the world, we can perceive the world as a state space where each state represents a possible situation or configuration of the world. Every state has a degree of usefulness, or **utility**, to an agent. In different states, the agent has to make different decisions on what actions to perform to maximize the expected utility. In view of this, substantial research has been conducted in the AI planning domain.

2.1 Uncertainty

In the past, planning research has been unrealistic in that complete knowledge of both states and actions have been assumed. The realization that problems are often associated with uncertain initial conditions and action effects leads to growing interest in **uncertain** or **probabilistic reasoning**.

2.1.1 Types of Uncertainty

Uncertainty means that many of the simplifications that are possible with deductive inference are no longer valid. Below are three kinds of uncertainty described in [CHEU95].

Uncertainty of Action Effects

Given a state, we would like to choose the action that will maximize the expected utility. However, there may be chances that the effect of a certain action is not what is actually wanted or predicted. This can be described by a matrix of *transition probabilities* — probabilities associated with the possible transitions between states after a given action. In the machine-maintenance example in [SMSO73], if the alternative (i.e., action) "manufacture" is chosen, then the transition matrix for the machine is

$$\begin{bmatrix} 0.81 & 0.18 & 0.01 \\ 0 & 0.9 & 0.1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Given there is a 0.1 probability that any of the two internal components will break down during the manufacture of a product, this matrix represents all the transition probabilities under the action "manufacture" for a machine that begins the production cycle with zero, one, or two internal components that have failed and ends up in one of these states after the cycle.

Partial Observability

As mentioned before, an agent receives information about evidence variables through its percepts. These percepts, however, are not perfect in that they may not give complete, or even correct, information for the agent to determine the state in which it is. For example, an agent may have to decide whether to bring an umbrella by observing the weather, but its sensor may have a failure chance of 0.2 on rainy days. Thus, with a probability of 0.2 it may *think* it is sunny when it is actually raining. This **partial observability** is well captured in the **partially observable Markov decision process** (POMDP) [SMSO73] where the maximum extent of uncertainty is taken into account during the decision process.

Exogenous Events

In systems where process-oriented problems [BOPU95][BOUT95][CHEU95] typically arise, a third source of uncertainty comes from the existence of **exogenous events**. They occur from time to time, changing the state of a system in certain ways independent of the agent's actions. An important class of such events is the class of *user commands* such as "coffee requests". (A user request can be thought of to cause facts like "there is an outstanding request to do X" to become true.) Often, these events are represented by evidence variables an agent has to observe to choose the next action, and usually these variables are not completely observable.

2.1.2 Basic Probability Notation

Since an agent almost never has access to the whole truth about its environment, the agent's knowledge can at best provide only a **degree of belief** in relevant propositions. The main tool for dealing with degrees of belief will be **probability theory**, which assigns a numerical degree of belief between 0 and 1 to propositions.

Prior Probability

We use the notation $P(A)$ for the **unconditional** or **prior probability** that the proposition A is true. For example,

$$P(A) = 0.9$$

means that *without any other information*, the agent will assign a probability of 0.9 to the event of A being true.

A proposition is usually represented by an equality involving so-called **random variables** [PEAR88]. Each random variable X_i has a **domain** Ω_i of possible values it may assume. For example,

$$P(X_1=x_1) = 0.7$$

tells us the prior probability of X_1 having the value x_1, for some $x_1 \in \Omega_1$, is 0.7. We shall hereafter use the short notation $P(x_i)$ for the probabilities $P(X_i = x_i)$, $x_i \in \Omega_i$ for some random variable X_i, and we will write $P(x_J)$ for the probabilities $P(X_J = x_J)$, $x_J \in \Omega_J$ for some set of variables X_J. In the latter case, x_J is a *configuration* [PEAR88] of X_J, and Ω_J is the cross product space of the domains of all variables in X_J.

Conditional Probability

In the presence of other information (evidence), prior probabilities are no longer applicable. Instead, we use **conditional** or **posterior** probabilities, with the notation $P(A \mid B)$, meaning "the probability of A given *all we know* is B." For example,

$$P(x_1 \mid x_2) = 0.6$$

means that given only that X_2 has a value x_2, the probability of X_1 having the value x_1 is 0.6.

Probability Distribution

When we talk about the probabilities of all the possible values of a random variable X_i, we use the notation $\mathbf{P}(X_i)$ to denote X_i's **probability distribution** — a vector of values for the probabilities of each state of X_i. For example,

$$\mathbf{P}(X_1) = < 0.5, 0.3, 0.1, 0.1 >$$

tells us the probabilities of each of the four states of X_1. We may also write $\mathbf{P}(X_{i1}, ..., X_{in})$ to denote the probabilities of all combinations of the values of a set of random variables, getting a value for each element in the cross-product space $\Omega_{i1} \times \ldots \times \Omega_{in}$. In the case where $i1 = 1$ and $in = n$, we have a **joint probability distribution** (or "joint" for short) which completely specifies an agent's probability assignments to all propositions in the domain. From the joint we can compute any probabilistic statement in the domain by expressing the statement as a disjunction of *elementary events*[1] and adding up their probabilities. For example, if we have a domain comprising two random variables X and Y, and the joint for the domain can be represented by the table

	Y	$\neg Y$
X	0.04	0.06
$\neg X$	0.01	0.89

then $P(X)$ can be computed by $P(X, Y) + P(X, \neg Y) = 0.04 + 0.06 = 0.1$. In general, the probability of any variable X_i having a certain value x_i can be computed by

$$P(x_i) = \sum_{w_i} P(x_i, w_i),$$

where w_i ranges over the possible states of all variables except X_i.

While the joint contains all information we need about the domain, it is in general not practical to define all the $\prod_{i=1}^{n} \#\Omega_i$ entries for the joint over n variables. Representations that address this problem will be discussed in the next section. Hereafter we will focus on discrete variables, but it should

[1] An *elementary event* [PEAR88] is a conjunction in which every atomic proposition appears once.

be noted that continuous variables can be described using representations such as Gaussian conditional densities [PEAR88].

Conditional Independence

Conditional independence is crucial to making probabilistic systems work effectively. It is therefore beneficial to state its formal definition below [PEAR88].

Definition 2.1: Conditional Independence

Let X_N be a finite set of variables with discrete values. Let $P(\cdot)$ be a joint probability function over the variables in X_N, and let X_I, X_J, and X_K be any three subsets in X_N. X_I and X_J are said to be conditionally independent given X_K if

$$P(x_I \mid x_J, x_K) = P(x_I \mid x_K) \text{ whenever } P(x_J, x_K) > 0$$

for all $x_I \in \Omega_I$, $x_J \in \Omega_J$, and $x_K \in \Omega_K$.

■

Intuitively, what the definition says is that given three sets of variables X_I, X_J, and X_K, if $\mathbf{P}(X_I \mid X_J, X_K) = \mathbf{P}(X_I \mid X_K)$, then we say X_I and X_J are *conditionally independent* given X_K. We will use the notation $I(X_I, X_K, X_J)_\mathbf{P}$, or simply $I(X_I, X_K, X_J)$, to denote the conditional independence of X_I and X_J given X_K; thus,

$$I(X_I, X_K, X_J)_\mathbf{P} \text{ iff } P(x_i \mid x_j, x_k) = P(x_i \mid x_k)$$

for all values x_i, x_j, and x_k such that $P(x_j, x_k) > 0$. [2]

[2] The notation $I(X, Z, Y)_\mathbf{P}$ renders the probability distribution $\mathbf{P}$ a *dependency model*, i.e., a rule that determines a subset I of triplets (X, Z, Y) for which the assertion "X is independent of Y given Z" is true.

2.1.3 Bayes's Rule

We would often like to compute unknown probabilities from known ones, especially when we have conditional probabilities based on causal relationships and want to derive a diagnosis. (See Section 2.2.2 for the nature of probabilistic inferences.) For example, when reversing the arc (Section 3.3) between two arbitrary nodes i and j in a BN (Section 2.2.1), we have to compute $\mathbf{P}(X_i \mid X_j)$ from $\mathbf{P}(X_j \mid X_i)$, $\mathbf{P}(X_i)$ and $\mathbf{P}(X_j)$. Bayes's rule is employed for this purpose.

We can derive Bayes's rule from the **product rule**:

$$P(A \wedge B) \;=\; P(A|B)P(B), \tag{2.1}$$

where A and B are propositions and $\wedge$ is the logical-AND operator. It stems from the fact that for A and B to be true, we need B to be true, and then A to be true given B. From (2.1) we have

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}, \tag{2.2}$$

which is just the definition of conditional probabilities in terms of unconditional probabilities. By interchanging the variables in the proposition in (2.2) and applying the product rule, we get

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.3}$$

This equation is known as **Bayes's rule**. We can also use the **P** notation for probability distributions over random variables X_i and X_j:

$$\mathbf{P}(X_i|X_j)=\frac{\mathbf{P}(X_j|X_i)\mathbf{P}(X_i)}{\mathbf{P}(X_j)}. \tag{2.4}$$

If we have some background evidence in addition to B, we will have to use a more general version of Bayes's rule. By replacing B with $B \wedge E$ in (2.2) we can derive the following:

$$\begin{aligned}P(A|B\wedge E)&=\frac{P(A\wedge B\wedge E)}{P(B\wedge E)}=\frac{P(B\wedge A\wedge E)}{P(B\wedge E)}=\frac{P(B|A\wedge E)P(A\wedge E)}{P(B\wedge E)}\\&=\frac{P(B|A\wedge E)P(A|E)P(E)}{P(B|E)P(E)}=\frac{P(B|A\wedge E)P(A|E)}{P(B|E)}.\end{aligned}$$

Using the **P** notation for random variables X_i and X_j and the set of evidence variables X_E, we have

$$\mathbf{P}(X_i|X_j,X_E)=\frac{\mathbf{P}(X_j|X_i,X_E)\mathbf{P}(X_i|X_E)}{\mathbf{P}(X_j|X_E)} \tag{2.5}$$

2.2 Probabilistic Reasoning Systems

Although the joint probability distribution can answer any question about the domain, it becomes intractably large as the number of variables grows. In view of this, a number of inference algorithms have been proposed to make probabilistic inference efficient in many practical situations. The underlying inference mechanism centers around the BN, a data structure used to represent the direct dependence between variables and to give a concise specification of the joint probability distribution. In the following section, we describe the BN based on the literature in [PEAR88].

2.2.1 Belief Network

A BN is a **directed acyclic graph** (DAG)[3], G = (N,A), consisting of a set of nodes N and a set of directed arcs A. Each node i in N represents a variable X_i in the system. X_i may be binary or multi-valued. A directed arc (j,i) exists between i and j for each node j that has a direct causal influence on node i. The strengths of these influences are quantified by conditional probabilities stored in the CPT, or *link matrix*, of i; the link matrix $\mathbf{P}(X_i \mid X_{C(i)})$ describes the conditional distribution of X_i given different configurations of $X_{C(i)}$, the set of variables perceived to be direct causes of X_i.

Dependence Semantics

The semantics of a BN postulates a clear correspondence between the topology of a DAG and the dependence relationships it portrays. This correspondence is based on a separability criterion called *d*-separation.

Definition 2.2: d-*separation*

If X, Y, and Z are three disjoint subsets of nodes in a DAG D, then Z is said to *d*-separate X from Y, denoted $< X \mid Z \mid Y >_D$, if there is no path between a node in X and a node in Y along which the following two conditions hold:

1. Every node with both path arrows leading in is in Z or has a descendent in Z, and
2. Every other node is outside Z.

■

[3] A *DAG* is a graph that contains directed arcs but no uni-directional cycles. This implies the total number of arcs in a DAG cannot exceed $n(n-1)/2$, where n is the number of nodes in the graph.

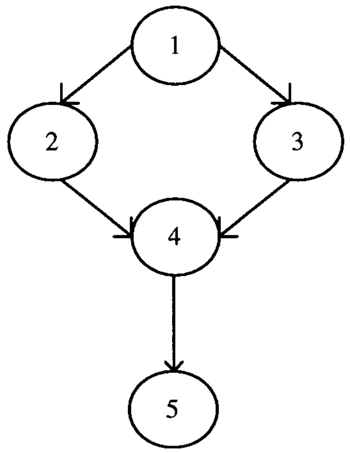


Figure 2.1: A multiply connected network

A path satisfying the above conditions is said to be *active*; otherwise, it is said to be *blocked* by Z. [RUNO95] puts this definition in a more direct way: a set of nodes Z d-separates two sets of nodes X and Y if every undirected path[4] from a node in X to a node in Y is blocked given Z. A path is blocked given a set of nodes Z if there is a node i on the path for which one of three conditions hold:

1. i is in Z and i has one arrow on the path leading in and one leading out.
2. i is in Z and i has both path arrows leading out.
3. Neither i nor any of its descendents is in Z, and both path arrows lead in to i.

Taking the example in [PEAR88], we can see in Figure 2.1 that $X = \{2\}$ and $Y = \{3\}$ are d-separated by $Z = \{1\}$; the path $2 \leftarrow 1 \rightarrow 3$ is blocked by $1 \in Z$, and the path $2 \rightarrow 4 \leftarrow 3$ is blocked because 4 and

[4] An *undirected path* is a path through the network that ignores the direction of the arrows.

all its descendents are outside Z. X and Y are not d-separated by $Z' = \{1, 5\}$, however, because the path $2 \rightarrow 4 \leftarrow 3$ is rendered active: learning the value of node 5 makes nodes 2 and 3 dependent.

Together with the above definition, the following two definitions help define a BN from a dependence perspective.

Definition 2.3: I-map

A DAG D is said to be an I-map of a dependency model M if every d-separation condition displayed in D corresponds to a valid conditional independence relationship in M, i.e., if for every three disjoint sets of nodes X, Y, and Z we have

$$< X \mid Z \mid Y >_D \Rightarrow I(X, Z, Y)_{\mathrm{M}}.$$

A DAG is a minimal I-map of M if none of its arrows can be deleted without destroying its I-mapness.

■

Definition 2.4: Bayesian Network

Given a probability distribution P on a set of variables X_N, a DAG $D = (N, A)$ is called a Bayesian Network of P iff D is a minimal I-map of P.

■

Given the above definitions, we see that all conditional independencies portrayed in a BN (by way of d-separation) are valid in the underlying distribution **P** and no two independencies therein are redundant.

Joint Representation

The usefulness of the BN, and indeed any inference network, rests on the assumption that knowledge is decomposable and can be represented by a relatively sparse graph, where each variable is directly influenced by only a few other variables [HENR88]. If a topological ordering of nodes is maintained, the BN serves as a correct decomposed representation of the joint for the domain. This can be shown by first considering a generic entry in the joint and rewriting it using the definition of conditional probability:

$$P(x_1,\ldots,x_n) = P(x_n \mid x_{n-1},\ldots,x_1)P(x_{n-1},\ldots,x_1).$$

By successively reducing each conjunctive probability to a conditional probability and a smaller conjunction, we obtain

$$\begin{aligned} P(x_1,\ldots,x_n) &= P(x_n \mid x_{n-1},\ldots,x_1)P(x_{n-1} \mid x_{n-2},\ldots,x_1)\cdots P(x_2 \mid x_1)P(x_1) \\ &= \prod_{i=1}^{n} P(x_i \mid x_{i-1},\ldots,x_1) \end{aligned}$$

Given the conditional independencies encoded in the BN, we observe that

$$P(x_i \mid x_{i-1},\ldots,x_1) = P(x_i \mid x_{C(i)}),$$

provided that $C(i) \subseteq \{1, 2, \ldots, i\text{-}1\}$ for all $i \in N$. As a result, we have

$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} P(x_i \mid x_{C(i)}), \qquad (2.6)$$

provided that $C(i) \subseteq \{1, 2, ..., i\text{-}1\}$ for all $i \in N$. This shows that a BN represents each entry in the joint by the product of the appropriate elements of the CPTs in the network given a topological node ordering (which can easily be satisfied by labeling the nodes in any order that is consistent with the partial order implicit in the graph structure).

In essence, the CPT provides a decomposed representation of the joint, thus making the BN more compact than the full joint especially in **locally structured systems** [RUNO95], where each variable is directly influenced by at most k others, for some constant $k << n$. Assuming a variable has at most q values, the amount of information needed to specify the CPT for a node will be at most q^k numbers, so the complete network can be specified by nq^k numbers. The joint, on the other hand, contains q^n numbers.

2.2.2 Inference in Belief Networks

There are two major types of inference we would like to perform. **Predictive** or **causal** inference involves reasoning from evidence about root nodes down through the network in the direction of the arcs to the leaf nodes, e.g., from a disease to its symptoms. **Diagnostic** inference involves reasoning in the reverse direction, e.g., from observations of symptoms to diseases. A coherent probabilistic inference scheme can support both predictive and diagnostic inference and combinations of the two, according to what evidence is available and what hypotheses are of interest [HENR88]. While such algorithms do exist for **singly connected** networks (e.g., Pearl's polytree algorithm), our main concern is how we can make accurate and efficient inference in **multiply connected** networks. A multiply connected graph is one in which two nodes are connected by more than one path. Figure 2.1 is an example of such a graph, where the parents of node 4 share a common ancestor (node 1).

There are three classes of algorithms for evaluating multiply connected networks [PEAR88]:

- Clustering
- Conditioning
- Stochastic simulation

The first two yield exact solutions by transforming the network while the third gives an approximation of the exact solution by generating a large number of sample models consistent with the network distribution. Since exact inference in belief networks is known to be NP-hard (and so is approximate inference), for very large networks, the appropriate choice of evaluation algorithm would be approximation using SS. The next section will cover several SS algorithms of significance.

2.3 Stochastic Simulation Methods

Currently there are quite a number of belief network inference algorithms that belong to this category. In this sampling-based simulation approach, conditional probabilities and other statistics are estimated by recording the fraction of times that events occur in a random series of sample instantiations of the network where each variable is assigned a particular deterministic value. The size of the sample set governs the accuracy of simulation: the more samples (or "scenarios" [HENR88]) we have, the more accurate the approximation.

2.3.1 Approaches

[HENR90] classifies these algorithms into two major sub-categories:

- forward propagation
- Markov simulation

The major difference between the two approaches lies in the way the nodes are updated. The first method sequentially updates all variables in each simulation run according to some topological ordering of the nodes in the network. The instantiation of a variable then depends on the current instantiations of its parents. The second method, however, performs in each simulation cycle only local numerical computations for a variable; variables are updated in a random order in successive cycles and, in each cycle, depend on the instantiations of neighboring node variables in the previous cycle. Figure 2.2 shows a diagrammatic classification of various simulation algorithms.

Forward Propagation

In the **forward propagation** (or forward simulation) approach, each instantiation is created by following the influence arrows in the network starting from the root nodes. In other words, the sequence of instantiations follows a particular topological node order consistent with the graph structure. In this way, we can make sure all nodes are instantiated before any other nodes dependent on them. This criterion in turn ensures that the process of sampling will continue until all nodes in the network are instantiated.

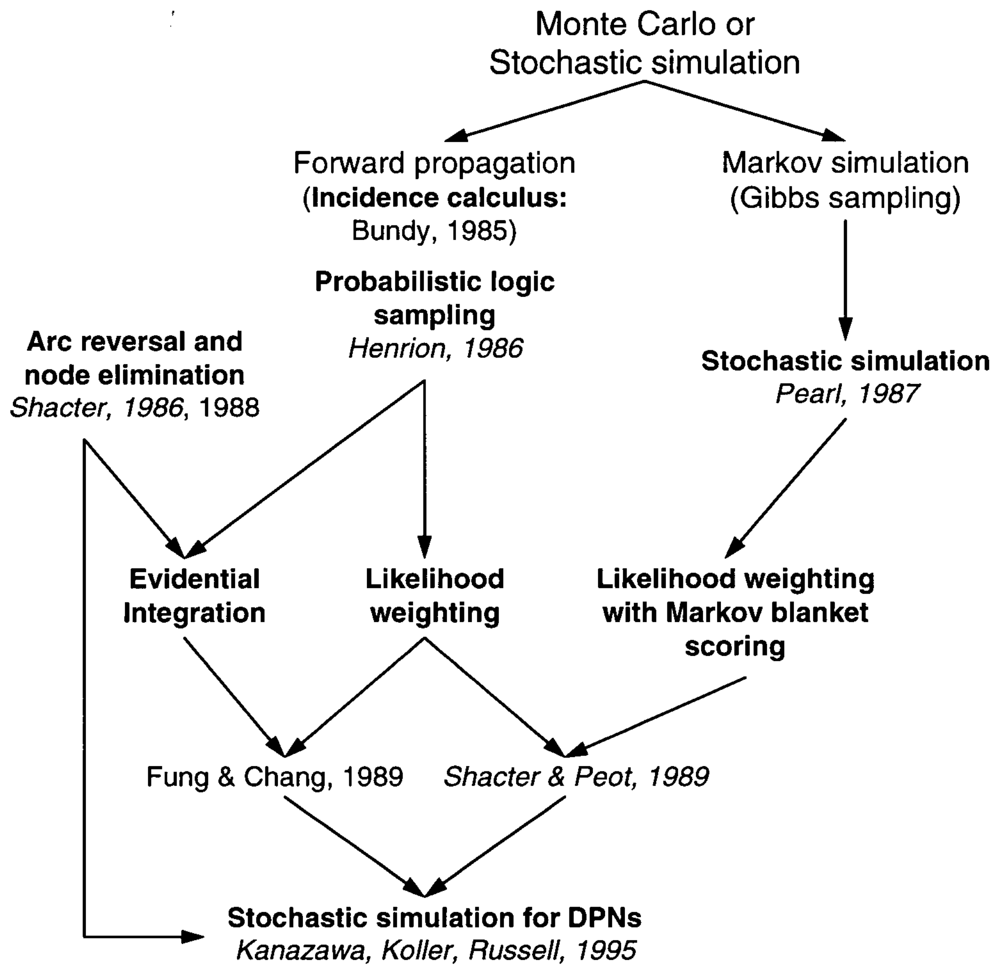

Monte Carlo or Stochastic simulation
Forward propagation (Incidence calculus: Bundy, 1985)
Markov simulation (Gibbs sampling)
Probabilistic logic sampling Henrion, 1986
Arc reversal and node elimination Shacter, 1986, 1988
Stochastic simulation Pearl, 1987
Evidential Integration
Likelihood weighting
Likelihood weighting with Markov blanket scoring
Fung & Chang, 1989
Shacter & Peot, 1989
Stochastic simulation for DPNs Kanazawa, Koller, Russell, 1995

Figure 2.2: Monte Carlo simulation approaches to inference in BNs. (Adapted from *[HENR90]*) *Italics refer to articles examined in detail in this thesis.*

Essentially we completely sweep through the network in each simulation run, instantiating all nodes from top to bottom according to their specified priors (if they are source nodes) or conditional probabilities (if otherwise). The sampled values of the nodes are then recorded. After a specified number of runs, diagnostic inference is performed by estimating the probability of a hypothesis as the fraction of simulations that lead to the observed set of evidence. In Section 2.3.2, we will examine the simplest yet most fundamental forward propagation scheme — Henrion's *probabilistic logic sampling* [HENR88] — on which many later algorithms are based.

Markov Simulation

In **Markov simulation** (originally developed by Pearl [PEAR87]; see Section 2.3.2), propagation is in either direction along arcs, generally in a random sequence, and is localized within the *Markov blanket* of each variable. The Markov blanket consists of the variable's parents, children, and children's parents, and it shields the variable from the rest of the network. In each simulation cycle, a variable is chosen at random and its conditional distribution given all its Markov neighbors is first computed. (This means the instantiation of a variable is influenced by the *previous* instantiations of its neighbors.) The distribution is then sampled and the variable instantiated accordingly. As simulation continues, the probability of each node can be estimated on demand as the fraction of simulation cycles for which it is true.

2.3.2 Algorithms

We will examine some of the more influential SS algorithms below. The notations used are defined in Table 2.1. Probability notations follow those in Section 2.1.2 with short forms as in equation (2.6). For example, $P(x_{J1}|x_{J2})$ is the short form for $P(X_{J1}=x_{J1}|\ X_{J2}=x_{J2})$, for some $J1$, $J2 \subseteq$ N.

Logic Sampling [HENR88]

Table 2.2 gives additional definitions with respect to arbitrary propositions A and B. Since the algorithm assumes the use of binary variables, the logic sample $L(X_i=1)$ automatically records both the

Table 2.1: Global definitions

Symbol	/	Definition	Description
N	=	$\{1, \ldots, n\}$	set of all n nodes in the system, for some $n \in$ N
E	$\subset$	N	set of evidence nodes
C(i)	$\subset$	N	set of conditional predecessors of node i for some $i \in$ N
S(i)	$\subset$	N	set of direct successors of node i for some $i \in$ N
X_i			variable associated with node i for some $i \in$ N
Ω_i			set of possible values X_i may assume, for some $i \in$ N
x_i	$\in$	Ω_i	an instantiation of X_i, for some $i \in$ N
X_J	=	$(X_{j1}, \ldots, X_{jn})$	set of variables associated with J for some $J = \{j1, \ldots, jn\} \subseteq$ N
Ω_J	=	$\Omega_{j1} \times \ldots \times \Omega_{jn}$	cross-product space of J for some $J = \{j1, \ldots, jn\} \subseteq$ N
x_J	$\in$	Ω_J	an instantiation of X_J, for some $J \subseteq$ N

Table 2.2: Definitions for Logic Sampling

Symbol	/	Definition	Description
$L_k(B)$	=	$\begin{cases} 1, & \text{if } x \text{ is true in scenario } k \\ 0, & \text{otherwise} \end{cases}$	truth of B in *scenario*[5] k for some $k \in [1, m]$, $m \in$ N
$L_k(B\|A)$	$\equiv$	$(L_k(A) \Rightarrow L_k(B))$	truth of B conditioned on A in scenario k for some $k \in [1, m]$, $m \in$ N
$L(B)$	$\equiv$	$[L_1(B), \ldots, L_m(B)]$	*logic sample*[6] of B over all m scenarios for some $m \in$ N

truth and falsehood of X_i over all scenarios. For convenience, here we will use x_i to represent the truth of variable X_i; thus $L(x_i)$ denotes the logic sample of the proposition $X_i = 1$.

This scheme uses an uninstantiated belief network as a scenario (or possible world) generator which, in each scenario k, assigns random values to all system variables. Starting with priors specified

[5] The term "scenario" can be related to the more commonly used term "sample" that reflects one of the many possible worlds in the domain. In Henrion's context, a scenario is a set of truth values for those propositions concerned.

[6] Note the difference between the *sample* and the *logic sample*. The former contains the multi-valued states of all variables in one simulation run; the latter contains the binary states (truth or falsehood) of one proposition over all runs.

for all source variables and conditional distributions for all others, a random number generator is used to generate a sample truth value $L_k(x_i)$ for each source variable X_i and a *conditional* truth value $L_k(x_j|x_{C(j)})$ for all $x_{C(j)} \in \Omega_{C(j)}$, for each non-source variable X_j. Then by following the arrows from the source nodes, the truth of each variable is obtained using the following logical operation:

$$L_k(x_j) = \bigvee_{x_{C(j)} \in \Omega_{C(j)}} L_k(x_j | x_{C(j)}) \wedge L_k(x_{C(j)}), \qquad (2.7)$$

where $\bigvee_{i \in N}$ is the logical OR operator $\vee$ defined over some finite set N.[7] The simulation process then repeats itself m times, after which a logic sample $L(x_i)$ is obtained for each variable X_i. Belief distributions are then calculated by averaging the frequency of events over those cases in which the evidence variables agree with the data observed. The algorithm is as shown in Algorithm 2.1.

After logic samples are obtained, we can estimate the prior probability of any simple or compound event x by the *truth fraction* of its logic sample:

$$P(x) \approx T[L(x)] \equiv \sum_{k=1}^{m} L_k(x) / m.$$

In other words, $T[L(x)]$ is the proportion of scenarios in which x is true.

We can also estimate the conditional probability of any event x conditioned on any set of observed evidence y by

[7] This is the deterministic counterpart of the probabilistic chain rule: $P(x_j) = \sum_{x_{C(j)} \in \Omega_{C(j)}} P(x_j | x_{C(j)}) P(x_{C(j)})$.

Algorithm 2.1: Logic Sampling

```
/* We start with a belief network with $\mathbf{P}(X_i)$ specified $\forall$ root variables $X_i$ &
   $\mathbf{P}(X_j|X_{C(j)})$ specified $\forall$ non-root variables $X_j$. */
procedure LOGIC_SAMPLING( priors, CPTs )

    /* Repeat $m$ times to obtain $L(x_i)$ for each variable $X_i$ */
    for $k = 1$ to $m$
        for each $X_i \in$ {root variables}

            /* Use a random-number generator to "flip the coin " */
            Produce $L_k(x_i)$ according to $\mathbf{P}(X_i)$;

        /* Sweep thru the network following the arrows from the root nodes to get a
           truth value for each non-root variable $X_j$. */
        for each $X_j \in$ {non-root variables}
            $L_k(x_j) = 0$;
            for each $x_{C(j)} \in \Omega_{C(j)}$
                Produce $L_k(x_j \mid x_{C(j)})$ according to $\mathbf{P}(X_j \mid x_{C(j)})$;
                $L_k(x_j) = L_k(x_j) \vee L_k(x_j \mid x_{C(j)}) \wedge L_k(x_{C(j)})$;
```

$$P(x|y) = \frac{P(x,y)}{P(y)} \approx \frac{T[L(x \wedge y)]}{T[L(y)]} = \frac{T[L(x) \wedge L(y)]}{T[L(y)]} = \frac{\sum_{k=1}^{m} L_k(x) \wedge L_k(y)}{\sum_{k=1}^{m} L_k(y)}.$$

An obvious advantage of logic sampling is that logic samples can be efficiently represented using bitstrings. All operations (i.e., equation (2.7)) to compute logic samples for derived variables and compound events then involve simple Boolean operations on these bitstrings. Moreover, for a given level of precision, the complexity $O(mn)$ is linear in the number of nodes, irrespective of the degree of connectedness of the graph. Since all the sample instantiations are independent, it is also possible to estimate the precision of probability estimates as a function of the sample size using standard statistical methods: the standard deviation σ in probability estimates is inversely proportional to $\sqrt{m}$ [HENR88]. As the required precision increases, however, we experience a quadratic increase in m ($\propto 1/\sigma^2$), and hence complexity.

The main problem, however, arises when probability estimates are desired *given some observed evidence.* In logic sampling there is no way to account in advance for evidence known to have occurred until the corresponding variables are sampled. If the sampled values conflict with the observed evidence, then the sample has to be discarded. For rare combinations of evidence, this may result in an excess number of simulation runs. In fact, the complexity of logic sampling is exponential in the number of observed evidence variables [HENR88][HENR90]. Despite its drawbacks in the complexity aspect, logic sampling is of importance in its concept of sample generation which leads to many later algorithms.

Stochastic Simulation [PEAR87]

In view of the negligence of evidence in logic sampling, Pearl devised a two-phase Markov simulation algorithm which emphasizes local numerical computation and clamps the evidence variables to the values observed. It first computes the conditional distribution for some variable X given the states of all its neighboring variables. Then it samples the distribution computed in the first step and instantiates X to the value selected by the sampling. The cycle then repeats itself by sequentially scanning through all the variables in the system. The algorithm is described in pseudo-code in Algorithm 2.2. A variable superscripted by an asterisk (*) represents a given value of the variable; when a variable is topped by a caret (^), it means the current sampled value of the variable.

Algorithm 2.2: Pearl

```
/* Given: $\mathbf{P}(X_i)$ ∀ root variables $X_i$, $\mathbf{P}(X_j|X_{C(j)})$ ∀ non-root variables $X_j$, $X_E = x^*_E$
   for some $x^*_E \in \Omega_E$ */
procedure PEARL( priors, CPTs, $x^*_E$ )
        Instantiate $X_E$ to $x^*_E$;

        /* Initial instantiation by forward simulation */
        for each $i \in N \setminus E$

                /* Reset arrays for use w/ probability estimation */
                for each $x_i \in \Omega_i$
                        $sum_i(x_i) = P(x_i | \hat{x}_{C(i)})$;          /* or $P(x_i)$ if $X_{C(i)} = \emptyset$ */
                $count_i = 1$;

                /* Sweep thru the net using a random-number generator to
                   "flip the coin " */
                Instantiate $X_i$ according to $\mathbf{P}(X_i)$ or $\mathbf{P}(X_i | X_{C(i)})$;

        for ever
                Choose some $i \in N \setminus E$;
                for each $x_i \in \Omega_i$

                        /* Compute $\mathbf{P}(X_i)$ using Markov blanket */
                        $P(x_i | \hat{x}_{N\setminus\{i\}}) = P(x_i | \hat{x}_{C(i)}) \prod_{j \in S(i)} P(x_j | \hat{x}_{C(j)})$;

                        $sum_i(x_i) = sum_i(x_i) + P(x_i | \hat{x}_{N\setminus\{i\}})$;
                Normalize $\mathbf{P}(X_i | \hat{x}_{N\setminus\{i\}})$ so that $\sum_{x_i \in \Omega_i} P(x_i | \hat{x}_{N\setminus\{i\}}) = 1$;
                $count_i = count_i + 1$;
                Flip the coin for $X_i$ using $\mathbf{P}(X_i | \hat{x}_{N\setminus\{i\}})$;
                if ( query )    /*    Estimate $P(x_q | x^*_E)$ for some $q \in N$, $x_q \in \Omega_q$ */
                        Input $q$, $x_q$;
                        $p = sum_q(x_q) \,/\, count_q$;
                        Output $p$;
```

The advantage of Markov simulation is that the algorithm can be implemented as a network of parallel processors simulating the happening of concurrent events. In general, however, if the network contains conditional probabilities close to 0 or 1, convergence will be very slow [SHPE90] [HENR90]. Moreover, since successive cycles in Markov simulation are not independent, the simulation can get trapped in particular states or sets of states [HENR90].

Another limitation is that we can only estimate the conditional probabilities for single variables but not the joint probabilities of sets of variables. In the algorithm we have already used as many as $\sum_{i=1}^{n} \# \Omega_i$ accumulators (i.e., the array *sum*) to refine $P(x_i | x_E^*)$ for all $x_i \in \Omega_i$, $i \in$ N. Yet this only accounts for the conditional probabilities for each value of each variable considered separately. It is infeasible, however, to account for the joint probabilities of J for all $J \subseteq$ N using time and space proportional to $\sum_{J=1}^{k} \# \Omega_J$ where $k = \sum_{r=1}^{n} {}_nC_r$ is the number of possible subsets in N.

Likelihood Weighting [SHPE90][8]

Shacter and Peot extended logic sampling by assigning weights, or scores, to the samples generated. Instead of discarding samples that conflict with the observed evidence, each sample is weighted by the joint likelihood of the observations conditioned on their unobserved predecessors. Table 2.3 introduces some new definitions.

Essentially we are assigning a score, Z, to any given sample s_k selected from the joint distribution of X_N. The score is equal to the original probability of s_k divided by the probability of <u>selecting</u> s_k, i.e., generating s_k by likelihood weighting. Thus,

[8] [FUCH90] proposed the same algorithm, which they called "evidence weighting".

Table 2.3: Definitions for Likelihood Weighting

Symbol	/	Definition	Description
s_k	$\equiv$	$[\hat{x}_1, \ldots, \hat{x}_n]$	the k-th sample
Z_k	$\in$	$\Re$	score of the k-th sample
$Z(X_i=x_i)$	$\in$	$\Re$	cumulative score over those samples in which $X_i=x_i$, for some $i \in \mathrm{N}$, $x_i \in \Omega_i$

$$Z_k = \frac{P(s_k | x_E^*)}{P(\text{selecting } s_k | x_E^*)} = \frac{\prod_{i \in N} P(x_i | x_{C(i)})}{\prod_{i \notin E} P(x_i | x_{C(i)})} = \prod_{i \in E} P(x_i | x_{C(i)}).$$

The algorithm, which they called *Basic*, is shown in Algorithm 2.3.

If we have a set of query variables X_Q and we want to find out the conditional probability for $X_Q = x_Q$ for some $x_Q \in \Omega_Q$, $Q \in \mathrm{N}$, we can make an estimation by normalizing $Z(X_Q = x_Q)$:

$$P(x_Q | x_E^*) \approx \frac{Z(X_Q = x_Q)}{\sum_{k=1}^{m} Z_k}. \tag{2.8}$$

This step can also be incorporated into the Basic algorithm in a similar fashion as Markov simulation so that simulation can be interrupted at any time to return the best quality answer thus far.

Just like Markov simulation, likelihood weighting (LW) is suitable for parallel processing architectures. It also avoids the problem with logic sampling where samples inconsistent with the evidence are discarded. Unlike Markov simulation, however, likelihood weighting can also account for the joint probabilities of sets of variables rather than just the conditional probabilities for single variables. We can simply compare x_Q with the values of the variables in X_Q over all samples, calling for

Algorithm 2.3: Basic

```
/* Given: $\mathbf{P}(X_i)$ ∀ root variables $X_i$, $\mathbf{P}(X_j|X_{C(j)})$ ∀ non-root variables $X_j$, $X_E = x^*_E$
  for some $x^*_E \in \Omega_E$ */
procedure BASIC( priors, CPTs, $x_E^*$ )
    Instantiate $X_E$ to $x_E^*$;
    for $k = 1$ to $m$

        /* sweep through by forward simulation */
        for each $i \in N \setminus E$
            Flip the coin for $X_i$ according to $\mathbf{P}(X_i)$ or $\mathbf{P}(X_i | X_{C(i)})$;
            Add $\hat{x}_i$ to $s_k$;

        /* Compute the score for the $k$-th sample */
        $Z_k = \prod_{i \in E} P(x_i^* | \hat{x}_{C(i)})$;
```

only $m\#Q$ comparisons. For example, if we want to estimate $P(X_2{=}true, X_3{=}true \mid X_5 = true)$ in Figure 2.1, assuming all variables are binary, then we can go through each sample (with X_5 clamped to *true*) and compare the values of X_2 and X_3 with the query. If the values match in sample k, then we add Z_k to $Z(X_2{=}true, X_3{=}true)$; otherwise, the next sample is considered. Assuming there are $m = 1000$ samples, we only need $1000 \times 2 = 2000$ comparisons in total.

There are cases where likelihood weighting performs quite poorly. When the likelihood product (score) varies greatly among samples most of the samples are effectively ignored, and therefore additional samples must be taken. [SHPE90] suggested we can reverse the arcs into those evidence nodes most responsible for the variation in likelihood to remedy the situation. [SHAC86] gave a detailed theoretical derivation for inference with arc reversals. We will examine its application in DPNs in the next chapter.

3. Temporal Processes

While a BN is an attractive approach for representing uncertain expert knowledge in a coherent probabilistic form, it has seldom been focused on the temporal aspect of real-world processes. In order to capture this dynamic nature, we use a data structure called the **dynamic probabilistic network**[9] (DPN).

3.1 Dynamic Probabilistic Network

The DPN can be thought of as a chain of BNs joined by temporal links (Figure 3.1). Each unit of the chain is a **time slice** representing a snapshot of the evolving temporal process. If we have *n* time slices, we will have *n* BNs chained together, in a way such that nodes in time slice *t* are connected only to nodes in time slice $t + 1$ and other nodes in slice *t*. In other words, the history of the process can be ignored; or more precisely, information at time *t* summarizes all relevant features of the world. This property is called the **Markov property**. For temporal projection (i.e., prediction from the state at some time into the future), this property allows us to focus on only two time slices at a time, moving the "window of focus" in one-slice increments.

[9] The idea first appeared in [DEKA89] and was reiterated in [KAKR95]. It was called *dynamic belief network* (DBN) in [RUNO95].

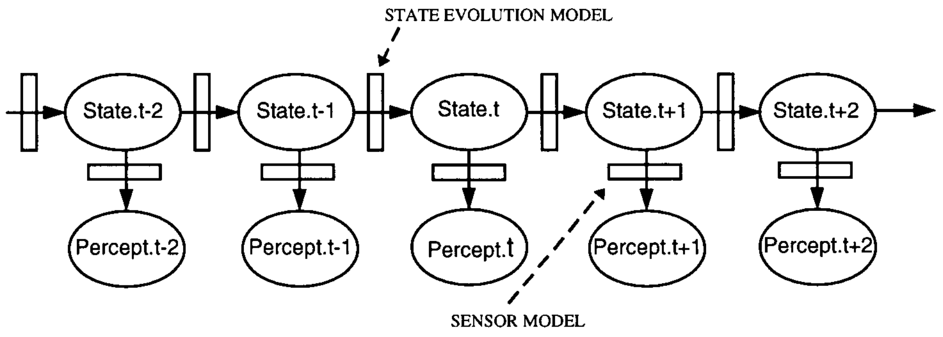


Figure 3.1: Generic structure of a DPN (adapted from [KAKR95]). *In an actual network, there may be many state and sensor variables in each time slice.*

3.2 Evidence Reversal

Standard simulation algorithms such as those in Section 2.3.2 often give fast, accurate approximations to posterior probabilities in belief networks, and are the methods of choice for very large networks. However, when we come to temporal processes, these algorithms sometimes perform very poorly. In essence, the simulation trials diverge further and further from reality as the process is observed over time. In the case of logic sampling, because it discards trials whenever a variable instantiation conflicts with observed evidence, it is likely to be ineffective in DPN-based tasks where evidence is observed throughout the temporal sequence.[10] As for likelihood weighting, a straightforward application generates simulations that simply ignore the observed evidence and therefore become increasingly irrelevant. This is especially true in systems where the state evolution model is weak (i.e., more or less random). Although uncertainty in observation may be very low, the samples are nonetheless evolved according to the state evolution model. A weak model (which is typical in systems where autonomous

[10] On the contrary, logic sampling is very effective for projection because no evidence is observed in future slices.

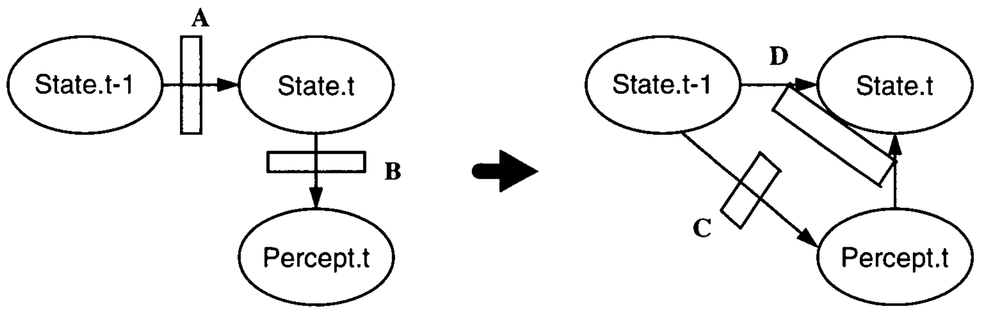


Figure 3.2: Schematic diagram of evidence reversal transformation for DPNs: ***A*** *is the state evolution model;* ***B*** *is the sensor model;* ***C*** *and* ***D*** *are the new CPTs we have to compute.* (Adapted from *[KAKR95]*)

agents are monitored) means the sample distribution will get more random over time, unrelated to the observed evidence. As a result, the weighting process will assign extremely low weights to (or even discard) almost all the samples. Only a very small number of samples closest to the true state will obtain relatively high scores and thus dominate the estimated distribution. The effective number of samples therefore diminishes rapidly over time, resulting in large estimation errors.

In general, forward simulation methods will ignore the *actual* observations made when determining the sample state trajectory and *future* observations. In Figure 3.1 this is due to the fact that the causal direction of influences does not permit our observations (percepts) to bias the state samples. As a remedy, [KAKR95] adopted Shacter's suggestion and added an arc reversal (AR) step to likelihood weighting for use with DPNs. However, instead of reversing all the arcs into the evidence over time, we simply reverse the arcs within slice *t*, so that the evidence at *t* and the state at *t* - 1 become the parents of the state at time *t* (Figure 3.2).[11] This is possible because each sample, once it has instanti-

[11] The idea is based on the *evidential integration* method proposed in [FUCH90]. In evidential integration only the arcs between the evidence and its parents are reversed (in non-temporal BNs), thus "partially" integrating the evidence into the network.

ated variables in time slice t - 1, d-separates all preceding time slices from the state at time t. After AR, the current evidence becomes a parent of the current state; therefore, it can influence the process of extending the samples to the state variables at t. Consequently, the sample population can be repositioned closer to reality according to the observed evidence. In Chapter 6 we will show in an experimental case how convergence may be improved by reversing arcs into evidence.

Algorithm 3.1 shows the evidence reversal algorithm (which is modified from the likelihood weighting algorithm). Where necessary we have added time arguments ranging from 0 to T, for some $T \in \mathbb{N}$. If we want to know $P(x_Q(t)|x_E^*(t))$ for some $x_Q(t) \in \Omega_Q$, $t \in [0,T]$, we can apply equation (2.8) to time slice t. Note that we assume X_E remains constant over time, i.e., the set of evidence variables is fixed. This is in general the case for decision-theoretic agents since they base their actions on the same set of observable events over time (i.e., their perceptual inputs). If the opposite is true, however, then the arc reversal step (marked with `/* ( 1 ) */`) has to be placed inside the loop (at the place marked `/* ( 2 ) */`). This means for each time slice we have to reverse arcs into the evidence variables in that time slice, thus making the simulation process much slower.

From the pseudocode we see [KAKR95] laid out their algorithm in such a way that m partial samples are generated and recorded in one slice followed by another m partial samples in another slice. While this can be made to work (though there is insufficient detail in [KAKR95]), it is not clear what advantages this approach offers. On the contrary, a considerable amount of work has to be done to book-keep each sample since the temporal process is repeatedly sampled in its entirety. In Section 5.1 we will present the evidence reversal algorithm in a query-directed approach that will alleviate this problem.

Algorithm 3.1: Likelihood weighting with evidence reversal (adapted from [KAKR95])

```
/* Given: CPTs ∀ variables; we assume the CPTs to remain constant over time */
/* Modified from BASIC */
procedure LIKELIHOOD_WEIGHTING( CPTs )

        /* Time slice 0 */
        Obtain $x_E^*(0)$;
        Instantiate $X_E$ to $x_E^*(0)$;

        /* ( 1 ) */
        Reverse arcs into $E$;

        for $k = 1$ to $m$
                Sweep through the net, flipping the coin for each variable;
                Add $\hat{x}_N(0)$ to $s_k$;

                /* Compute the score for the k-th sample at time 0 */
                $w_k = \prod_{i \in E} P(\hat{x}_i(0) | \hat{x}_{C(i)}(0))$;
                $Z_k(0) = w_k$;

        /* Time slices 1 to T */
        for $t = 1$ to $T$
                Obtain $x_E^*(t)$;
                Instantiate $X_E$ to $x_E^*(t)$;

                /* ( 2 ) */

                for $k = 1$ to $m$
                        Sweep through the net, flipping the coin for each variable;
                        Add $\hat{x}_N(t)$ to $s_k$;

                        /* Compute the score for the k-th sample at time t */
                        $w_k = \prod_{i \in E} P(\hat{x}_i(t) | \hat{x}_{C(i)}(t))$;
                        $Z_k(t) = Z_k(t-1) + w_k$;
```

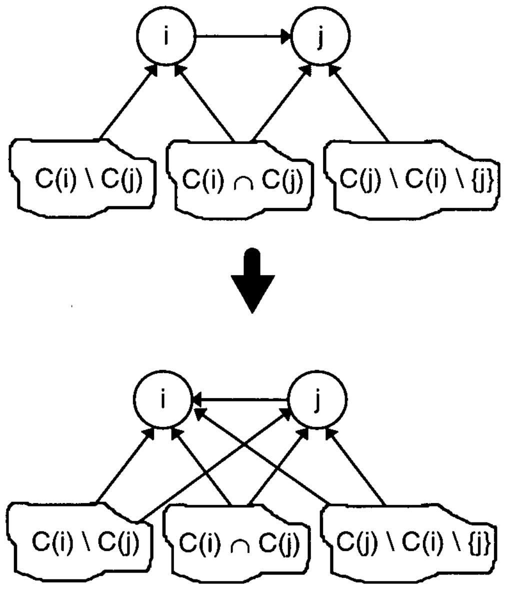


Figure 3.3: Reversing an arc between arbitrary nodes i *and* j. (Adapted from *[SHAC86]*)

3.3 Arc Reversal

The arc reversal step is critical in that it uses the current evidence to bring the simulation closer to reality. [SHAC86] proved that for all $i, j \in \mathrm{N}$, arc (i, j) can be replaced by arc (j, i) if there is one and only one arc (i, j) between nodes i and j. After the reversal, both nodes inherit each other's parents. More formally,

$$\begin{aligned} \mathrm{C}^{\mathrm{new}}(j) &= \mathrm{C}^{\mathrm{old}}(i) \cup \mathrm{C}^{\mathrm{old}}(j) \setminus \{i\} \\ \mathrm{C}^{\mathrm{new}}(i) &= \mathrm{C}^{\mathrm{new}}(j) \cup \{j\} \end{aligned} \qquad (3.1)$$

Figure 3.3 is a pictorial representation of the arc reversal step. Besides the parents, the CPTs for i and j are modified as follows:

$$P(x_j | x_{C^{new}(j)}) = \sum_{x_i \in \Omega_i} P(x_j | x_{C^{old}(j)}) P(x_i | x_{C^{old}(i)}) \qquad (3.2)$$

$$P(x_i | x_{C^{new}(i)}) = \frac{P(x_j | x_{C^{old}(j)}) P(x_i | x_{C^{old}(i)})}{P(x_j | x_{C^{new}(j)})} \qquad (3.3)$$

for all $x_i \in \Omega_i, x_j \in \Omega_j, x_{C^{new}(j)} \in \Omega_{C^{new}(j)}$. Putting this into our DPN, we are essentially using **A** and **B** to compute **C** and then **D** in Figure 3.2. Equation (3.2) lets us compute each entry in **C** while **D** is computed by equation (3.3), which is just the Bayesian inversion formula, or Bayes's rule. (The denominator can be seen as a normalizing constant if we substitute the first equation into the second.) To be more specific, let X_S', X_S, and X_E be *State*(*t*-1), *State*(*t*) and *Percept*(*t*) respectively. Then we have, for each entry of **C**,

$$P(x_E | x_S') = \sum_{x_S \in \Omega_S} P(x_E | x_S) P(x_S | x_S'),$$

and, for each entry of **D**,

$$P(x_S | x_E, x_S') = \frac{P(x_E | x_S) P(x_S | x_S')}{P(x_E | x_S')}.$$

The AR process is then complete.

A major drawback of AR is the increase in size of CPTs, proportional to the size of the cross-product space of the domains of both nodes' parents. This is a direct consequence of equation (3.1), where the parent set of each node becomes the union of both parent sets. Although in our discussion above we only showed one state variable X_S and one percept X_E, the expansion of **D** is trivial. In general, more than one state variable may be present, and there may be an n-to-1 interdependency between the two time slices (i.e., a variable in time t may depend on more than one variable in time t - 1). Similarly, more than one percept may exist and may depend on one or more state variables. In such cases, a series of AR steps has to be taken to make the percept depend only on variables in t - 1. Nevertheless, the above simplified illustration suffices to show the evidence reversal algorithm takes into account the observed evidence rather than ignoring it.

4. Compact Representation

In a BN, CPTs have often been represented in a "plain" fashion using arrays. For example, the four CPTs in Figure 3.2 would be represented as two-dimensional arrays. This representation, though simple in nature, is not compact in that it requires an entry for every state of the system. A system is usually reasonably locally structured (Section 2.2.1) so that many states have the same probability values and thus can be compressed. Furthermore, a structured representation allows irrelevant distinctions to be ignored, as we shall see in the following section.

4.1 Tree-Structured CPTs

The tree-structured representation was first introduced in [BODG95]. The use of such representation for CPTs allows the capture of independence of variable assignments.[12] It can identify relevant distinctions in the belief space encoded in the CPT of a node. As a result, values associated with a collection of states may be represented once. Figure 4.1 shows a simple binary tree-structured CPT for a variable A. (For binary variables, our convention is to use left-arrows for "true" and right-arrows for "false".) Each branch determines a partial assignment to the parents of the "owner" variable of the tree (i.e., A in this case) with some parents not mentioned; for example, the left subtree of B is traversed when B is true, without considering C. Each leaf denotes the probability distribution of the variable given conditions consistent with that branch.[13] So in the figure, the rightmost leaf corresponds to

[12] Other representations such as the use of rules representing conditional probability distributions also capture such independence [POOL93].

[13] We only show the probability values for the variable being true because the probability of falsehood can always be computed by one minus the truth probability. This representation with *assumed last values* will be discussed in Section 5.4.

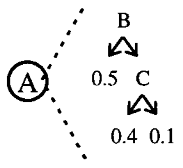
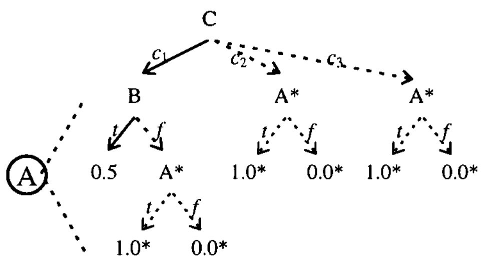


Figure 4.1: A simple tree-structured CPT

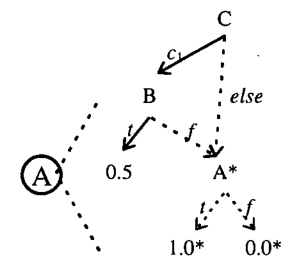
the probability distribution of A given B and C are both false. Normally, if we use a plain CPT representation, we will have to enlist all 2^2 CPT entries. The tree-structured CPT, however, allows us to capture the independence of A on C when B is determined to be true. Thus, only one value, $P(a|b)$, is needed to represent both $P(a|b,c)$ and $P(a|b,\bar{c})$. As a result only three values in total need be specified, spatially enhancing the system.

In [BOGO96], the tree-structured representation is extended by the inclusion of persistence subgraphs. This feature is especially useful in reasoning about actions where persistence relations often arise. Figure 4.2a shows a tree-structured CPT with persistence subgraphs represented by broken arcs and marked nodes, generated automatically if unspecified by the user. From the figure we can observe that these subgraphs can actually be compressed into one. This leads to the graph representation of the CPT as shown in Figure 4.2b. The "else" branch corresponds to all values of C other than c_1.

Although the graph representation is more compact, we would adopt the tree structure in this thesis as performing AR using graphs is more complicated. In Section 5.2 we will see how the tree-structured representation can help in identifying relevant variables to speed up simulation.



(a) Decision Tree



(b) Decision Graph

Figure 4.2: Compact representation of CPT of A

4.2 Structured Arc Reversal

Obviously, the tree structure is capable of representing regularities in the interdependence among variables in the system. Since regularities may prove useful computationally, we would like to employ the tree structure in our simulation process. In Sections 3.2 and 3.3 we have seen how AR can be used to bring simulation "back on track" under plain CPT representation. For large CPTs, however, working with trees may be more efficient in the AR process. This requires some special thoughts, though: while we would like the AR steps to increase the accuracy of simulation, we also want to preserve the tree structures the CPTs had before the reversal so that after AR, regularities can be retained to gain certain computational efficiency. In the following sections we present the operations that will be used along with the *structured arc reversal* (SAR) algorithm. To avoid confusion, we use the terms "t-nodes" and "t-arcs" to represent nodes and arcs in a tree (as opposed to nodes and arcs in a BN) [BFGK96]. We will first give an example to illustrate.

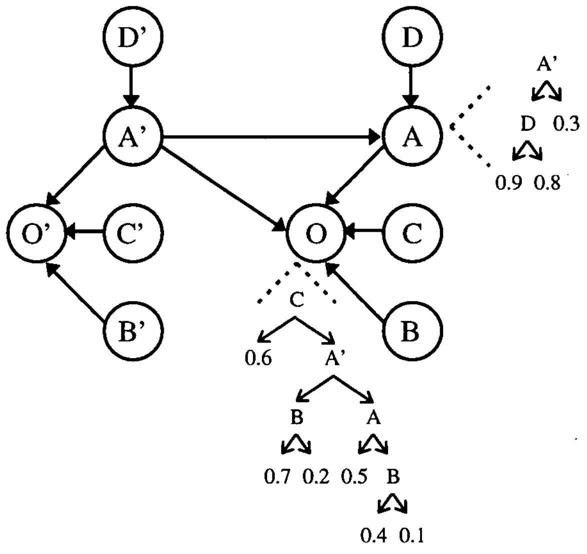


Figure 4.3: An example system

4.2.1 Example

The SAR algorithm is perhaps best demonstrated with an example. Figure 4.3 shows a 2-slice DPN encoding an example system with four system variables (A through D) and one observation (evidence) variable (O). For simplicity, we write the names of the variables directly on the corresponding nodes and show only the CPTs for A and O, assuming B, C, and D have arbitrary distributions. Figure 4.4 and Figure 4.5 demonstrate how we reverse the arc between A and O (i.e., nodes i and j in Section 3.3).

Construction of O*'s CPT*

In Figure 4.4, we construct a new CPT for variable O as follows:

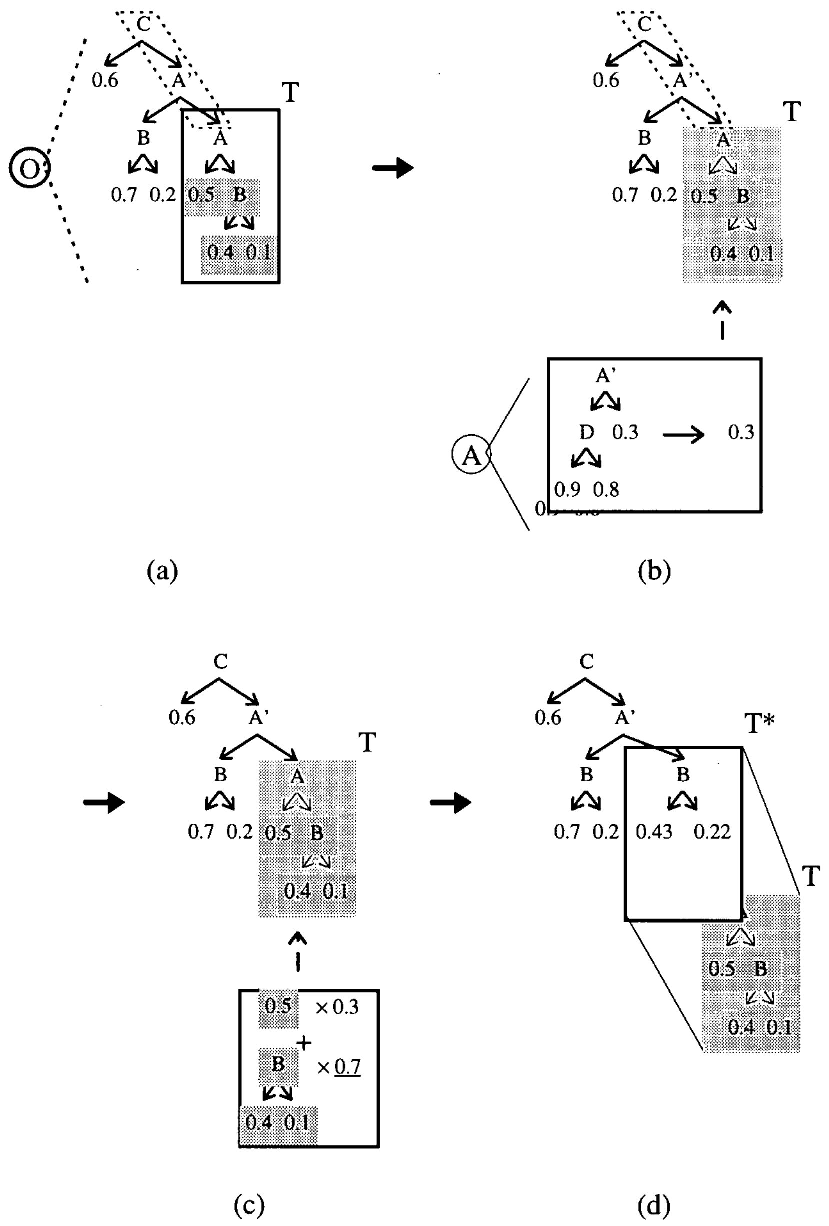


Figure 4.4: Construction of new CPT for node j *(variable* O*)*

1. First we search the old CPT of O for a subtree T with a root node labeled A (diagram (a)), since T will be the only part of the CPT for O that changes; all other parts of the tree remain the same because they do not depend on A and thus will not be affected by AR. In the search process we record all t-nodes and t-arcs traversed in a *path* (enclosed by a dotted line) starting from the root for use in the next step.

2. In diagram (b), we duplicate the CPT of A and carry out *tree reduction* to the copy using information stored in the path above. Intuitively, tree reduction is a process that removes redundant t-nodes in the tree, especially when we are appending a tree to another. The path then serves as an index of t-nodes that are already present in the original tree. In our case, O inherits the parents of A on reversal, so we have to append the CPT of A to that of O to make O depend on the parents of A. By carrying out tree reduction we remove t-nodes considered redundant in the appending process. The path, tree reduction, and the appending of trees will be elaborated in more detail in the next section.

3. Then we use the probabilistic chain rule to merge the child subtrees of T with the reduced copy (diagram (c)). In essence, we are removing the t-node A from the tree (which corresponds to deleting the arc from A to O in the network) and adjusting the probability distributions of A's children according to equation (3.2). The application of the chain rule in trees will be discussed in the next section.

4. The final tree is shown in diagram (d) where T has been saved and then replaced by T^*, the result from step 3 above. T is saved for later use, when we have to refer to values in the old CPT of O to update the CPT of A. T^* is then marked "altered" to indicate an altered portion of the

old CPT of O. (In general, there will be more than one T, and hence T^*.) This will help reduce computational efforts when we apply Bayes's rule later.

A Note: Reduction in Dependency Complexity

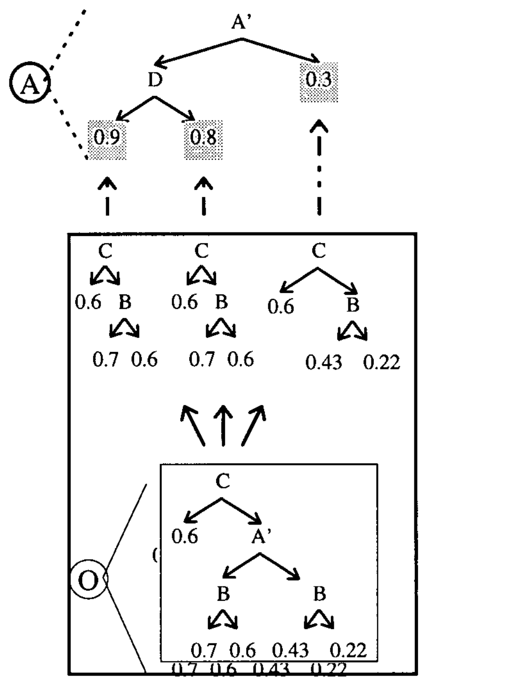
At this point we are able to see that the new CPT for O does not depend on D. This would not have been true if we had used Shacter's algorithm [SHAC86], which states that both nodes inherit each other's parents. This reduction in dependency complexity actually occurred when we reduced the "appending" tree (diagram (b)) by pruning the branch containing the label D. Another place where dependency complexity may be reduced is in diagram (d), when we performed the tree merge. If we have deterministic values, say, $P(a|\overline{a'}) = 1$, then when we perform the merge by the chain rule, a zero (instead of 0.7) will be multiplied to the subtree labeled B, thus removing B's influence when C and A' are both false. If the label is not B but some variable other than A', B, and C, then the influence of that variable can be totally removed from the CPT.

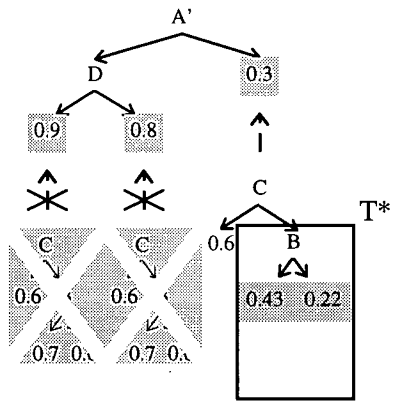
Although reduction in dependency complexity is not a necessary consequence of tree reduction[14], it is still a major saving in time and space when there is a possibility of making fewer steps in tree traversals, especially during simulation in large networks containing hundreds of nodes and tens of thousands of interdependencies.

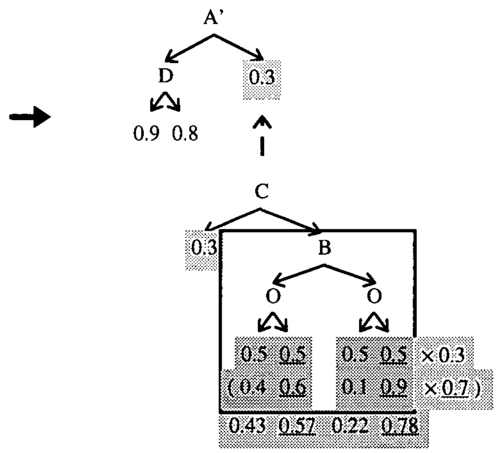
Construction of A*'s CPT*

We go on to update the CPT of A in-place in Figure 4.5.

[14] For example, if we swap the two branches of T_i^{old} so that the D branch is traversed when $A' = false$, then on reduction (diagram (b)) the D branch will remain, and we will not be able to remove D's influence on O.


(a)


(b)


(c)

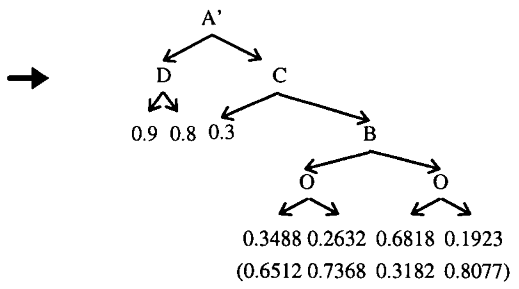

(d)

Figure 4.5: Construction of new CPT for node i *(variable* A*)*

1. We first duplicate the new CPT of O and reduce the copy with respect to each path from root to leaf in A's CPT (diagram (a)). This is to remove the t-nodes considered redundant when we append the new CPT of O to each leaf of the CPT of A. Since there are three leaves in A's CPT, we have three reduced CPTs of O in the diagram.

2. Then we search the reduced copies for subtrees marked "altered" and find that only the third one contains altered parts (T^* in diagram (b)). T^* is inherited from the computation of O's new CPT. It represents the part of the tree that will be extended by the application of Bayes's rule in the next step. The first two reduced copies are discarded, leaving the corresponding leaves in A's CPT unchanged, because the absence of altered parts means the terms $P(x_j | x_{C^{new}(j)})$ and $P(x_j | x_{C^{old}(j)})$ in Bayes's rule equal and hence cancel out each other. We will come to more details in Section 4.2.3.

3. In diagram (c), since the left branch of C is not in T^*, the distribution at the right branch of A' (< 0.3, 0.7 >) is propagated there. T^*, on the other hand, is extended by one level at each leaf with the insertion of a new tree rooted at O. The distributions at the new leaves are computed using Bayes's rule. Note the correspondence between the shaded entries in the diagram: the darkest shade refers to entries in the subtree T in Figure 4.4, the lightest shade refers to entries in A's CPT, and the medium shade refers to entries in T^*. The underlined numbers represent *assumed last values* (Section 5.4) while those in parentheses show the computations that would have taken place if we had not used the assumed-last-value representation for probability distributions.

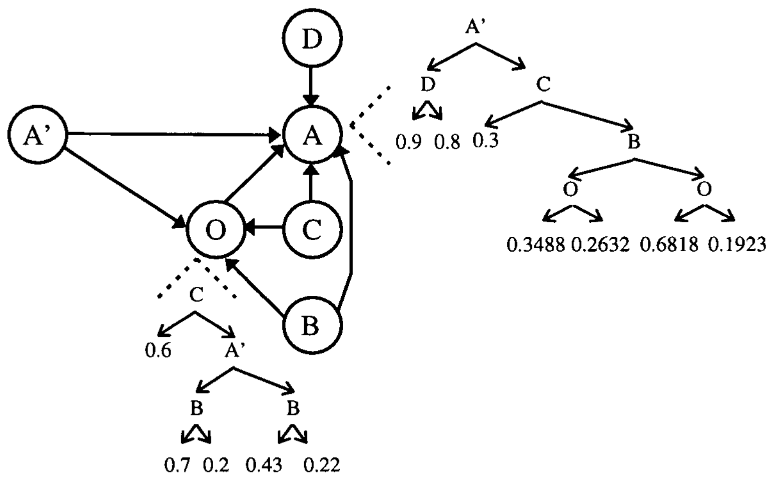


Figure 4.6: Network after reversing the arc between A *and* O

4. The final tree is shown in diagram (d) where the rightmost leaf of A's CPT has been replaced by the result in step 3 above.

The network now becomes as shown in Figure 4.6, where the effect of reduction in dependency complexity is shown by the missing arc between D and O.

4.2.2 Procedure

Having seen our previous example, we are ready to examine the SAR algorithm in more detail. Let T_i and T_j denote the CPTs of arbitrary nodes i and j in G, a given BN. Then Algorithm 4.1 will reverse (i, j) into (j, i) correctly while preserving the tree-structure of both T_i and T_j.

For each node j corresponding to an evidence variable, we reverse the arc between j and each of its parents in $C(j)$. In each such reversal, we build a new CPT for j and update the CPT of i in place. The operations involved are explained below.

Algorithm 4.1: SAR

```
/* SAR: reverse arcs into evidence; each node $i$ in graph $G$ has associated with it a tree-
structured CPT, $T_i$ */
void SAR( Graph G, Evidence E )
{
        Tree $newT_j$, $T$, $T^*$, $Tp$;
        Path $p$;

        for ( each $j \in E$ )
                for ( each $i \in C(j)$) {
                        $newT_j = T_j$;

                        /* Build new CPT for $j$ */
                        for ( each subtree $T$ of $newT_j$ with $root(T) == i$ ) {
                                $Tp = treduce( T_i, path( newT_j, T ))$;
                                $T^* = tchain( T, Tp )$;
                                save $T$ and replace with $T^*$;
                                mark $T^*$ as altered;
                        }

                        /* Build new CPT for $i$ */
                        for ( each leaf $l$ of $T_i$ ) {
                                $Tp = treduce( newT_j, path( T_i, l ))$;
                                $T^* = tbayes( T_j, T_i, Tp )$;
                                replace $l$ with $T^*$;
                        }

                        $T_j = newT_j$;
                        update $C(j)$;
                }
}
```

***Tree Reduction* (treduce)**

In the *SAR* algorithm, the basic operation is the appending of a tree T_1 to another tree T_2, i.e., the replacing of a leaf of T_2 with T_1. Though conceptually straightforward, this operation may make the resulting tree have duplicate node labels. Therefore, a **decision tree reduction** algorithm has to be carried out to delete redundant nodes. Theoretically, the reduction can be done either before or after the appending, but since we have to perform arithmetic operations on trees (as we shall see below), it would be desirable to carry out the reduction early to reduce computational effort. (For example, we reduced the CPT of A in Figure 4.4b so that only the leaf node with distribution <0.3, 0.7> need be considered when we applied the chain rule in Figure 4.4c.) In view of this, whenever appending of trees is required, we choose to reduce the "appending" tree (T_1) before appending it to the "appended" tree (T_2).

As we have seen in the example in the previous section, a tree is reduced with respect to a path (of another tree). The path is an ordered list of label-value pairs. Each pair consists of a t-node label and a number specifying which child subtree to traverse next. Formally, a path is defined as follows:

Definition 4.1: Path

Let *label*(u) denote the node label of some t-node u. Let T be a decision tree and *path*(T, u) be a path of T from root to a certain t-node u in T. Then,

$$path(T, u) \equiv (label_1, value_1, label_2, value_2, \ldots, label_n, value_n),$$

where $label_i$ is the label of the i-th t-node traversed starting from the root of T,
$value_i$ is the value with which the t-arc between $label_i$ and $label_{i+1}$ is labeled,
$label_1 = label(T)$, and
$label_{n+1} = label(u)$.

■

In essence, the path records t-nodes that should be pruned during tree reduction, the process of which can be described by the following definition of a reduced tree[15]:

Definition 4.2: Reduced Tree

Let p be a path of some decision tree T'. Let $child(\ i,\ T\ )$ denote the i-th child of an arbitrary tree T. Further let T be a decision tree with root R and immediate subtrees $T_1, \ldots, T_m$ where $T_j = child(\ j,\ T)$. Then the reduced tree $T(\ p)$ is defined recursively as follows:

$$T(p) = \begin{cases} R \text{ with subtrees } T_j(p) \text{ for all } j \in [1,m], & \text{if } label(R) \text{ is not in } p \\ T_j(p), \text{ where } j = value_i, & \text{if } label(R) = label_i \text{ is in } p \end{cases}$$

■

Put in words, the reduced tree $T(p)$ is a version of the tree T such that every path in T that is "inconsistent" with p is removed.

With these definitions in mind, decision tree reduction is straightforward. The reduction algorithm is shown in Algorithm 4.2.

*Chain Rule (*tchain*)*

Recall in our example that O takes the role of variable X_j while A, X_i. In the following discussions we will use the more general terms X_i and X_j to represent A and O respectively.

[15] [BFGK96] described how a reduced decision tree can be produced under a certain *context* of variable assignments. This is applicable in our SAR algorithm if increased computational efficiency is not a major concern. Since we would like to prune away all redundant t-nodes before applying arithmetic operations on trees, we modified the idea to be used with respect to *paths* instead of *contexts*.

Algorithm 4.2: Tree Reduction

```
/* Reduce a tree T with respect to a path p.  The result is returned as newT. */
Tree treduce( Tree T, Path p )
{
        Tree newT;

        /* Determine if the root label of T exists on p. */
        if ( label( T ) == $label_i$ for some $label_i \in p$ )
                newT = treduce( child( $value_i$, T ), p );
        else {
                label( newT ) = label( T );

                /* Reduce each child of T. */
                for ( i = 0; i < #children( T ); i++ )
                        child( i, newT ) = treduce( child( i, T ), p );
        }

        return newT;
}
```

After we have found a subtree T with root node labeled i in T_j, the CPT of j, and obtained $T_i(p)$ where T_i is the CPT of i and $p = path(T_j, T)$, we apply the tree version of the probabilistic chain rule to T and $T_i(p)$: we replace each leaf l of $T_i(p)$ with the tree produced by "chaining" T with the distribution at l. Referring to Figure 4.4c, we see that this requires multiplying each value in the distribution at l with the corresponding child subtree of T and then "summing up the products". The products are probabilities given values of X_i. Thus, summing up the products means summing up the terms $\mathbf{P}(X_j | x_i, X_{C^{old}(j)\backslash\{i\}})$ for all $x_i \in \Omega_i$. The multiplication step is straightforward because we only have to scale each leaf of the tree by the probability value specified. The summation step, however, is more complicated. If one of the addends is a scalar number, then we just add it to every leaf of the other (tree) addend, in much the same way as we perform scaling. If, however, both addends are trees,

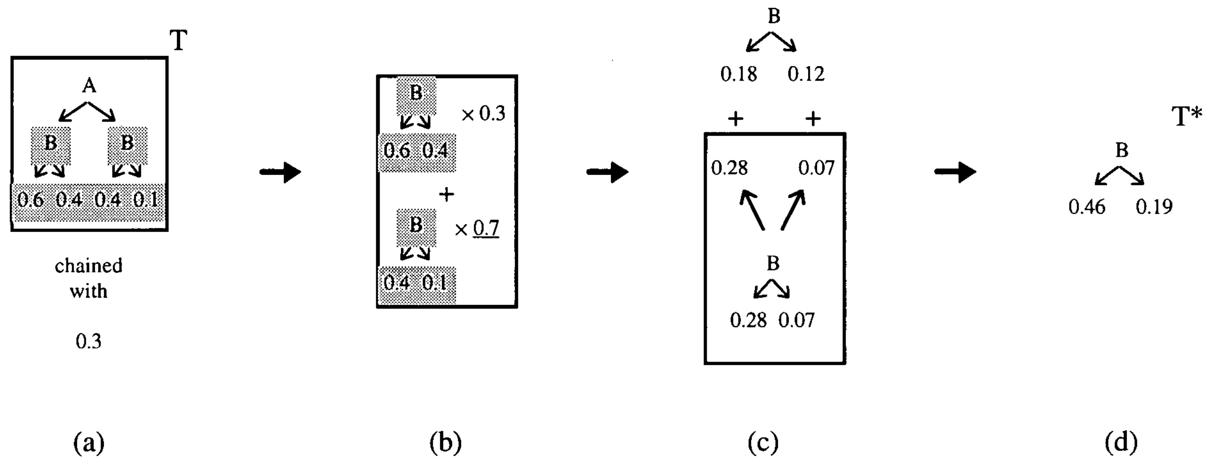


Figure 4.7: Breakdown of tree chaining

then we have to append, with possible reduction, one tree to every leaf of the other. For example, if T was a binary full tree, then the sequence of chaining would become as shown in Figure 4.7, where tree reduction is required in diagram (c).

In any case, the correspondence between the normal chain rule and its tree version,

$$\mathbf{P}(X_j | X_{C^{new}(j)}) = \sum_{x_i \in \Omega_i} \mathbf{P}(X_j | X_{C^{old}(j)\setminus\{i\}}, x_i) P(x_i | x_{C^{old}(i)}), \tag{4.1}$$

can be readily observed: the term on the left corresponds to T^*, the first term in the summation corresponds to T, and the last term represents each probability value in $T_i(p)$. By successively applying the tree chain rule to each subtree T found, a new CPT for j can be constructed.

***Bayes's Rule* (tbayes)**

In the construction of a new CPT for i, we apply Bayes's rule to update each leaf l of T_i using T_j^{old}, l, and $T_j^{new}(p)$, where $p = path(T_i, l)$. (Here we superscript T_j by "old" and "new" to denote the CPT of j before and after AR, respectively. This is not necessary for T_i because we are updating T_i in place.) Due to the complexity of the updating process, we choose to present the idea using an algorithmic way as follows:

Traverse $T_j^{new}(p)$ post-orderly. For each node k thus traversed, if k has been marked "altered", i.e., if the subtree rooted at k is part of the subtree T^* in T_j^{new},

1. for each leaf f of k, replace f with a 2-level, n-leaf tree T' with root(T') = j and $n = \#\Omega_j$. The distributions at the leaves of T' are determined by the **P** version of Bayes's rule, i.e.,

$$\mathbf{P}(X_i | X_{C^{new}(i)}) = \frac{\mathbf{P}(X_j | X_{C^{old}(j)})\mathbf{P}(X_i | X_{C^{old}(i)})}{\mathbf{P}(X_j | X_{C^{new}(j)})}, \qquad (4.2)$$

where $\mathbf{P}(X_j | X_{C^{old}(j)})$ is retrieved from the subtree T saved along with k,

$\mathbf{P}(X_i | X_{C^{old}(i)})$ is the distribution at l, and

$\mathbf{P}(X_j | X_{C^{new}(j)})$ is the distribution at f.

2. else if k is a leaf, replace the distribution at k with the distribution at l.

3. else if no child of k is altered, replace the subtree rooted at k with the distribution at l.

Basically, the entries in T_i are computed as in the normal Bayes's rule. However, since we would like to retain the CPTs' compactness after AR, we have to take special measures to prevent the original tree from being fully expanded. We have found that the distribution at l remains unchanged if there is no altered part in $T_j^{new}(p)$ (Figure 4.5b), and that the unaltered portion of $T_j^{new}(p)$ does not grow but merely inherits the distribution at l (Figure 4.5c). As we have mentioned before, this is due to the canceling out of the terms $\mathbf{P}(X_j | X_{\mathrm{C}^{\mathrm{old}}(j)})$ and $\mathbf{P}(X_j | X_{\mathrm{C}^{\mathrm{new}}(j)})$ in equation (4.2), making the two sides equal. Therefore, we need only expand the altered part of $T_j^{new}(p)$ by applying Bayes's rule. The expansion is done in step 2 above, where $T_j^{new}(p)$ is extended by one level with the distribution at the new leaves computed by Bayes's rule (Figure 4.5c). Now X_j's influence is introduced into the tree and the result can be appended to T_i at l. When all leaves of T_i are adjusted as above, the construction of a new CPT for i is complete. This corresponds to the addition of an arc from node j to node i in the network.

Remarks

Note that we have to update the parent set of j in each AR cycle. This is because during AR new arcs may be introduced into node j. If these arcs come from nodes in time t, then they in turn have to be reversed. For example, D was originally not a parent of O before AR in Figure 4.3. If we switched the two branches of A's CPT, then after reversing the arc between A and O, O would depend on D and there would be an arc from D to O in Figure 4.6. Since D is in time t, the new arc would have to be reversed also. Therefore, it is necessary to keep track of the parent set of O throughout the AR process.

4.2.3 Analysis

Regarding the SAR algorithm developed in the last section, we have made some observations and analysis of its operation.

Constructing $\mathbf{T_j}$

In the construction of a new CPT for node j, we examined how the chain rule is applied in a "tree" context. We note that all parts of T_j^{old} other than T remain the same (Figure 4.4), whether structurally or with respect to both structure and values. The two cases are described as follows:

- If there is a path from root to leave not containing i, then both the structure of the internal nodes on the path and the distribution at the leaf will not change, since the path does not depend on i, and thus will not be affected by the reversal.
- For all other paths, the portion from the root down to but not including i remains the same (i.e., the structure of that portion will not change). The subtrees below i are merged according to the method described, and it is easy to see that the distribution at each of the leaves is in fact a vector of sums of products as implied by equation (4.1), the tree version of the chain rule.

We also observe that when appending a tree to another we have a choice over which one is the "appending" tree and which one the "appended" tree. If we carefully consider the sizes (number of t-nodes) of the trees to be merged, the merge can be done more efficiently.

Consider two binary full trees T_m and T_n with the former having m leaves and the latter n. Then T_m has $2m$-1 nodes while T_n has $2n$-1. Assume m is greater than n. There are two cases:

1. If we append T_m to T_n, then we have to append $2m$-1 nodes from T_m to each of the n leaves of T_n, requiring a total of $n(2m-1) = 2mn-n$ copies.

2. If we append T_n to T_m, then we have to append $2n$-1 nodes from T_n to each of the m leaves of T_m, requiring a total of $m(2n-1) = 2mn-m$ copies.

Since $m > n$, case 2 requires fewer copies than case 1. Hence, we conclude that we should append a smaller tree to a larger one. This conclusion is, nevertheless, more theoretical than practical since it takes $O(\ n\ )$ time to determine the size of an n-node tree (unless the size is stored with each tree). Although the time spent in determining the *initial* tree sizes of CPTs can be absorbed as overhead cost by the system set-up phase when CPTs are built (through console or file input), the presence of tree reductions and appending keeps these sizes varying with time. As a result, just *determining* which tree should be the "appended" tree costs even more than the saving in copying it will bring. Therefore, in practice we will ignore the tree sizes and choose an arbitrary appending order.

Constructing $\mathbf{T}_\mathrm{i}$

In the construction of a new CPT for node i, we can see if T^* and T_j^{new} are disjoint, then everything on $p = path(\ T_i, l\ )$ remains the same (Figure 4.5). In particular, the distribution at l, i.e., $\mathbf{P}(X_i | x_{C^{old}(i)})$ for the $x_{C^{old}(i)}$ observed at l, remains unchanged. This is justified because T^* and $T_j^{new}(\ p)$ are disjoint implies the latter is entirely contained in T_j^{old}, and if we were to apply Bayes's rule (equation (4.2)), then at every leaf node k of $T_j^{new}(\ p\ )$ we would have $\mathbf{P}(X_j | x_{C^{new}(j)}) = \mathbf{P}(X_j | x_{C^{old}(j)})$ for the $x_{C^{new}(j)}$ observed at k; corresponding individual values in these two distributions would cancel out each other, leaving $\mathbf{P}(X_i | x_{\mathrm{C}^{new}(i)}) = \mathbf{P}(X_i | x_{\mathrm{C}^{old}(i)})$ for the $x_{\mathrm{C}^{old}(i)}$ observed at l. Since we are considering the k's with respect to the same l, all k's now have the same distribution and are compressed into one node. As a result, we have not changed anything with respect to l.

For the non-T^* portion of $T_j^{new}(\ p\ )$, a similar argument applies. The difference is that it applies not to the whole tree but only part of it, and thus node collapsing does not propagate to the root level. The T^* portion, on the other hand, is extended by one level with the appending of a T' at each leaf of $T_j^{new}(\ p\ )$. This is necessary to bring the influence of O into the tree, and is the case where the corresponding entries of the first multiplication term and the denominator in the equation do not cancel out.

Reversal Order

In our example in Section 4.2.1 we chose to reverse the arc between A and O first. In general, however, there is no fixed rule in reversal order. For example, given the same system our actual implementation of SAR will reverse the arc between C and O first (see Case 3 in Section 6.2) based on a "first come first serve" basis: it reverses the arc between the first evidence node (in some topological order) and the first parent in time t encountered in its CPT. As it traverses the CPT down to the leaves, the arcs between the node and the parents encountered (if they are in time t) are successively reversed. The next evidence node is then considered, and the process continues until all evidence nodes have been considered.

The difference in reversal order will lead to different resulting CPTs, and thus different network structures after AR. As a general rule, the sooner a node has its arc into the evidence node reversed, the later it will be in topological order in the resulting network. The reason can be traced out as follows: the first reversal will make the first node depend on all (ignoring tree reduction) parents of the evidence. The arcs introduced will then remain unchanged during the rest of the AR process. In the reversal with the second node, the evidence node will have one parent (the first node) fewer than before. So the second node will depend on all parents of the evidence except the first node after AR. This makes the second node come before (or at least in the same position as) the first in topological order. In a similar way, the third node comes before the second, and so on. As a result, when the whole AR process has finished, the first node will be placed last in topological order, preceded by the second node, then the third node, and so on. The evidence node will eventually become the first in topological order since no incoming arc from nodes in time *t* remains, achieving our ultimate goal of making the evidence influence other nodes in time *t*. As an example, we can see in Figure 6.5 that *C* is placed last in topological order (with three incoming arcs from nodes in time *t*), preceded by *A* (three arcs), then *B* (one arc), then *D* (no arc), and then *O* (no arc). If we reverse the arc between *A* and *O* first, then *A* will be the last in topological order (with four arcs in Figure 4.6), and the rest of the node ordering can be inferred from the reversal sequence with the remaining nodes in the aforementioned fashion.

Although different networks may arise from different AR orders, we do not anticipate any major impact imposed by this phenomenon on simulation. The different networks are merely different representations of the same underlying system. We have not established mathematical proofs of the correctness of these representations, but we believe that assuming the existence of a "truly random" number generator the degree of accuracy in probabilistic inference will be the same across representations.

5. Efficient Inference

With our SAR algorithm established, we can incorporate it into Algorithm 3.1 and carry out simulation using this *structured ER* (*SER*) algorithm. However, as practical BNs usually involve hundreds of nodes, performing probabilistic inference in such networks, especially dynamic ones, could be quite a cumbersome task without techniques to improve inference efficiency. We attempt to speed up inference by the following methods.

5.1 Query-directed Approach

Traditional simulation algorithms are straightforward in that they sequentially "flip the coin" for every variable in the system and then compute any posterior probabilities concerned. All variables have to be sampled even though some (probably many) of them may actually not be referenced at all. In the example shown in Figure 2.1, X_2, X_3, and X_4 will never be sampled if we only want to know $P(x_1 \mid x_2, x_3)$, assuming the variables are binary, since nothing influences X_1. We only need to clamp X_2 and X_3 to their given values and make coin flips for X_1.

If we use a query-directed approach, however, we may be able to save on computation (the number of "flips"). In this approach, simulation starts only after a query has been given, and proceeds according to the given information. In particular, only those variables *relevant* to the query are sampled (see next section). In other words, variables that will not affect the query result are ignored. As most systems are locally structured this greatly reduces the number of random numbers generated

Algorithm 5.1: Structured evidence reversal

```
/* SER: evidence reversal using a tree-structured CPT representation */
/* Given: a graph G which is a BN; we assume the CPTs in G to remain constant over time */
void SER( Graph G )
{
        /* Obtain evidence and query */
        Input $E$, $x_E^*$;
        Input $Q$, $x_Q$;

        /* Reverse arcs into evidence using the SAR algorithm */
        SAR( G, E );

        for ( $k = 0$; $k < m$; $k$++ ) {
                /* Initialize score for sample k */
                $Z_k = 1$;

                for ( $t = 0$; $t <= T$; $t$++ ) {
                        Instantiate $X_E$ to $x_E^*(t)$;
                        Sweep through the net, flipping the coin for each variable;

                        /* Update score for sample k */
                        $Z_k = Z_k \prod_{i \in E} P(\hat{x}_i(t) | \hat{x}_{C(i)}(t))$;
                }
        }
}
```

during simulation, thus increasing computational efficiency. Algorithm 5.1 shows our SER algorithm using a query-directed approach.

5.2 Relevant Variables

The notion of relevant variables can be understood at two levels. In the last section we examined this notion based on a *network* level of view, which is adequate when a plain CPT representation is used with the network. When we come to the tree-structured representation, however, a more appropriate view on relevance will be at the *CPT* level. In the following sections we will examine these levels in turn.

5.2.1 The Network Level

Relevance viewed at the network level means the extraction of variables relevant to a query based on the network structure. The following definition helps clarify this. For simplicity's sake, we have left out the time dimension, but it is conceptually straightforward to understand the definition in the time domain.

Definition 5.1: Relevant Variables (Network Level)

Let $V_{net}(Q, E)$ be the set of nodes relevant to the posterior probability $P(x_Q \mid x_E^*)$ at the network level, for some $x_Q \in \Omega_Q$, $Q \subseteq N$, given some $x_E^* \in \Omega_E$, $E \subseteq N$. Then, for all $i \in N$, $i \in V_{net}(Q, E)$ iff

1. $i \in Q$, or
2. $i \in C(j)$ for some $j \in Q \cup E$, or
3. $i \in C(k)$ for some $k \in V_{net}(Q, E)$.

■

This is a recursive definition that accounts for all ancestors of Q and E together with Q itself. The first two conditions can be treated as the base case while the third the recursive step. We do not include the condition $i \in E$ because E is given and need not be "flipped".[16]

[16] In logic sampling E has to be flipped also. Nevertheless, our description will centre around LW, where E is given and need not be flipped.

In LW, this set of nodes, i.e., $V_{net}(Q, E)$, corresponds to all variables that need be considered (and no others) regarding a particular query $P(x_Q \mid x_E^*)$. This is because to obtain $\hat{x}_Q$ we have to instantiate X_Q. Since X_Q is conditioned on its parents, before instantiating X_Q we have to instantiate $X_{C(i)}$ for all $i \in Q$, and to do so we in turn have to instantiate $X_{C(j)}$ for all $j \in C(i)$, which requires the instantiation of $X_{C(k)}$ for all $k \in C(j)$, and so on and so forth until we have at some point instantiated the set of "first generation" ancestors that are not conditioned on any other variable. Hence, $V_{net}(Q, E)$ must at least contain Q and the ancestors of each node in Q. This is expressed in a recursive way by conditions 1 and 3 and half of condition 2 in the above definition.

We also have to account for E in a similar fashion because to compute $P(x_Q \mid x_E^*)$ we have to score the samples based on $P(x_E^* \mid \hat{x}_{CE})$ where $CE = \bigcup_{i \in E} C(i)$. This requires the instantiation of X_{CE} and, in turn, its ancestors. So we see that $V_{net}(Q, E)$ should include also the ancestors of each node in E, and the other half of condition 2 above completes our definition.

The set of nodes $V_{net}(Q, E)$ helps us focus on only those variables relevant to the query, without wasting computational effort to instantiate irrelevant variables as in traditional simulation methods like logic sampling and LW. This reduction in computational complexity is especially significant when X_N, the set of all variables in the system, is large. When applied in DPNs, the extraction of relevant variables may have an even more conspicuous effect in saving the number of flips as Q and E tend to spread out over time so that not all query and evidence nodes need be considered in all time slices. It is conjectured that the number of relevant variables per slice in a DPN is smaller than in a non-temporal BN where all query and evidence nodes are located in the same time slice.

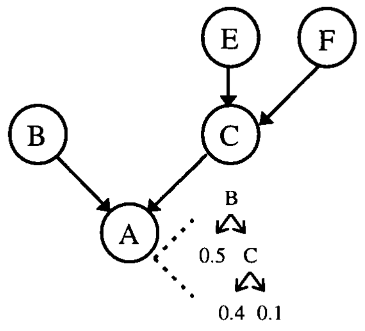


Figure 5.1: Example network

5.2.2 The CPT Level

Though it may be less complicated to analyze relevant variables at the network level, it may not be the most restrictive way possible. In Figure 4.1, for example, if we have instantiated *B* to the value *true* (or, *b*), then *C* will become irrelevant. More specifically, if the entire network is as shown in Figure 5.1, and we have the query *P*(*a* | *b*), then in the instantiation of *A* we can directly retrieve the distribution <0.5, 0.5> from the CPT of *A* and flip the coin accordingly. The right subtree of the root, corresponding to $\overline{b}$, need not be considered at all. As a result, *C* is rendered irrelevant in this context. Furthermore, this irrelevance of *C* makes *C*'s parents *E* and *F* irrelevant also, since *A* does not depend on *E* or *F*. So we actually only need to flip for *A* (*B* is given). With our definition of relevance in the previous section, however, we will have to flip for *C, E,* and *F* because they are the ancestors of *A*, no matter to what value *B* is instantiated. In other words, this *context-specific independence* [BFGK96] cannot be recognized by the above definition. To do so we have to analyze relevant variables at the

CPT level (assuming the use of a tree structure), as the following definition will show. Again, we have omitted the time dimension for simplicity.

Definition 5.2: Relevant Variables (CPT Level)

Let *label*(T) be the root label of tree T, $\hat{x}_i$ be the sampled value of X_i, and T_i denote the CPT of node i for some $i \in$ N. Let $V_{CPT}(\ Q, E\ )$ be the set of nodes relevant to the posterior probability $P(\ x_Q \mid x_E^*\ )$ at the CPT level, for some $x_Q \in \Omega_Q$, $Q \subseteq$ N, given some $x_E^* \in \Omega_E$, $E \subseteq$ N. Define $r(\ T\ )$ such that for any arbitrary tree T,

$$r(T) = \begin{cases} label(T), & \text{if } T \text{ has more than one node} \\ \perp \text{ (null)}, & \text{otherwise} \end{cases}.$$

Further, for some $W \subseteq$ N, let $S(\ W)$ be a set such that for all $k \in$ N, $k \in S(\ W)$ iff

1. $k = r(\ T_j\ )$ for some $j \in W$, or
2. $k = r(\ T_j\ )$ for some $j \in S(\ W)$, or
3. $k = r(\ child(\hat{x}_{r(T_j)}, T_j))$ for some $j \in S(\ W)$.

Then,

$$V_{CPT}(\ Q, E\ ) \equiv Q \cup S(\ Q \cup E\ ).$$

■

This recursive definition captures the set of relevant variables in a *dynamic* fashion, as opposed to the definition of relevance at the network level where the set of relevant variables is statically determined before simulation starts, when the network structure is already available. At the CPT level, relevant variables are determined "on the fly" as simulation proceeds and variables are sampled.

This dynamic nature is encoded in the third condition for membership in S: the sampled value $\hat{x}_{r(T_j)}$ will be available only after $X_{r(T_j)}$ has been instantiated. Furthermore, in different simulation runs $\hat{x}_{r(T_j)}$ may be different, resulting in different sets of relevant variables from simulation to simulation. In the following paragraphs we will give the intuition behind this definition and how its dynamic property helps us to minimize the set of relevant variables chosen.

In order to obtain $\hat{x}_Q$, we have to instantiate X_Q, so just like $V_{net}(Q, E)$, $V_{CPT}(Q, E)$ must at least contain Q so that we can determine whether the query is satisfied in a sample for proper scoring. Since X_Q is conditioned on $X_{C(i)}$ for all $i \in Q$, we start to find out which node belongs to $C(i)$ by looking at the root of T_i. This is reflected in the first condition for membership in S. Naturally, we will then try to traverse the whole tree in some order to find out all parents of i, but this is essentially what we were doing in the search for relevant variables at the network level. Considering that a variable can be instantiated to only one value at any point of time, a wiser way would be not to traverse the whole tree but wait till X_j, where $j = r(T_i)$, has been instantiated and select the appropriate branch to traverse. This requires that $X_{C(j)}$ be instantiated beforehand. Thus we proceed to the root of T_j to search for the parents of j. This is reflected in the second condition for membership in S. (Note that condition 1 has already made $S(Q)$ contain $r(T_i)$ for all $i \in Q$.) Using the same argument, we then proceed to the root of T_k where $k = r(T_j)$, and so on and so forth until we have reached a T_o where $r(T_o) = \perp$ for some $o \in$ N. At this point, since X_o is not conditioned on any other variable, we can instantiate X_o and begin to "bounce back" the recursion. For simplicity, assume we reached T_o from T_k, i.e., $r(T_k) = o$. Because X_o is now instantiated to $\hat{x}_o$, we can select the corresponding subtree, namely

$child(\hat{x}_o, T_k)$, of T_k to continue our search for relevant variables. This is described by condition 3 in the definition, where we proceed to look at $r(child(\hat{x}_o, T_k))$ and search for relevant variables using conditions 2 and 3 recursively. After a path from root to leaf is established in T_k (i.e., all relevant parents of k in the current context are instantiated), we can retrieve the distribution for k at the leaf and instantiate X_k accordingly. Recursion then bounces back to T_j, where we select $child(\hat{x}_k, T_j)$ to continue the search. The process goes on until we have the whole set of X_Q instantiated, at which point the set $S(Q)$ is complete.

For E, we follow the same sequence to build $S(E)$ progressively, starting at $r(T_i)$ for all $i \in E$ until a path is formed from root to leaf in T_i. We can then compute the score of the sample according to the distribution at the leaf and the given evidence.

After $S(Q)$ and $S(E)$ have been completely specified, our task in determining $V_{CPT}(Q, E)$ has virtually finished. The only thing left is to prove $S(Q) \cup S(E) = S(Q \cup E)$, but this is trivial if we hold on to our definition of S and the definition of the union operation in basic set theory.

Summing up, the difference between relevance at the network level and that at the CPT level lies in the dynamic nature of the latter. To determine $V_{CPT}(Q, E)$, we dynamically order variables to be sampled depending on previous sampled values of ancestral nodes so that there is one and only one path from root to leaf traversed in each CPT. Any node not on the path is ignored, and it is this property that makes $V_{CPT}(Q, E) \subseteq V_{net}(Q, E)$.

5.3 Satisfiable Queries

Another saving we can make is to stop sampling query variables whenever any value contradictory to the given query is sampled. This is justified because if any part of a query is not satisfied, then a score of zero will be assigned to the whole query. Therefore, there is no point in sampling further query variables once it has been determined that the sample does not satisfy the given query.

More formally, given some set of query values $x_Q \in \Omega_Q$ for some $Q \subseteq N$, if at any point during simulation a value $\hat{x}_i$, for some $i \in Q$, is sampled such that $\hat{x}_i \neq x_i$, then no sampling will be done for all $j \in Q' \cup S(Q')$ where Q' is the set of unsampled query nodes. Simulation proceeds for all $k \in S(E')$ where E' is the set of evidence nodes not taken into account, and the score for the query, $Z(X_Q = x_Q)$, equals zero.

This is yet another example of dynamically selecting nodes for sampling. It helps further restricting the set of relevant nodes chosen. Note that this technique is applicable only if a *conjunctive* query is posed. For disjunctive queries, we have to modify the technique in such a way that the sampling of query variables is stopped whenever a value $\hat{x}_i$, for some $i \in Q$, is sampled such that $\hat{x}_i = x_i$, because the satisfaction of a disjunctive component means the whole query is satisfied. Our implementation assumes the use of conjunctive queries, but in either case, provided it is not a combination of both, the more complicated the query, the more we can save through this technique.

5.4 Assumed Last Values

By observing that all probability values in a distribution add up to one, we can omit the last value when representing a distribution. The last value can always be found by one minus the sum of the previous values, which is just one arithmetic operation if we use a cumulative distribution representation. (For example, <0.1, 0.2, 0.3, ...> is represented as <0.1, 0.3, 0.6, ...>.) This is especially useful when computationally demanding operations such as arc reversals have to be performed: we can reduce all operations devoted to computing the last entry of any distribution to just one subtraction. We have already seen an example of how computations are saved in Figure 4.5c, where the values and the multiplication in parenthesis represent assumed values and the operation saved.[17]

[17] Another probability representation scheme would be one which relaxes the constraint that the probabilities in a distribution add up to one. We can then use the unnormalized numbers as probability ratios. Although it can let us avoid normalization, this scheme still requires arithmetic operations, which can become computationally intensive when tree components are present, to compute each entry in the distribution. In other words, the arithmetic operations devoted to computing the last entry cannot be saved as in the *assumed-last-value* representation.

6. Implementation and Results

As we have seen in previous chapters, LW does not use observed evidence directly to constrain the generation of the sample. This leads to poor accuracy in the simulation of DPNs. On the other hand, ER takes into account the observed evidence by reversing arcs that go into evidence variables. Thus, answers to queries are really based on the given evidence, resulting in more accurate simulation.

We have also seen that the usual "plain" CPT representation (using arrays) requires the specification of probability values for all combinations of values of a node's parents. The tree-structured representation, however, not only helps reduce the number of CPT entries required but also saves on the number of "coin flips" (random number generation) to a large extent in practical systems.

In order to confirm the aforementioned ideas, we implemented the "plain" version and the tree version of both the LW and the ER algorithms. A series of experiments was then carried out on a 486-DX66 machine. Hereafter we will use the abbreviations ER, LW, SER, and SLW to refer respectively to plain ER, plain LW, structured ER, and structured LW.

6.1 Experiments

The aim of our experiments is to show that:

- ER helps speed up convergence of answers to queries, as compared to LW;

Table 6.1: Simulation performance under different settings

	Plain		Tree	
Measurement	LW	ER	LW	ER
#distributions avg #flips avg time	1, 2, 4, 5	1, 3	2	3, 4, 5

- the tree-structured CPT representation can reduce the number of flips during simulation, thus increasing computational efficiency;
- the tree-structured CPT representation can reduce the number of CPT entries by discarding irrelevant distinctions that would be present in a "plain" representation.

Accordingly, we arranged our experiments in different settings as shown in Table 6.1. The small numbers refer to the cases described below. For each case, we compared simulation performance under two of the settings using a sample size of 1000. (For example, in Case One we compared the performance of LW and ER, as shown by the two small "ones" in the "LW" and "ER" columns under "Plain" in the table.) Analysis was then made, assuming the existence of an unbiased, or "truly random" number generator that accurately produces states of the network according to their true probabilities, contingent on the given evidence.

In the first row of the table, we measured the number of probability distributions required. For the plain representation, we obtained the number by summing up the total number of rows in all CPTs in the system. For the tree representation, the number was computed by counting the number of leaf nodes in all the CPTs. (For example, the table for variable O in Figure 6.3a contains four distributions while the tree for *O* in Figure 6.3b has three.) In the ER and SER settings, this number reflects the number of distributions generated *after* AR had been performed.

In the second row, the average number of flips over all samples was recorded. This shows the average number of random numbers generated in a sample and corresponds to the set of *relevant variables* (Section 5.2) actually simulated. Again, in the ER and SER settings, the number was recorded *after* AR had been performed.

In the last row we recorded the execution time averaged over all samples. We included this test because we observe that most languages are developed with arrays (rather than trees) as a central component; if we perform our simulation using a tree-structured representation, we might incur more runtime overhead than when using arrays. By examining the average execution time we can see whether the structured representation (or AR) has any influence on execution speed, and if so, to what extent it might be.

In each of the five cases below, we ran the simulation 100 times using a query-directed approach. For cases 1, 4, and 5 we analyzed convergence performance based on the variance of the probability values obtained across all runs and plotted variance against the number of samples in graphs. The variance computed is based on the total population, i.e.,

$$\text{Variance} = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_i\right)^2,$$

where x_i is the probability value computed for the query variable during the i-th run, and

n is the total number of runs.

Convergence is reached when the variance falls below an epsilon value computed by

$$\text{Epsilon} = 10^{2\lfloor \log(\bar{x}) \rfloor},$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

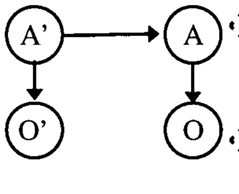
6.2 Results and Analysis

Case 1

For the network shown in Figure 6.1a, we made 100 simulation runs using the LW and ER algorithms to find the posterior probability

$$P(A(0) = A(10) = A(20) = A(30) = A(40) = A(50) = a_1 | O(0) = O(10) = O(20) = O(30) = O(40) = O(50) = o_1).$$

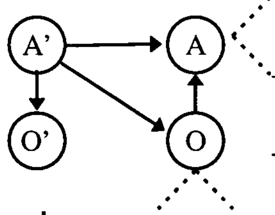
(Underlined entries represent "assumed" values, i.e., the ones that can be computed by subtracting corresponding cumulative probabilities from one.) Performance results are shown in Table 6.2. We see that in general ER doubled all the measurements we made as compared to LW. The increase in the number of distributions (and thus CPT entries) can be projected from our discussion in Section 3.3. In this case, the CPT of A increased from three rows to nine rows because after AR O becomes a parent of A and the value set of O has a cardinality of three. The average number of flips per sample also nearly doubled because A is now influenced by one more variable, namely O; for time slices where no



	A		
A'	a_1	a_2	a_3
a_1	.4	.3	.3
a_2	.3	.4	.3
a_3	.3	.3	.4

	O		
A	o_1	o_2	o_3
a_1	.9	.05	.05
a_2	.05	.9	.05
a_3	.05	.05	.9

(a) Before AR



		A		
A'	O	a_1	a_2	a_3
a_1	o_1	.9231	.0385	.0384
	o_2	.0656	.8852	.0492
	o_3	.0656	.0492	.8852
a_2	o_1	.8852	.0656	.0492
	o_2	.0385	.9231	.0384
	o_3	.0492	.0656	.8852
a_3	o_1	.8852	.0492	.0656
	o_2	.0492	.8852	.0656
	o_3	.0385	.0385	.9230

	O		
A'	o_1	o_2	o_3
a_1	.39	.305	.305
a_2	.305	.39	.305
a_3	.305	.305	.39

(b) After AR

Figure 6.1: Network — Case 1

evidence value was given, we had to make two flips instead of one to obtain the value of A. In turn, this resulted in a nearly 100% increase in the average execution time per sample.

As for convergence, we plotted in Figure 6.2 the variance of the probability values obtained versus the number of samples. An epsilon value of 0.01 was used. As shown, simulation using ER

Table 6.2: Performance — Case 1

Measurement	LW	ER
#distributions	6	12
avg #flips	51	96
avg time	8.4e-4	1.53e-3

converged very early, at a sample size of about 50 (variance $\approx$ 6e-3). Using LW, however, simulation did not converge even using 1000 samples (variance $\approx$ 0.08). This shows that convergence is at least 20 times faster in the ER case where AR played a major role in bringing simulation back on track.[18]

The network used is a DPN where the state evolution model is weak, as can be seen from the CPT of variable A where states evolve quite randomly. Variable O can be thought of as a sensor of A. Although this sensor has an error of only 0.1 in tracking the current state of A, LW ignored this evidence and let the system proceed according to its weak state evolution model, disagreeing with the evidence most of the time. In each run, therefore, the weighting process assigned extremely low weights to almost all the samples. On the other hand, a very small number of samples closest to the true state were assigned relatively high scores. These samples imposed a disproportionately large influence on the sample score distribution and dominated in the calculation of the posterior in concern. Most of the samples were therefore effectively ignored, resulting in large estimation errors. For this example, the score of a sample varies from 0.05^6 = 1.6e-8 (when all six A's have either a_2 or a_3 as values) to $0.9^6 = 0.53$ (when all six A's have the value a_1). Since these scores fluctuate by factors of as large as 10^7, the probability computed in one run might be very different from that in another. The

[18] In [FUCH90], simulation results showed that by reversing arcs into evidence nodes convergence was at least 100 times faster than without evidential integration (the non-temporal counterpart of ER).

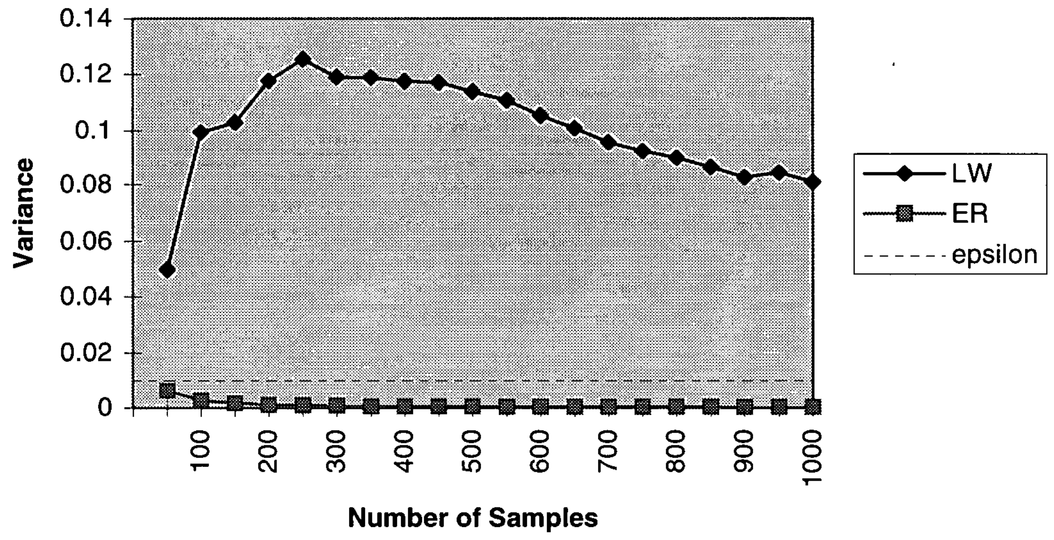


Figure 6.2: Convergence behavior — Case 1

resulting variance is so great that the posterior obtained cannot be considered as accurate. Unless significantly more samples are taken, LW fails in producing a reliable answer when systems with a weak state evolution model are simulated.

ER helps bring simulation closer to reality by using the observed evidence to influence state variables. Figure 6.1b shows the network structure and the CPTs after AR. Now sample scores do not vary to a great extent: assuming random initial states, the score of a sample varies from $1/3 \times 0.305^5 = 8.8e\text{-}4$ to $1/3 \times 0.39^5 = 0.003$ after arcs have been reversed (including time slice 0). Moreover, variable A now has a chance of about 0.9 of having the value a_1 given $O = o_1$, as shown by the highlighted entries in the figure. As a result, not only is the degree of accuracy increased by the reduction in vari-

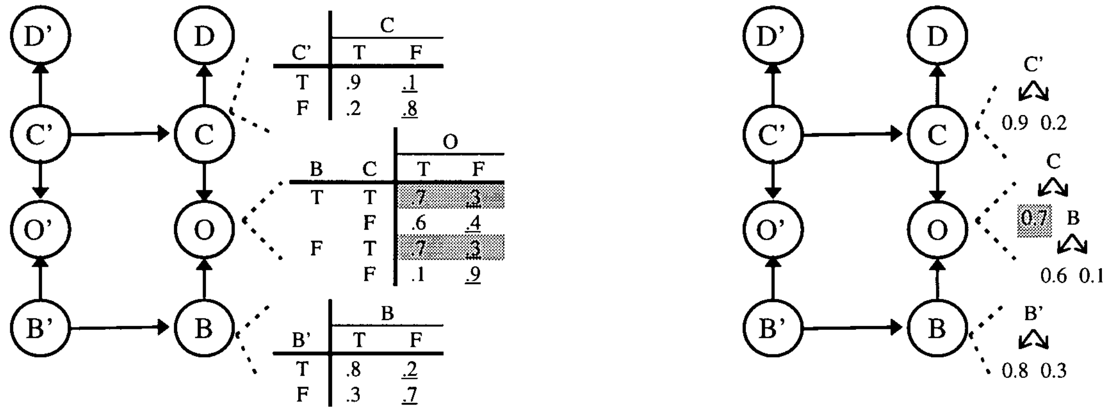


Figure 6.3: Network — Case 2

ance of the scores obtained but the "degree of realness" of simulation is also enhanced by the generation of more samples close to the true state of A given values of O.

Case 2

We compared the performance of LW and SLW using the networks shown in Figure 6.3. (We have left out the CPT for D because D will never be sampled given our query below.) Performance results are shown in Table 6.3.

First, we managed to save one probability distribution by using the tree-structured representation. (Note the compression of the two shaded entries into one.) Then we started simulation using the query $P(C(5) = T | O(5) = T)$, assuming both variables are binary. The average number of flips per sample using SLW is only two-third that using LW. This is mainly due to two places where flips were saved. The first is the saving of flips for variable D; since it is not the ancestor of C or O, it is not a

Table 6.3: Performance — Case 2

Measurement	LW	SLW
#distributions	10	9
avg #flips	12	8.174
avg time[19]	8.18e-4	7.27e-4

relevant variable and was not sampled at all in the simulation. The second saving lies in the CPT of O. We notice that once C is true, there is no need to flip for B. So, for this query, we can generate all values of $C(0)$ to $C(5)$ and then, if $C(5)$ is true, generate $O(5)$ and these are all the variables we need. $B(0)$ to $B(5)$ will only be generated if $C(5)$ is false. Thus we see that if a variable depends on the previous value of itself, and the variable need not be generated in a certain time slice t, then in all slices up to and including t no sampling for the variable is necessary. When traced from time t (i.e., five in this case) back to time zero, this chain of savings can increase computational efficiency significantly. Moreover, since a variable may depend on more than one variable in the previous time slice, a saving in time t will mean *several* chains of savings from time t back to time 0.

In Figure 6.4 we show an excerpt from the execution results of SLW. The blank entries correspond to the $B(t)$'s skipped during simulation. As can be seen, as many as 50% of the total number of relevant variables (perceived at the network level) were skipped for most samples. For the case where a plain CPT representation was used, however, no such savings could be made. This explains why the average execution time per sample is longer using LW than using SLW, and proves that our worry that using a tree-structured representation may slow down execution is not necessary.

[19] Since we turned on the "verbose" option in this experiment in order to produce excerpts from execution results, these values are actually greater than they should be when the verbose option is off. Nonetheless, as our comparison between LW and SLW is local to this experiment only, this increase in value does not matter in a relative sense.

```
Run 1:

      |                                     Variable                                      |
      | ----------------------------------------------------------------------------------| Z(sample) Z(query)
Round | C(0)   C(1)   C(2)   C(3)   C(4)   C(5)*  B(0)   B(1)   B(2)   B(3)   B(4)   B(5) |
------+-----------------------------------------------------------------------------------+-------------------
    1 |   T      T      T      T      T      T                                            |   0.7000    0.7000
    2 |   T      T      T      T      T      F      T      T      T      T      T      T  |   0.6000    0.0000
    3 |   F      F      F      F      T      T                                            |   0.7000    0.7000
    4 |   F      F      F      F      F      T                                            |   0.7000    0.7000
    5 |   T      T      T      T      T      T                                            |   0.7000    0.7000
    6 |   T      T      T      T      T      T                                            |   0.7000    0.7000
    7 |   F      F      F      F      T      T                                            |   0.7000    0.7000
    8 |   T      T      T      T      T      T                                            |   0.7000    0.7000
    9 |   F      F      F      F      F      F      T      F      T      T      T      T  |   0.6000    0.0000
   10 |   T      T      T      T      T      T                                            |   0.7000    0.7000
   11 |   F      F      F      F      F      T                                            |   0.7000    0.7000
   12 |   F      T      T      T      F      F      T      T      T      T      T      T  |   0.6000    0.0000
   13 |   F      F      F      F      T      T                                            |   0.7000    0.7000
   14 |   T      F      F      F      F      F      T      T      T      T      T      T  |   0.6000    0.0000
   15 |   T      T      T      T      T      T                                            |   0.7000    0.7000
   16 |   F      F      F      F      T      T                                            |   0.7000    0.7000
   17 |   F      F      F      F      F      T                                            |   0.7000    0.7000
   18 |   F      F      F      T      T      T                                            |   0.7000    0.7000
   19 |   T      T      T      T      T      T                                            |   0.7000    0.7000
   20 |   T      T      T      T      T      T                                            |   0.7000    0.7000
   21 |   T      T      T      F      F      F      F      F      T      T      F      T  |   0.6000    0.0000
   22 |   T      T      T      T      F      F      F      F      F      T      T      T  |   0.6000    0.0000
   23 |   T      T      T      T      T      T                                            |   0.7000    0.7000
   24 |   T      T      T      T      T      T                                            |   0.7000    0.7000
   25 |   T      T      T      T      T      T                                            |   0.7000    0.7000
                                      :                           :
                                      :                           :
```

Figure 6.4: Excerpt from execution results of SLW — Case 2

We do not examine convergence behavior in this experiment because the difference in CPT representations does not affect convergence and there will be no difference between LW and SLW in this aspect.

Case 3

Using the system shown in Figure 4.3, we carried out simulation with both the ER and the SER algorithms. Since our implementation reverses arcs on a "first come first serve" basis, it chose to reverse the arc between variables O and C first because C is the parent of O first encountered. The resulting system is as shown in Figure 6.5. The query posed was $P(C(5) = T|O(5) = T)$. Performance results are shown in Table 6.4. Again, we do not investigate convergence behavior for this case for the aforementioned reason.

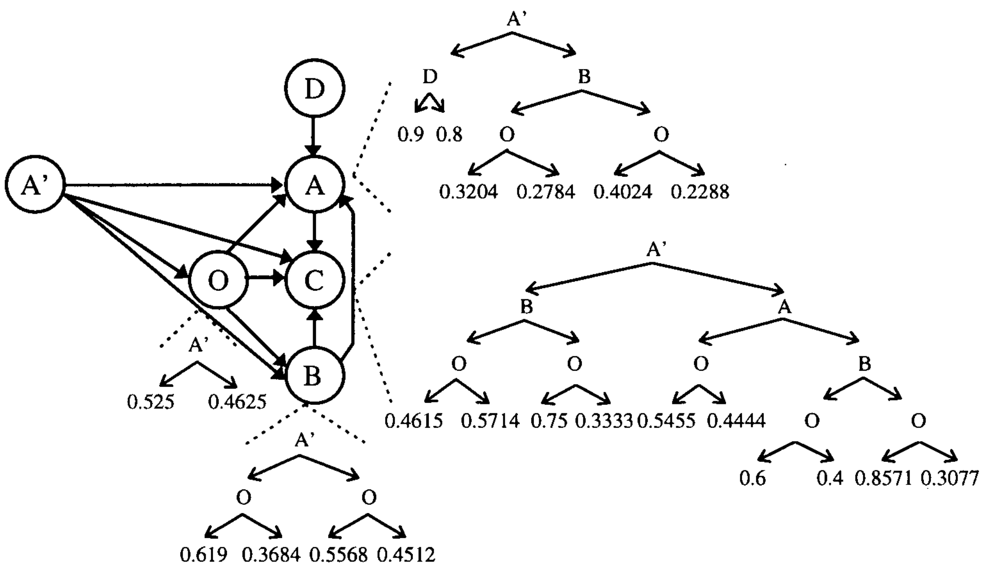


Figure 6.5: Network after SAR — Case 3

As expected, the plain ER algorithm led to full CPTs (with 62 rows in total) after AR while our SER algorithm managed to shrink the CPTs to contain merely 23 leaves. (Note that the CPT of D remained unchanged: it retained its original random distribution because the influence of D on O that would have been introduced during AR between O and A was removed in SAR (Section 4.2.1).) For large networks, this reduction in space complexity (even with internal nodes counted) is an invaluable asset. SER also generated a much lower average number of flips of 15.87 per sample[20], reflecting the ability of the tree-structured representation to capture variable independence (or, more precisely, *context-specific independence* (CSI) [BFGK96]) during simulation. An excerpt from the execution results

[20] The total number of relevant variables perceived in the network level was 24 after SAR. This is slightly different from that (27) in ER due to the difference in the orders of arcs reversed. Our ER implementation also reverses arcs on an FCFS basis, but since the order of parent specification for a node does not matter in a plain CPT representation, we chose to specify the parents of a node in alphabetical order for consistency. Thus, variable A was encountered first in ER while SER first came across variable C. As a result, two different network structures were produced.

Table 6.4: Performance — Case 3

Measurement	ER	SER
#distributions	62	23
avg #flips	27	15.87
avg time[21]	1.55e-3	1.55e-3

of SER is shown in Figure 6.6. As many as 10 variables could be skipped before the requested posterior was computed in each sample, though the average execution time per sample was not reduced due to the overhead in AR and the relatively small size of the system. For a larger, more practical network, it is expected this AR overhead will have a much smaller impact on execution time.

We also observe in the figure that variable *D* in time slice 5 never entered into consideration during simulation, as shown by the blank column under the underlined heading "D(5)" near the right end of the figure. The reason can be found by examining the set of variables relevant to the posed query. Starting (in time 5) at the root of the CPT of variable *C* we encounter *A'* (Figure 6.5), which means *A*(4) must be flipped before other parents of *C*(5). If *A*(4) turns out to be true, then *C*(5) only depends on *B*(5). From the CPT of *B* we know that variable *B* does not depend on *D*. Therefore, if *A*(4) is true, then *D*(5) will never be considered. On the other hand, if *A*(4) is false, then the next flip has to go for *A*(5). Referring to the CPT of *A*, given that *A'* is false, *D* is skipped. The remaining parents are *B* and *O*, neither of which depends on *D*. So, if *A*(4) is false, *D*(5) will also not be flipped. Hence, we conclude that *C*(5) in no circumstance depends on *D*(5), explaining why the column in concern is blank.

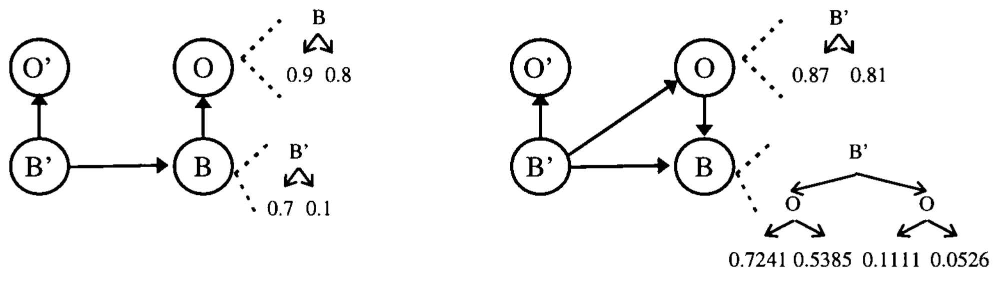
[21] We again turned on the verbose option in this experiment to obtain excerpts from execution results.

```
                                                                  :                               :
                                                                  :                               :
Run 11:                                                           :                               :

      |                                                             Variable                                                                    |
      |-------------------------------------------------------------------------------------------------------------------------------------------| Z(sample) Z(query)
Round | O(0)  D(0)  B(0)  A(0)  O(1)  D(1)  B(1)  A(1)  O(2)  D(2)  B(2)  A(2)  O(3)  D(3)  B(3)  A(3)  O(4)  D(4)  B(4)  A(4)  D(5)  B(5)  A(5)  C(5)*|
------+------------------------------------------------------------------------------------------------------------------------------------------+--------------------
    1 |  T     F     T     F     T           F     T           T           T           T           T           F           F           F     F     T   |  0.4625    0.4625
    2 |  T     T     T     F     T           F     T           F           T           T           T           T           T           T           F   |  0.5250    0.0000
    3 |  F     F     T     T           F           T           T           T           F           T           F           T           T           F   |  0.4750    0.0000
    4 |  F     F     F     T           F           T           T           T           T           T           T           F           F     T     F   |  0.5375    0.0000
    5 |  T     T     T     T           T           F     F           T     F     T           T     F     T           F     F           T     T     F   |  0.4625    0.0000
    6 |  F     F     T     T           T           T           T           T           T           T           F           T           T           T   |  0.4750    0.4750
    7 |  F     F     F     F     T           T     T           T           T           T           T           F           T           F           T   |  0.5250    0.5250
    8 |  T     T     T     F     F           T     F     F           F     F     T           T     T           T           F           T     F     T   |  0.4625    0.4625
    9 |  F     F     F     F     T           F     F     F           F     F     F           T     T           F           F           T     F     T   |  0.5375    0.5375
   10 |  F     F     T     F     F           F     T           T           T           T           T           T           T           F           T   |  0.4750    0.4750
   11 |  T     F     F     T           T           T           T           T           F           T           F           T           T           F   |  0.5250    0.0000
   12 |  T     T     T     T           T           T           T           T           F           T           F           T           T           T   |  0.5250    0.5250
   13 |  F     F     F     F     T           T     F     F           F     F     T           T     F     T           F     F           F     F     T   |  0.4625    0.4625
   14 |  T     T     T     F     F           T     F     T           T     F     F           F     F     T           T     T           T           T   |  0.5250    0.5250
   15 |  F     F     T     F     F           F     T           T           T           T           F     T           T     F           F     T     F   |  0.4625    0.0000
   16 |  T     F     T     T           T           T           T           T           T           T           T           T           T           F   |  0.5250    0.0000
   17 |  T     F     T     F     F           T     T           T           T           T           F     F           T     F           T     F     F   |  0.5375    0.0000
   18 |  T     T     T     T           F           T           F           T           T           T           T           T           T           T   |  0.5250    0.5250
   19 |  F     F     T     T           T           T           F           T           T           T           T           T           T           F   |  0.4750    0.0000
   20 |  F     F     F     T           T           T           T           T           T           T           T           T           F           T   |  0.4750    0.4750
   21 |  T     F     F     T           T           T           T           T           F           F     T           F     F           F     F     T   |  0.4625    0.4625
   22 |  F     F     F     T           T           T           T           T           T           T           F           T           F           T   |  0.4750    0.4750
   23 |  T     T     F     T           F           F     T           F     F     T           T     T           F           T           T           T   |  0.5250    0.5250
   24 |  T     F     F     F     T           F     T           F           T           F           T           F           T           F           T   |  0.5250    0.5250
   25 |  T     T     T     T           T           T           T           T           T           T           T           T           T           T   |  0.5250    0.5250
                                                                  :                               :
                                                                  :                               :
                                                                  :                               :
  976 |  T     T     T     T           T           T           T           T           F           F     F           F     T           F           F   |  0.4750    0.0000
  977 |  F     T     F     F     T           F     F     F           T     F     T           T     T           T           F           T     F     T   |  0:4625    0.4625
  978 |  F     F     T     T           T           T           F           T           F           T           F           T           F           T   |  0.4750    0.4750
  979 |  F     F     T     F     T           T     F     T           T     F     F           F     F     T           F     F           T     F     T   |  0.4625    0.4625
  980 |  T     T     F     T           F           T           F           T           T           F     F           T     T           F           T   |  0.4750    0.4750
  981 |  T     T     T     F     F           T     F     T           T     F     T           F     F     F           T     F           T     F     T   |  0.5375    0.5375
  982 |  F     T     F     T           F           T           T           T           T           T           T           T           F           T   |  0.4750    0.4750
  983 |  F     T     F     F     T           F     T           F           T           F           F     T           F     F           F     F     T   |  0.4625    0.4625
  984 |  F     F     F     F     F           F     T           T           F     T           T     F     T           T     F           T     F     T   |  0.4625    0.4625
  985 |  F     T     F     F     T           T     T           F           T           F           T           F           T           F           T   |  0.5250    0.5250
  986 |  T     T     T     T           T           T           F           F     F           T     F     F           F     F           T     F     F   |  0.5375    0.0000
  987 |  F     T     F     T           F           T           F           F     T           F     T           T           T           F           F   |  0.5250    0.0000
  988 |  T     T     T     T           F           F     T           T     F     F           T     F     F           F     T           F           F   |  0.4750    0.0000
  989 |  F     T     F     T           T           T           T           T           T           T           T           T           F           T   |  0.4750    0.4750
  990 |  T     F     T     T           F           T           F           T           F           T           F           T           T           F   |  0.5250    0.0000
  991 |  T     T     T     T           F           T           T           T           F           T           F           T           F           T   |  0.5250    0.5250
  992 |  T     T     F     F     T           F     F     T           F     F     T           F     F     F           F     F           F     T     F   |  0.5375    0.0000
  993 |  F     F     T     F     T           F     T           F           T           T           T           F           F           F     F     T   |  0.4625    0.4625
  994 |  F     F     T     F     F           F     F     T           T     F     T           F     F     F           F     T           T           F   |  0.4750    0.0000
  995 |  F     F     F     T           F           T           F           F     T           F     T           T           T           F           T   |  0.5250    0.5250
  996 |  T     T     F     F     T           T     F     T           T     F     T           F     T           T           T           T           T   |  0.5250    0.5250
  997 |  F     T     T     T           T           T           T           T           F           T           T           T           T           T   |  0.4750    0.4750
  998 |  T     F     T     T           F           T           T           T           T           T           F           T           F           T   |  0.5250    0.5250
  999 |  T     F     T     T           F           T           F           T           F           T           T           F           T     T     T   |  0.4625    0.4625
 1000 |  F     T     F     T           F           T           F           F     T           F     T           F           F           F     T     F   |  0.4625    0.0000
                                                                  :                               :
                                                                  :                               :
                                                                  :                               :
```

Figure 6.6: Excerpt from execution results of SER — Case 3

In this experiment (and the previous one) we can see the tree structure lends itself nicely to performing simulation in DPNs. Despite the relatively small size of our networks, the savings in the number of flips were tremendous. To a large extent it is this ability to capture CSI and the resulting efficiency enhancement that motivates the use of a tree-structured CPT representation, especially when hundreds of variables are present in practical systems.



(a) Before SAR (b) After SAR

Figure 6.7: Network — Case 4

Case 4

Though AR and the tree-structured representation are useful, there are cases where neither of these techniques can help. One such system is shown in Figure 6.7a. We ran LW[22] and SER using this system with the query

$$P(B(0)=B(2)=B(4)=B(6)=B(8)=B(10)=T \mid O(0)=O(2)=O(4)=O(6)=O(8)=O(10)=T).$$

The tree structure did not buy us anything before AR, since all CPTs were full trees. It did not make CPTs more compact after AR either (Figure 6.7b). In Table 6.5 we see that SER apparently managed to save some flips (0.96 out of 16). This saving, however, was due to the incorporation of our technique where sampling of query variables is stopped on insatisfiability of any component of the query[23] (Section 5.3). In other words, the tree structure had no contribution to the saving of flips in this experiment.

[22] We show only the tree version of the CPTs for convenience. The plain version can be obtained in a straightforward manner.

[23] Query variables will still be sampled if they are relevant to the given evidence.

Table 6.5: Performance — Case 4

Measurement	LW	SER
#distributions	4	6
avg #flips	11	15.04
avg time	1.82e-4	2.73e-4

In Figure 6.8, we can see AR did not speed up convergence significantly in this case because the sample score before AR was already quite stable (0.9 versus 0.8) across samples and only a little more stable (0.87 versus 0.81) after AR. In addition, ten time slices is not sufficient, given the strength of the state evolution model, to allow simulation to diverge "too far" from reality. Therefore, simulation using either of the algorithms converged to an epsilon of 1e-4 at about the same number of samples. (In fact, SER needed even more samples (670) than LW (420) in this case.) The average execution time per sample also increased after SAR due to the increase in the number of arcs and thus the average number of flips. So, in this case, instead of going into all the troubles to reverse the arcs, we would prefer using the non-AR version. Nonetheless, systems similar to this one are not common in practice. Real systems are usually much larger so that a larger extent of local structures can be exploited. At the same time, the errors of the sensors are normally much smaller (e.g., $P(O = T \mid B = F)$ $<<$ 0.8) so that the effect of AR is much more conspicuous.

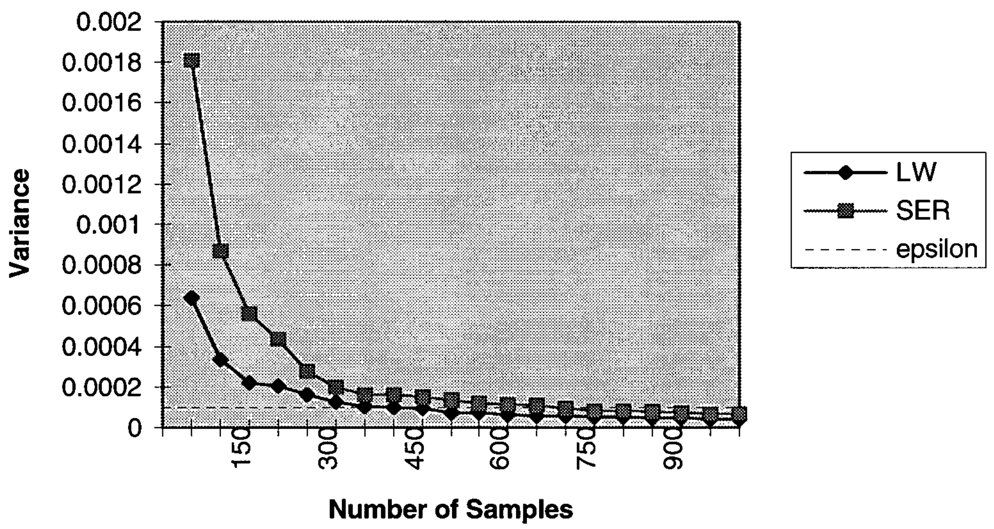


Figure 6.8: Convergence behavior — Case 4

Case 5

In an attempt to explore the effects of SAR on probabilistic inference in a larger network, we doubled the size of the network in Case 3 and compared the performance of LW and SER. Figure 6.9 shows the network used in the experiment. As before, variable *A* depends on its previous value. All other variables, i.e., *B* - *E*, *O*, *W* - *Z*, only depend on variables within the current time slice. Source variables (*B* - *D*, *W* - *Z*) are assumed to have random distributions and their CPTs are not shown.

After posing the query

$$P(D(0) = D(10) = D(20) = D(30) = D(40) = D(50) = F \mid O(0) = O(10) = O(20) = O(30) = O(40) = O(50) = F)$$

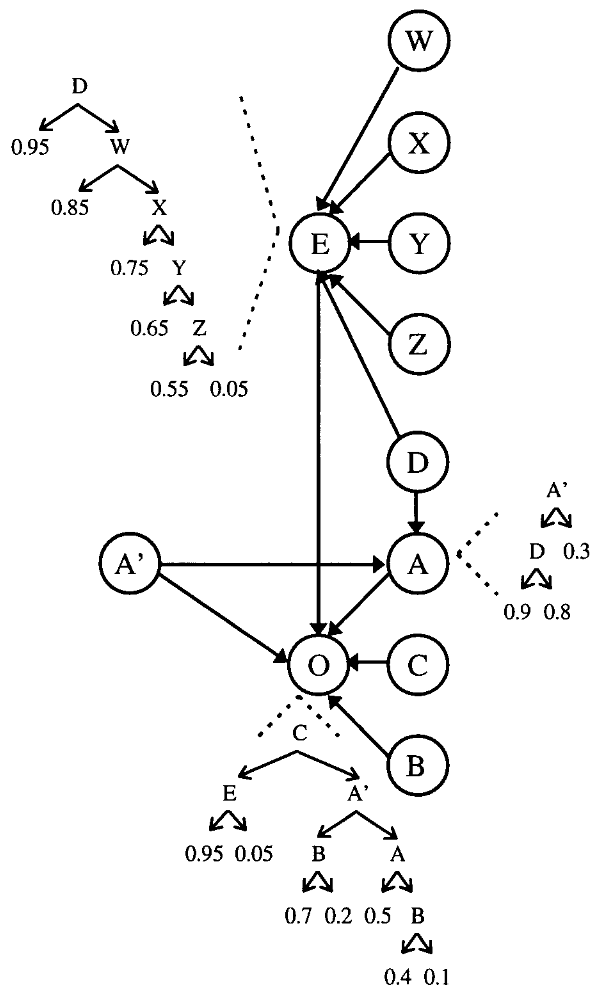


Figure 6.9: Network — Case 5

we ran both the LW and the SER algorithms 100 times. The epsilon used was 1e-4, since the posterior probability was of the order 0.01. Performance results are shown in Table 6.6.

From the table we notice that SER more than doubled the number of probability distributions in the resulting system after SAR. This can be explained by the vast number of parents of both the

Table 6.6: Performance — Case 5

Measurement	LW	SER
#distributions	75	192
avg #flips	144	363.1
avg time	0.0031	0.00979

evidence variable O and the state variable E in the original network (where both variables had 5 parents). Since there was a directed arc from E to O, this arc had to be reversed, which means O had to inherit all five of E's parents. This in turn called for further reversal of arcs between O and the parents of E. Adding to the AR of O's own parents, these AR sequences produced an intermingled resulting network where the number of parents increased substantially for all variables except O. As a result, the CPTs for these state variables expanded.

We also observe in the table that the average number of flips per sample is considerably greater in SER than in LW. This is a direct consequence of the increase in CPT size. Although this is a 1.5-fold increase, the tree structure already helps reduce this number (as well as the size of CPTs) to a minimum by ignoring irrelevant variables — in a by-experiment we conducted, we had 366 distributions in total and 500 flips per sample on average when the unenhanced ER algorithm was used.

The average execution time per sample of SER is three times that of LW. This may seem unpromising at first sight, but given SER converged at 170 samples while LW still did not converge at 1000 samples (Figure 6.10), SER still beats LW in terms of accuracy and total execution time. The only way for LW to decrease its variance in the computed posterior seems to be at least a 5-fold increase in the sample size, demanding significant increase in total execution time.

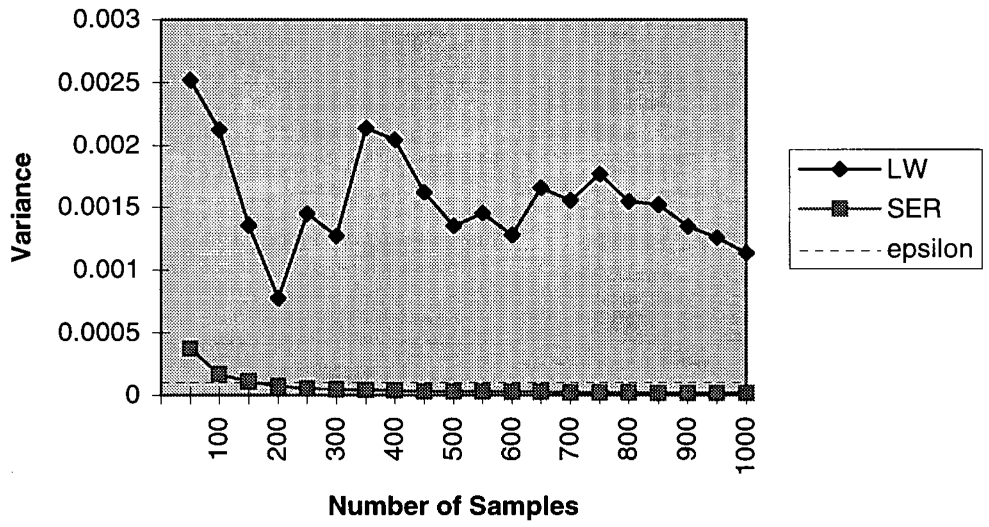


Figure 6.10: Convergence behavior — Case 5

7. Conclusion and Extensions

In this thesis we investigated the problem of divergence from reality when LW is used to perform probabilistic inference in DPNs [KAKR95]. An experiment was conducted to demonstrate the undesirable effect of a straightforward application of LW where it generated simulations that ignored the observed evidence and therefore became increasingly irrelevant. Posterior probability values thus obtained induced great variance across runs and virtually did not converge even at a sample size of one thousand.

As a remedy, we modified the ER algorithm in [KAKR95], which employs the AR algorithm in [SHAC86], with an intention to guide simulation back on track. Experimental results show that ER was indeed capable of bringing simulation closer to reality, providing accurate computed values that converged quickly. However, the additional space and time complexity required by AR were substantial.

In view of these problems, we devised the SER algorithm using a tree-structured CPT representation [BODG95]. SER avoids the problem of divergence from reality by using SAR, which was presented in detail in Chapter 4, to keep simulation on track while reducing the size of CPTs with a compact representation. We also used a query-directed approach and presented the techniques of relevant variable extraction and satisfiable query sampling to speed up probabilistic inference. A series of experiments was conducted to test our SER algorithm incorporated with the aforementioned techniques. With the exception of an ad-hoc counterexample, results show that the SER algorithm outperformed both LW and ER in all our experimental cases in terms of rate of convergence as well as compactness of CPT representation.

From the performance of SER in our experiments, it is postulated that SER will be found practical in applications involving complex models with highly interdependent variables. We would like to see SER being applied to specific problem structures, especially in a partially observable environment where decisions have to be made based on some set of computed posterior probability values.

As for refinement of the algorithm and its implementation, there is room for exploration in how determinism can be exploited effectively in performing SAR. We have seen that the presence of deterministic values in a tree-structured CPT may help remove the influence of some of the parents of the node in concern during SAR. The ability to detect and exploit determinism can certainly increase both computational and spatial efficiencies.

Another challenge lies in the development of a graphical user interface to visualize the simulation process. A visual simulator of this kind was implemented in [CHEU95] where *policies* can be visually simulated. A reward structure was also built in for policy evaluation purposes. It is yet unknown whether the SER algorithm can find its place in this area where causal inference (or projection) rather than diagnostic inference plays a major role.

Bibliography

[BFGK96] Craig Boutilier, Nir Friedman, Moisés Goldszmidt and Daphne Koller, "Context-Specific Independence in Bayesian Networks", in *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, (to appear). Portland, OR, 1996.

[BODG95] Craig Boutilier, Richard Dearden and Moisés Goldszmidt, "Exploiting Structure in Policy Construction", in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1104-1111. Montreal, 1995.

[BOGO96] Craig Boutilier and Moisés Goldszmidt, "The Frame Problem and Bayesian Network Action Representations", in *Proceedings of the Eleventh Biennial Canadian Conference on Artificial Intelligence*, pp. 69-83. Toronto, 1996.

[BOPO96] Craig Boutilier and David Poole, "Computing Optimal Policies for Partially Observable Decision Processes Using Compact Representations", in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, (to appear). Portland, OR, 1996.

[BOPU95] Craig Boutilier and Martin L. Puterman, "Process-Oriented Planning and Average-Reward Optimality", in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1096-1103. Montreal, 1995.

[BOUT95] Craig Boutilier, "Some Considerations on Process-Oriented Planning". (manuscript), 1995.

[CHCO90] R. Martin Chavez and Gregory F. Cooper, "An Empirical Evaluation of a Randomized Algorithm for Probabilistic Inference", in *Uncertainty in Artificial Intelligence*, **5**:191-207. Elsevier: North-Holland, 1990.

[CHEU95] Adrian Y. W. Cheuk, "Simulating Plan Execution in Stochastic Domains", (manuscript), 1995.

[COOP90] Gregory F. Cooper, "The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks", in *Artificial Intelligence Journal*, **42**:393-405, 1990.

[DEKA89] Thomas Dean and Keiji Kanazawa, "A Model for Reasoning about Persistence and Causation", in *Computational Intelligence*, **5**(3):142-150, 1989.

[FUCH90][24] Robert Fung and Kuo-Chu Chang, "Weighing and Integrating Evidence for Stochastic Simulation in Bayesian Networks", in *Uncertainty in Artificial Intelligence*, **5**:209-219. Elsevier: North-Holland, 1990.

[HENR88][25] Max Henrion, "Propagating Uncertainty in Bayesian Networks by Probabilistic Logic Sampling", in *Uncertainty in Artificial Intelligence*, **2**:149-163. Elsevier: North-Holland, 1988.

[HENR90] Max Henrion, "An Introduction to Algorithms for Inference in Belief Nets", in *Uncertainty in Artificial Intelligence*, **5**:129-138. Elsevier: North-Holland, 1990.

[KAKR95] Keiji Kanazawa, Daphne Koller, and Stuart Russell, "Stochastic Simulation Algorithms for Dynamic Probabilistic Networks", in *UAI 95 Proceedings*, pp. 346-351. Montreal, 1995.

[PEAR87] Judea Pearl, "Evidential Reasoning Using Stochastic Simulation of Causal Models", in *Artificial Intelligence*, **32**:245-257. Elsevier: North-Holland, 1987.

[24] Originally published in *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, Windsor, Ontario, 1989.

[25] Originally published in *Proceedings Workshop on Uncertainty in Artificial Intelligence*, Philadelphia, PA, 1986, under the title "Propagating Uncertainty by Logic Sampling in Bayes' Networks".

[PEAR88] Judea Pearl, **Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference**. Morgan Kaufmann, 1988.

[POOL93] David Poole, "Probabilistic Horn Abduction and Bayesian Networks", in *Artificial Intelligence Journal*, **64**(1):81-129, 1993.

[RUNO95] Stuart Russell and Peter Norvig, **Artificial Intelligence: A Modern Approach**. Prentice Hall, 1995.

[SHAC86] Ross D. Shacter, "Evaluating Influence Diagrams", in *Operations Research*, **34**(6):871-882, 1986.

[SHPE90][26] Ross D. Shacter and Mark A. Peot, "Simulation Approaches to General Probabilistic Inference on Belief Networks", in *Uncertainty in Artificial Intelligence*, **5**:221-231. Elsevier: North-Holland, 1990.

[SMSO73] Richard D. Smallwood and Edward J. Sondik, "The Optimal Control of Partially Observable Markov Processes over a Finite Horizon", in *Operations Research*, **21**:1071-1088, 1973.

[26] Originally published in *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, Windsor, Ontario, 1989.