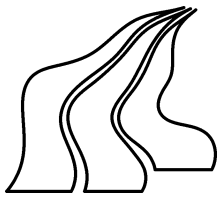
Blocking Gibbs Sampling for Inference in Large and Complex Bayesian Networks With Applications in Genetics

Claus Skaanning Jensen

AALBORG UNIVERSITY

Institute for Electronic Systems
Department of Computer Science



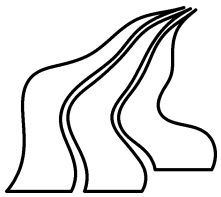
Blocking Gibbs Sampling for Inference in Large and Complex Bayesian Networks With Applications in Genetics

A thesis submitted to the
Faculty of Technology and Science at Aalborg University
for the degree of Doctor of Philosophy.

Claus Skaanning Jensen

AALBORG UNIVERSITY

Institute for Electronic Systems
Department of Computer Science
Fredrik Bajers Vej 7E, DK-9220 Aalborg Ø, Denmark



Preface

This thesis is the result of my Ph.D. study at the Department of Computer Science, Aalborg University, Denmark. The thesis consists of two parts, the first concerning the development of a stochastic method for inference in large and complex probabilistic expert systems (i.e., Bayesian networks). The second part concerns the practical application of the method for inference in pedigrees, i.e., pedigree analysis.

The first and second parts of the thesis correspond partly to four papers written during the Ph.D. study: Jensen, Kong & Kjærulff (1995), Jensen & Kong (1996), Jensen (1996a), Jensen & Sheehan (1997). Even though these four papers have not been directly reproduced in the thesis, they approximately correspond with chapters of the thesis in the following way :

Jensen et al. (1995): Chapters 6, 9 and 16.

Jensen & Kong (1996): Chapters 6, 9 and 17.

Jensen (1996a): Chapter 8.

Jensen & Sheehan (1997): Chapter 18.

Most of the chapters have been heavily rewritten to fit into the larger framework of the thesis, and new chapters have been added concerning work not covered by the papers. It has been attempted to make the thesis into a self-contained whole that can be read by people with no special knowledge of Bayesian networks, Markov chain Monte Carlo methods or genetics without consulting too many external references. For this reason, basic theory of Bayesian networks and pedigree analysis have also been included in the thesis.

The study has been partly theoretical and partly practical. As a result of the practical part, many computer programs have been written. Two of these, that have been put into the public domain, for performing pedigree analysis on general pedigrees, are documented in Appendix A.

Acknowledgements

Grant support was provided by the Danish Research Councils through the PIFT programme.

I wish to thank the members of the ODIN and DINA groups at Aalborg University for providing a stimulating environment, and in particular, Uffe Kjærulff and Finn V. Jensen for many valuable comments throughout the Ph.D. study.

From September, 1995 to March, 1996, I was a visiting student of University of Chicago, Illinois, USA. I wish to thank all the people there for a stimulating and interesting experience, and in particular, Augustine Kong, which showed me and

Jeanette great hospitality, and also provided me with much inspiration for further research through valuable discussions.

I also wish to thank Nuala Sheehan for many valuable and stimulating discussions during her stay at Aalborg University in 1995 and my stay at the University of Loughborough, UK in 1996, and a genuine interest in my work.

Finally, I am greatly indebted to Jeanette who during my three years of doctoral studies provided me with great support on the home front, and often carried much more than her share of the daily duties, in particular during the last months where I worked day and night on the thesis.

Aalborg, Denmark, May 1997 Claus Skaanning Jensen

Contents

Chapter 1

Introduction

1.1 Inference in Probabilistic Expert Systems

Reasoning in domains with uncertain knowledge is a very common activity for human beings. For many years it has been attempted to formalize this activity, and provide methods for efficient handling of complex problems with uncertain knowledge on computers. Some of the first attempts at handling such general problems lead to the development of rule-based systems and fuzzy logic. Though rule-based systems in the beginning did not provide any methods for handling of uncertain information, a method using the so-called certainty factors was later developed by Shortliffe & Buchanan (1975). The area of fuzzy logic was specifically developed to attempt handling the many problems with inherent uncertainty, see (Zadeh 1983).

These methods run into problems when several pieces of information with uncertainty are combined. They are not in general able to handle uncertainty in a completely consistent manner. A better way of representing uncertainty has been developed over the last couple of decades. In this area the uncertainty are modeled using probability theory in graphical models. Thus, the uncertainty (now probabilities of states of variables) can be combined using Bayes' formula, offering a consistent approach. The problem domains are modeled in so-called graphical models where the variables are represented as vertices in a graph and the dependences between variables are represented with edges in the graph (Darroch, Lauritzen & Speed 1980, Wermuth & Lauritzen 1983). Another benefit from this representation is that the conditional independences that are almost always present among many variables can be exploited and used for creating efficient inference methods.

These probabilistic graphical models have many names, but they have often been denoted *Bayesian networks* due to the usage of Bayes' formula, and this is also the name that will be used in this thesis. The context in which they are most often used is expert systems, and they are thus also denoted *probabilistic expert systems*. However, as Bayesian networks represent the problem-specific domain, and not the expert (such as, e.g., rule-based systems), we emphasize this difference by denoting them *decision support systems*, as they should only serve as decision support, not as "trusted" experts.

Several methods for inference in Bayesian networks (*belief updating*) have been proposed over the years. They fall in two categories, exact and stochastic methods. Exact methods have the inherent problem that inference in Bayesian networks is generally NP-hard (Cooper 1990), i.e., there is often an exponential relationship between the number of variables and the complexity of computation. This limits the problems for which exact inference methods are feasible. Thus, for problems

where exact inference is impractical, it is necessary to turn to stochastic methods.

For exact inference in Bayesian networks, methods for handling singly connected networks (i.e., trees) were developed by Kim & Pearl (1983). However, singly connected networks only constitute a small subclass of interesting real-world problems, so it was important to develop methods for exact inference in general networks. Methods in the *clustering* category have been the most successful for handling general networks (Pearl 1986b, Lauritzen & Spiegelhalter 1988, Shenoy & Shafer 1990, Lauritzen 1992). They basically transform the looped network into a tree by combining variables into clusters, and perform belief updating in this cluster tree by passing around messages containing "beliefs". It has been proved by Shachter, Andersen & Szolovits (1991) that all these clustering methods for exact probabilistic inference in Bayesian networks are fundamentally equivalent.

The exact methods are unable to handle many complex real-world networks, however, as the clusters sometimes have to contain many variables to be able to transform the network into a tree. As each cluster has a belief table associated with it, containing beliefs (e.g., probabilities) on each of the combinations of the values of its variables, these tables can grow to astronomical sizes. The *conditioning* method of Pearl (1986a) attempts to reduce the sizes of these cluster tables by basically trading storage requirements with time. This method, however, only makes exact inference feasible for a slightly larger class of problems.

To handle problems of general size and complexity, one has to turn to stochastic methods with which it is usually possible to obtain results accurate to some wanted precision. Stochastic methods are also referred to as *Monte Carlo* methods, due to their random nature. Monte Carlo methods in general draw samples from the required distribution, and then forms sample averages to approximate expectations. In general, producing independent samples from the required distribution is not feasible, but the samples do not necessarily have to be independent. Markov chain Monte Carlo (MCMC) methods are a subset of Monte Carlo methods that obtains dependent samples by running a Markov chain designed such that its equilibrium distribution is the distribution, we want to estimate. The samples can be more or less dependent depending on the MCMC method. MCMC methods have shown to be very practical and efficient methods for inference in general statistical models, and in particular, Bayesian networks. The first MCMC method, the Metropolis algorithm, was proposed by Metropolis, Rosenbluth, Rosenbluth & Teller (1953), and this method was later generalized to the Metropolis-Hastings algorithm by Hastings (1970). The Gibbs sampler is a special case of the Metropolis-Hastings algorithm, but has become the most popular due to its intuitive explanation and simple implementation, however, it is not necessarily the best choice in the general case. Interestingly, the Gibbs sampler had been known in the statistical physics literature for years as the *heat bath algorithm* (Creutz 1979, Ripley 1979) before it was suggested by Geman & Geman (1984) for image restoration and brought into wide recognition.

There are a few conditions that must be fulfilled for the Gibbs sampler to work. Usually, the most critical of these is the requirement that the Markov chain induced by the sampler must be irreducible. If this does not hold, the sample space contains noncommunicating sets that the Gibbs sampler is unable to reach, thus it will never be able to correctly estimate the required distribution. Also, even if the chain is irreducible, the chain may be near-reducible and mixing so slow that it requires astronomical sample sizes to move around the sample space.

For problems exhibiting these problematic characteristics, several advanced MCMC methods have been proposed. These fall in two groups, those that expand the state space, and those that do not. Among the state space expanding techniques, one of

the most promising is *simulated tempering* (Geyer 1991, Marinari & Parisi 1992, Geyer & Thompson 1995). Simulated tempering maintains a hierarchy of Markov chains ranging from the most "heated" chain that one can obtain independent samples from, to the "coldest" chain that has the distribution of interest as equilibrium distribution, and may be reducible and/or mixing very slowly. The simulated tempering method can thus overcome problems of reducibility and slow mixing in the general case. However, to work in practice it requires the construction of several "heated" chains which may be difficult, and the computational overhead of running these chains may also be large.

A method that does not expand the state space, and thus avoids producing samples that can not be used for estimating the distribution of interest, is *blocking Gibbs sampling* originally presented by Jensen et al. (1995). This algorithm combines a particular clustering method for exact inference, the *junction tree propagation method* (Lauritzen & Spiegelhalter 1988, Jensen, Lauritzen & Olesen 1990) with the Gibbs sampler. The algorithm makes it possible to implement the general Gibbs sampler where the components consist of many variables instead of a single one. Thus, many variables are updated jointly (in practice, usually more than 90%) often resolving problems of reducibility and slow mixing. Thus, Part I of the thesis covers all topics generally associated with the blocking Gibbs sampler.

1.2 Genetics Applications

In the area of genetics, a long standing problem has been the updating of probabilities in pedigrees, i.e., pedigree analysis. In pedigree analysis the variables represent, e.g., the genotypes and the phenotypes of individual members of the pedigree, and some of these may be observed. For probabilistic inference in pedigrees, the exact method of *peeling* was developed by Cannings, Thompson & Skolnick (1978). This method is yet another variation of the general cluster-tree algorithms, interestingly developed years before the first exact belief updating schemes appeared for Bayesian networks.

As the other exact methods, peeling also suffers from problems with astronomical table sizes when the pedigree gets complex. This problem has until now been handled in several ways in the genetics community. First, there has been a tendency to avoid complex pedigrees, and focus on smaller problems, e.g., by using the *sib-pair method* of Penrose (1935) later developed into the *affected* sib-pair method that only considers nuclear families, or by applying peeling to almost singly connected pedigrees. Another method has been an extension of the peeling method combining it with conditioning developed into the popular software packages, `LINKAGE` (Lathrop & Lalouel 1984, Lathrop, Lalouel, Julier & Ott 1985) and `FASTLINK` (Cottingham Jr., Idury & Schäffer 1993, Schäffer, Gupta, Shriram & Cottingham Jr. 1994). This method is only able to handle pedigrees with a very low number of loops.

Further, while pedigrees can be represented as Bayesian networks, due to the nature of the conditional probability tables inherent in pedigree analysis (e.g., penetrance probabilities), these problems often suffer from severe reducibility when applying MCMC methods. It has been shown that only in the case where diallelic loci are considered, irreducibility can (almost always) be guaranteed (Sheehan & Thomas 1993). When a locus with more than two alleles are considered, the underlying chain may or may not be irreducible. No general method exists for establishing whether a Gibbs sampler will be irreducible when applied to a specific problem in pedigree analysis.

The method of simulated tempering has been applied to pedigree analysis and

shows promise in this field. Also, the *sequential imputation* method (Kong, Cox, Frigge & Irwin 1993, Irwin, Cox & Kong 1994) for performing *linkage analysis* (the estimation of the distance between two genes) shows promise. The method is not iterative but handles one locus at a time using peeling and thus is not able to handle pedigrees with more than a few loops.

The blocking Gibbs sampler has also been used successfully in pedigree analysis. Jensen et al. (1995) successfully applied the blocking Gibbs sampler to simple pedigree analysis, and Jensen & Kong (1996) applied it to two-point linkage analysis. Thus, Part II of the thesis covers the topics associated with these genetics applications, and, among other things, describes the results of the two papers.

1.3 Overview of the Thesis

Part I covers all topics associated with the general definition and aspects of the blocking Gibbs sampling algorithm. First, in Chapter 2, a specific introduction to this part of the thesis is given. In Chapter 3, the intuition and theory behind Bayesian networks are outlined, followed by a description of the junction tree method for performing exact inference in Bayesian networks in Chapter 4. In Chapter 5, the theory of Markov chains and MCMC methods are outlined, including definition and other aspects of the Gibbs sampling algorithm. Then, in Chapter 6, the blocking Gibbs sampling algorithm is defined, followed by a non-rigorous proof of its irreducibility in Chapter 7. In Chapter 8, a feasible method for finding a legal configuration in a Bayesian network, and more specifically for finding a starting configuration for a Gibbs sampler, is given. This method is based on the *conditioning* method of Pearl (1986a) and is shown to be deterministic in practice when applied to some examples from genetics, but this is not proven for the general case. In Chapter 9, various methods for selecting the blocks of the blocking Gibbs sampler are given. The block selection methods are mostly based on considerations regarding the storage requirements of blocks. It is attempted to make the blocks contain as many variables as possible and at the same time keeping their storage requirements at a feasible level. However, the blocks must also be constructed according to other criteria, such as ensuring irreducibility of the induced Markov chain. Then, in Chapter 10, the forward sampling algorithm that is applied to the *barren* variables is presented. Barren variables are non-observed variables with no observed descendants. The forward sampling of these variables is beneficial, as independent samples can be obtained for these variables, resulting in faster convergence towards the required distribution.

Part II covers all topics associated with the genetics applications. First, in Chapter 11, a specific introduction to this part is given. In Chapter 13, it is detailed how pedigrees can be represented with Bayesian networks. Three representations are presented, the first representing genotypes as variables, the second representing genes as variables, and the third including information about recombination. Then, in Chapter 14, the different types of reducibility problems that can arise in pedigree analysis are introduced, and it is explained how most of them can be handled with the blocking Gibbs sampler. Sometimes a more serious problem is near-reducibility which occurs in completely different situations and can cause extremely slow mixing. In Chapter 15, two situations in pedigree analysis that cause near-reducibility are presented along with solutions to them using the blocking Gibbs sampler. In Chapter 16, the blocking Gibbs sampler is applied to pedigree analysis in a pedigree of 20,000 pigs attempting to estimate the marginal probabilities of the genotypes of the pigs conditional on the observed phenotypes of a subset of the pigs. In Chapter 17, the blocking Gibbs sampler is successfully applied to two-point linkage analysis in

a complex human pedigree infected with a rare heart disease. In Chapter 18, an algorithm by Lin, Thompson & Wijsman (1994) for identifying the noncommunicating sets of a single variable updating MCMC method applied to pedigree analysis is analyzed, and pointers towards a general algorithm are provided.

Finally, Chapter 19 provides a discussion ending the two parts of the thesis and pointers towards future research.

Part I

Blocking Gibbs Sampling

Chapter 2

Introduction to Part I

Over the last decade or so fast methods for exact inference have been developed for graphical models (Bayesian networks) of complex stochastic systems (Cannings, Thompson & Skolnick 1976, Cannings et al. 1978, Lauritzen & Spiegelhalter 1988, Shenoy & Shafer 1990, Jensen et al. 1990, Dawid 1992, Lauritzen 1992, Spiegelhalter, Dawid, Lauritzen & Cowell 1993). The success of the exact methods has become a reality despite the fact that computation in Bayesian networks is generally NP-hard (Cooper 1990), i.e., there is often an exponential relationship between the number of variables and the complexity of computation. Thus, for a large class of real-world problems, exact computation is impractical.

Stochastic simulation techniques (Monte-Carlo methods) have thus become increasingly popular alternatives to exact methods, since they are flexible, easy to implement, and their computational complexity tends to scale manageably with the size of the networks under consideration (Gelfand & Smith 1990, Thomas, Spiegelhalter & Gilks 1992, Gelman & Rubin 1992, Geyer 1992, Smith & Roberts 1993). Their main disadvantages are associated with difficulty in deciding whether the desired precision has been reached and the fact that even moderately sized problems may compute slowly. Using these simple Monte Carlo methods, computation time often exceeds any acceptable level when considering large networks (e.g., thousands of variables).

This part of the thesis suggests and evaluates a variant of Gibbs sampling (Geman & Geman 1984) involving simultaneous sampling of sets of variables using the junction tree architecture for exact local computations (Jensen 1988, Jensen et al. 1990, Dawid 1992). Since the method simulates set (*blocks*) of variables jointly, it will be referred to as *blocking Gibbs*.

In the following chapters, the Bayesian network paradigm and the exact local computation scheme will be introduced in Chapters 3 and 4. Then, elementary Markov chain theory will be introduced, leading to the description of the general Gibbs sampling algorithm in Chapter 5. Finally, the blocking Gibbs algorithm will be described in great detail in Chapter 6, and various issues related to the algorithm will be covered in subsequent chapters: irreducibility of the algorithm in Chapter 7, finding a legal starting configuration in Chapter 8, selecting optimal blocks in Chapter 9, and forward sampling of barren variables in Chapter 10.

Chapter 3

Bayesian Networks

In this chapter, the intuition and theory behind Bayesian networks will be outlined. As this chapter and the following are intended to give the reader a basic useful understanding of Bayesian networks and not a deep theoretical understanding, some theorems are shown but the proofs are left out. The interested reader can find most of these in (Jensen 1996b). The term *variable* will in this and the following chapters be used for both stochastic variables and nodes in Bayesian networks.

3.1 A Simple Example

Bayesian networks provide a way to model problem areas using probability theory. The Bayesian network representation of the problem can then be used to provide information on some variables given information on other variables. To provide the necessary intuition for understanding Bayesian networks, a small example will be shown first.

Figure 3.1 is taken from (Charniak 1991). Charniak (1991) describes the problem of Figure 3.1 as follows :

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

The Bayesian network in Figure 3.1 can be used to predict what will happen (e.g., if the family goes out, the dog goes out) or to infer causes from observed effects (if the light is on and the dog is out, then my family is probably out).

The ellipses of Figure 3.1 represent variables which have a finite number of states (or even a continuous state space). In this example, e.g., the variable *hear-bark* has two states, *true* and *false*, indicating whether or not dog barking is heard. The connections represent causal relations between variables such that in Figure 3.1 *light-on* is assumed causally dependent on *family-out*.

In Figure 3.1 the necessary probabilities have been shown as well. For root variables (variables with no predecessors), prior probabilities must be specified,

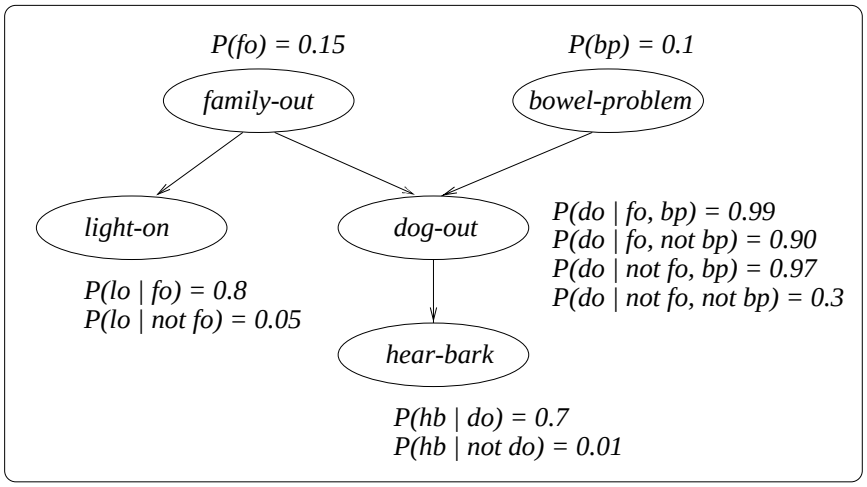


Figure 3.1: A Bayesian network representing a small decision support example.

e.g., we specify a probability of 0.1 that the dog has bowel-problems. For non-root variables, conditional probabilities must be specified given all possible combinations of the states of their direct predecessors (called *parents*).

Bayesian networks allow one to calculate the conditional probabilities of the variables in the network given that the values of some of the variables have been observed. For instance, if it is observed that the light is on (*light-on = true*) but the dog is not heard (*hear-bark = false*), the conditional probability of *family-out* can be calculated given these pieces of evidence. In this case it is 0.5. In the next sections it will be explained how this calculation is performed. But first, a definition of Bayesian networks is given.

3.2 Definition of Bayesian Networks

A Bayesian network consists of a set of *variables* (nodes) and a set of *directed edges* (connections between variables — also called *links*). Each variable has a finite set of mutually exclusive states. The variables together with the directed edges form a *directed acyclic graph* (DAG). A directed graph is acyclic if there is no directed path $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n$, $n > 1$, such that $v_1 = v_n$.

For each variable v with parents $w_1, \ldots, w_n$ there is defined a conditional probability table $P(v|w_1, \ldots, w_n)$. Obviously, if v has no parents, this table reduces to the marginal probability table $P(v)$.

We denote the set of variables represented in a Bayesian network as a *universe*, $U = \{v_1, \ldots, v_n\}$. If we have the joint probability table $P(U) = P(v_1, \ldots, v_n)$ we can calculate marginal probabilities for any variable v_i conditional on any of the remaining variables. However, the size of $P(U)$ grows exponentially with the number of variables in U, quickly making it impractical to store. Bayesian networks provide a more compact representation of $P(U)$ which still allows us to calculate marginals for any variable conditional on other variables. This is specified in Thm. (3.2.1).

Theorem 3.2.1 (The chain rule). *If BN is a Bayesian network over* $U = \{v_1, \ldots, v_n\}$,

then :

$$P(U) = \prod_i P(v_i|pa(v_i)),$$

where $pa(v_i)$ denotes the set of parents of v_i. We also say that P admits recursive factorization according to the DAG of BN.

We define the *descendants* of a set of variables, A, to be the variables $B = \{v|$ there exists a directed path from a variable in A to $v\}$. The descendants of a set, A, are also denoted $\mathrm{de}(A)$. The *nondescendants* of A are then $\mathrm{nd}(A) = U \setminus (\mathrm{de}(A) \cup \{A\})$. Further, we can define the *ancestors* of a set of variables, A, to be the variables $B = \{v|$ there exists a directed path from v to a variable in $A\}$. The ancestors of a set, A, are also denoted $\mathrm{an}(A)$. Finally, following the definition of parents of a variable v, $\mathrm{pa}(v)$, we define the children of a variable v as $\mathrm{ch}(v)$.

As mentioned in the previous section we want to use Bayesian networks for computing probabilities of some variables conditional on other variables. We sometimes denote these probabilities as *beliefs*, and denote this computing process as performing *inference* in the Bayesian network.

If a variable is observed, i.e., its value is known, we also denote this as the presence of *evidence* for the variable. We can have two types of evidence, *hard evidence* and *soft evidence*. Hard evidence denotes the certain knowledge that a variable is in a specific state while soft evidence denotes evidence where a belief is assigned to each state, not necessarily rendering any of the states impossible.

Chapter 4

Exact Local Computations

In this chapter we will describe how to change Bayesian networks into a more efficient representation suitable for exact computations, and it will be explained how these exact local computations are performed.

4.1 Junction Trees

A Bayesian network provides a representation of a universe of stochastic variables $U = \{v_1, \ldots, v_n\}$. We want to use the Bayesian network for performing updating of beliefs on the variables. However, the Bayesian network is not immediately suited for this. The reason for this is the possible presence of loops in the network.

Consider the small network with a loop in Figure 4.1. Imagine that evidence is inserted on v_4 and we want to know the conditional probabilities at v_3. In this network, the change at v_4 will affect v_3 in more than one way. Not only does v_3 have to account for the direct change in v_4 but also the change in v_1 that will be caused by v_4 through v_2.

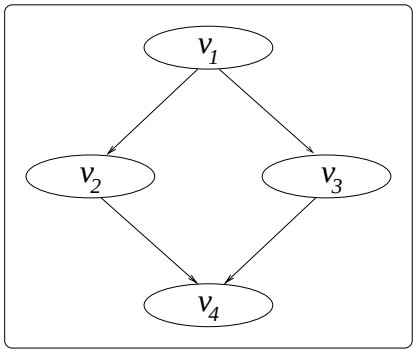


Figure 4.1: A Bayesian network with a loop.

To evaluate looped networks one has to turn the network into an equivalent singly connected one (a tree). There are a few ways to perform this task. The most common ways are variations on a technique called *clustering*. In clustering, one combines variables until the resulting graph is singly connected. Thus, to turn Figure 4.1 into a tree, one can combine variables v_2 and v_3. In this new Bayesian network, depicted in Figure 4.2, the variable resulting from the clustering of v_2 and

v_3 will have a state space consisting of all combinations of the states of v_2 and v_3. That is, if $|\mathrm{Sp}(v_2)| = s_2$ (the size of v_2's state space) and $|\mathrm{Sp}(v_3)| = s_3$, then the state space of the combined variables will contain $s_2 \cdot s_3$ configurations. In large networks with many loops, it is sometimes necessary to combine many variables thus getting very large state spaces. It then becomes important to make the optimal choice of variables to put in the same clusters. In the following we will present one variant of the clustering technique proposed by Lauritzen & Spiegelhalter (1988) and improved by Jensen et al. (1990). This technique which operates with the *junction tree* as the basic clustering of the variables in the Bayesian network, is currently the fastest exact algorithm for most applications.

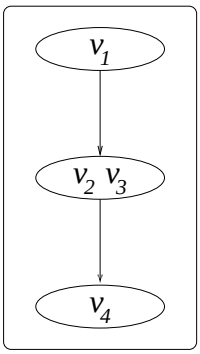


Figure 4.2: A singly connected Bayesian network equivalent to the one in Figure 4.1.

First, we will introduce the notion of *cluster trees*:

Definition 4.1.1 (Cluster tree). A cluster tree over a universe of variables U is a tree of clusters of variables from U. The clusters are subsets of U, and the union of all clusters is U.

To the links of the cluster tree is attached *separators* which consist of the intersection of the adjacent clusters. Furthermore, to each cluster and separator is attached a joint probability table over the configurations of its variables. Basically, Figure 4.2 is a cluster tree over the Bayesian network in Figure 4.1, except that the separators are not shown. A *junction tree* is a more restricted version of a cluster tree :

Definition 4.1.2 (Junction tree). A cluster tree is a junction tree if, for each pair of clusters C_1, C_2, all clusters on the path between C_1 and C_2 contain the intersection $C_1 \cap C_2$.

The requirement of Def. (4.1.2) for junction trees is also known as the *junction tree property*. When we consider junction trees, the clusters are denoted *cliques*, due to the fact that the clusters are formed from maximal sets of variables that are all pairwise linked. In Section 4.5, it will be described how these cliques are constructed.

4.2 Junction Tree Propagation

Propagation in junction trees provides a means of updating the beliefs of variables given evidence on a subset of the variables. In this section we will describe how such a propagation is performed.

So far we have not described how to construct a junction tree (see Section 4.5), but for now we will assume the following.

BN is a Bayesian network over a universe U. JT is a junction tree corresponding to BN, where

- For each variable v in BN there is at least one clique C in JT such that $\{v\} \cup \text{pa}(v) \subseteq C$.
- Initially, all cliques and separators of JT are given a probability table of all ones.
- For each variable v in BN we choose exactly one clique C in JT containing $\{v\} \cup \text{pa}(v)$, and we multiply $P(v|\text{pa}(v))$ on C's probability table.

It is then clear that the product of all the cluster tables of JT is equal to the product of all conditional probability tables in BN. This can be extended to the statement that $P(U)$ is equal to the product of all the cluster tables of JT divided by the product of all separator tables. This is clearly true as the separator tables contain only ones. This is important, as later we will require this equality to hold at all times, even when the separators contain probabilities different from ones.

4.2.1 Absorption in Junction Trees

We now introduce an operation called *absorption* in junction trees. It has the effect of propagating information from one clique to another. A simple absorption from one clique to another can be seen in Figure 4.3. At the same time, Figure 4.3 also illustrates how we will draw junction trees in the thesis; *cliques* as circles, and *separators* as boxes.

Definition 4.2.1 (Absorption). We let C_1 and C_2 be neighbours in a junction tree with separator S. t_{C_1}, t_{C_2} and t_S are their original probability tables. The absorption operation consists of the following steps :

- we calculate $t_S^* = \sum_{C_1 \setminus S} t_{C_1}$,
- then assign to S the new probability table t_S^*,
- and assign to C_2 the new probability table $t_{C_2}^* = t_{C_2} \frac{t_S^*}{t_S}$

We then say that C_2 has *absorbed* from C_1.

The intuition behind absorption is that the only information on C_1 with any relevance for C_2 is information on their common variables, i.e., the variables of S. Therefore, the absorption operation transmits only information on the variables of S, t_S^*. From this we can define the concept of *consistency*:

Definition 4.2.2 (Consistency). If C_1, C_2 and S hold the same information about S, that is if :

$$\sum_{C_2 \setminus S} t_{C_2} = t_S = \sum_{C_1 \setminus S} t_{C_1},$$

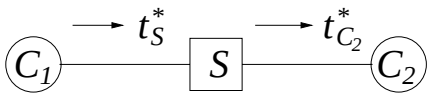


Figure 4.3: A small junction tree with two cliques, C_1 and C_2, and one separator, S. C_2 absorbs from C_1. $t_S^* = \sum_{C_1 \setminus S} t_{C_1}$ and $t_{C_2}^* = t_{C_2} \frac{t_S^*}{t_S}$.

then the link (consisting of C_1, C_2 and S) is consistent. Furthermore, if all links of the junction tree are consistent then we say that the tree is consistent.

If a link is consistent, absorption does not change anything, and if the entire junction tree is consistent, absorption will have no effect on it. Furthermore, a junction tree is *globally consistent* if for any cliques C_1 and C_2 with intersection S we have :

$$\sum_{C_1 \setminus S} t_{C_1} = \sum_{C_2 \setminus S} t_{C_2} \tag{4.1}$$

It can be shown that a consistent junction tree is also globally consistent.

It is not always possible to absorb from C_1 through S to C_2. It requires that t_S has non-zero entries corresponding to non-zero entries in t_{C_2}. This leads to the notion of a *supportive* link. A link is supportive if it allows absorption in both directions, and a junction tree is supportive if all its links are supportive. If a junction tree is initialized such as specified earlier this section by giving all cliques and separators probability tables of all ones, then it will be supportive, and it will remain so after absorption. This leads us towards one of the main results of this section :

Theorem 4.2.1. *Let JT be a supportive junction tree. Then the product of all clique probability tables divided by the product of all separator tables is invariant under absorption.*

Thm. (4.2.1) ensures that if BN is a Bayesian network over a universe, U, and JT is a junction tree representation of BN, then JT remains a representation of BN after a series of absorptions, and $P(U)$ can be calculated as the product of all clique tables divided by the product of all separator tables.

4.2.2 Message Passing in Junction Trees

Now that we have defined the absorption operation we can use it to perform a series of absorptions in the junction tree sending information around the tree. Such a scheme which propagates information around the junction tree using absorptions will be called a *message passing scheme*. A message passing scheme for probability updating in singly connected DAGs was first presented by Kim & Pearl (1983). We say that a clique C_1 *sends a message* to its neighbour C_2 when C_2 absorbs from C_1.

In a message passing scheme, a clique C_1 can send exactly one message to a neighbour C_2, and it may only be sent when C_1 has received a message from all its other neighbours.

An example of this is shown in Figure 4.4. First, the leaf cliques with only one neighbour, C_1, C_3, C_4 and C_6 can send a message to their neighbour. Then, C_2

can send a message to C_5, and C_5 can send a message to C_2. Then, finally C_2 can send messages to C_1 and C_3, and C_5 can send messages to C_4 and C_6.

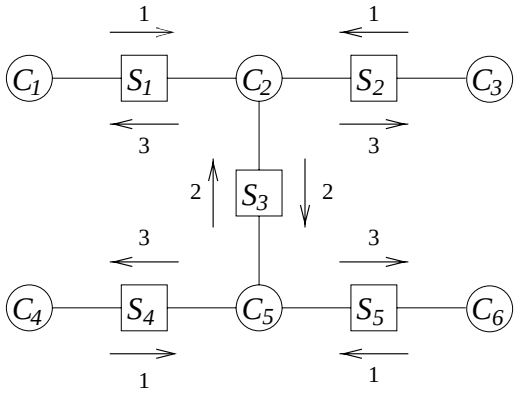


Figure 4.4: The message passing scheme in a small junction tree.

The message passing scheme thus continues to let cliques send off messages, until every clique has sent a message to each of its neighbours. This leads on to the theorem :

Theorem 4.2.2. *Let JT be a supportive junction tree. Let messages be passed around in JT according to the message passing scheme. Then the following is true :*

- *message passing can continue until a message has been sent in both directions through each link.*

- *when a message has been sent in both directions through each link, JT is consistent.*

The following theorem shows what the message passing scheme yields in the junction tree :

Theorem 4.2.3 (Junction tree propagation). *Let BN be a Bayesian network representing $P(U)$ for some universe, U, and let JT be a junction tree corresponding to BN. Let $e = \{f_1, \ldots, f_m\}$ be findings on the variables $\{v_1, \ldots, v_m\}$ in U, where f_i is a table over $Sp(v_i)$. For each $i \in \{1, \ldots, m\}$ find a clique containing v_i and multiply its table with f_i. Then, after a complete message passing we have for each clique C and for each separator S that :*

$$t_C = P(C, e) \qquad t_S = P(S, e) \qquad P(e) = \sum_C t_C = \sum_S t_S$$

The findings in Thm. (4.2.3) consist of zeros and ones, rendering some configurations impossible. Thm. (4.2.3) ensures that when evidence has been inserted, and message passing has been completed, then the probability tables of all cliques and separators reflect the evidence in a consistent manner.

A propagation method based on the step, $t_S^* = \sum_{C_1 \setminus S} t_{C_1}$, for sending messages is also called a *sum propagation* method.

4.3 HUGIN Propagation

HUGIN propagation as proposed by Jensen et al. (1990) is a modification of the algorithm proposed by Lauritzen & Spiegelhalter (1988). Further, the method has been implemented in the expert-system shell HUGIN (Andersen, Olesen, Jensen & Jensen 1989). Similar methods (*peeling*) have been introduced for pedigree analysis by Elston & Stewart (1971) and generalized by Cannings et al. (1978). These methods all belong to the category *exact local computations* as they perform exact probability updating in Bayesian networks using local computations.

HUGIN propagation is an optimization of the simple message passing scheme. It consists of two operations: *Distribute Evidence* and *Collect Evidence*.

Distribute Evidence: This operation distributes evidence from a single clique, C, to all other cliques in the junction tree. In practice, C sends messages to all of its neighbours who recursively send messages to all of their neighbours except the one from which the message came. The operation is illustrated in Figure 4.5.

Collect Evidence: This operation collects all evidence in the junction tree towards a single clique, C. In practice, C asks all its neighbours to send it a message, and if they are not allowed to do so, they recursively pass the request to all neighbours except the one from which the request came. Finally, the leaf cliques will be able to fulfill the request, and the cliques can start sending messages towards C. The operation is illustrated in Figure 4.6.

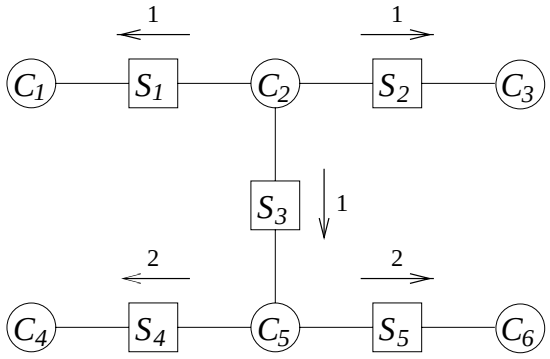


Figure 4.5: The *Distribute Evidence* operation applied to C_2.

HUGIN propagation combines these two operations. When evidence has been inserted on an arbitrary number of variables, a clique, C, (the root) is selected at random. First, *Collect Evidence* is applied to C, and then *Distribute Evidence* is applied to C.

In some situations it is not necessary to apply both operations. If evidence is only inserted in one clique, C, only *Distribute Evidence* needs to be applied to C, and if conditional probabilities are wanted only for one clique, C, it suffices to apply *Collect Evidence* to C.

HUGIN propagation is a general scheme that can use other operations for message passing than $t_S^* = \sum_{C_1 \setminus S} t_{C_1}$, some of which are mentioned in the next section.

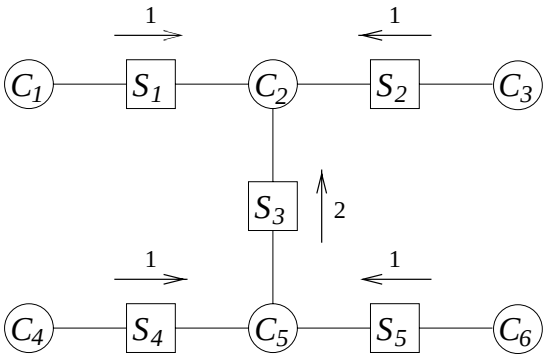


Figure 4.6: The *Collect Evidence* operation applied to C_2.

4.4 Random Propagation

Random propagation was first described by Dawid (1992). The result of the random propagation algorithm is a random configuration on the Bayesian network sampled from the correct distribution. The random propagation algorithm is a type of *exact sampling*. Random propagation uses a message passing scheme such as HUGIN propagation, and thus it requires the construction of a junction tree corresponding to the Bayesian network. As we will see later, the storage requirements of a junction tree can grow to enormous amounts making exact methods impractical. This also renders the random propagation method impractical, making stochastic methods such as Markov chain Monte Carlo methods necessary.

Random propagation proceeds as follows. First, a random clique, C_0, in the junction tree is selected. If the tree is inconsistent, apply *Collect Evidence* to C_0. Then, the distribution on C_0 is proportional to the desired distribution of the variables given evidence, and we can simulate a value for the variables in C_0. The distribution phase now proceeds as follows. We consider the passage outwards from a clique C to a neighbouring clique C', across a separator S. Prior to this we have simulated a value for the variables in C, denoted ξ_C from which we can extract ξ_S. We also have a probability distribution $t_{C'}$ on C'. We now need to simulate a value for the variables $A = C' \setminus S$. We define a restricted probability distribution on A, denoted p_A such that $p_A(x_A) = p_{C'}(y)$, where $y \in \mathrm{Sp}(C')$ is such that $y_A = x_A$ and $y_S = \xi_S$. A value ξ_A is now simulated for the variables in A from the table of probabilities obtained by normalizing p_A. ξ_A and ξ_S are then combined to form $\xi_{C'}$. Continuing like this, we shall eventually have simulated a value for every variable, from the desired joint distribution.

The random propagation method is used in the blocking Gibbs algorithm for simulating large sets of variables jointly.

A similar scheme described by Dawid (1992) allows us to find the most probable configuration in the Bayesian network. It is similar to the sum propagation scheme where messages are calculated as $t^*_S = \max_{C_1 \setminus S} t_{C_1}$ instead of $t^*_S = \sum_{C_1 \setminus S} t_{C_1}$. This scheme is called the *max propagation scheme*.

4.5 Construction of Junction Trees

In this section it will be described how to construct a junction tree from a Bayesian network. This process will also be denoted as compiling the Bayesian network. Basi-

cally, this process has the objective of turning the possibly looped Bayesian network into a singly connected network (a tree) in which calculations can be performed much more easily with, e.g., the HUGIN propagation. The required steps attempt to combine as few variables as possible into cliques, thus minimizing the storage requirements. They do this by adding an optimal set of links to the Bayesian network, and finally all cliques (maximal sets of variables that are all pairwise linked) can be found. The junction tree is then constructed by connecting these cliques in an optimal way. More formally, the procedure consists of four steps (illustrated in Figure 4.7):

1. The Bayesian network is *moralized* resulting in the *moral graph*, see Figure 4.7(b).
2. The moral graph is *triangulated* resulting in the *triangulated graph*, see Figure 4.7(c).
3. The *cliques* are obtained from the triangulated graph, see Figure 4.7(d).
4. The cliques are organized in the *junction tree*, see Figure 4.7(e).

Each of these four steps will be described in the following.

Moral graph: Obtaining the moral graph is very simple. The basic operation turning a Bayesian network into its corresponding moral graph simply adds an undirected link (a *moral link*) between any pair of variables with a common child, and drops the directions of the original links.[1] The result of this operation can be seen in Figure 4.7(b).

As mentioned earlier in this section, for each variable, v, in the Bayesian network there must be a clique containing v and its parents. This is taken care of by the moralization by linking all parents together. Thus, v and its parents are all pairwise linked, and will be included together in at least one clique. In the following, if we have a Bayesian net, BN, we will denote the corresponding moralized graph, BN^m.

Triangulated graph: An undirected graph is *triangulated* if any cycle of length > 3 has a chord. It has been proven that a junction tree can be created for a DAG, if and only if, the DAG is triangulated. In Figure 4.7(c) the graph has been triangulated by adding a chord e–f in the cycle of length 4. In practice the moral graph is triangulated by applying a *triangulation algorithm*. The result of the triangulation algorithm is an *elimination sequence*. A variable, v, is said to be *eliminated* by adding links (*fill-in links*) such that all of its neighbours are pairwise linked, and then removing v together with its links. The elimination sequence is simply a list of the variables describing the order in which they should be eliminated from the graph. The following can then be shown :

Theorem 4.5.1. *A graph is triangulated if and only if an elimination sequence exists such that following the order of this sequence all of its variables can be eliminated one by one without adding any links.*

Thm. (4.5.1) provides the basis for a triangulation algorithm. The goal of the triangulation algorithm is to construct an *optimal* elimination sequence. In this context, the elimination sequence is optimal if the storage requirements of the resulting junction tree are as small as possible. The general problem of

[1] The moral graph was originally named such as it marries unmarried parents.

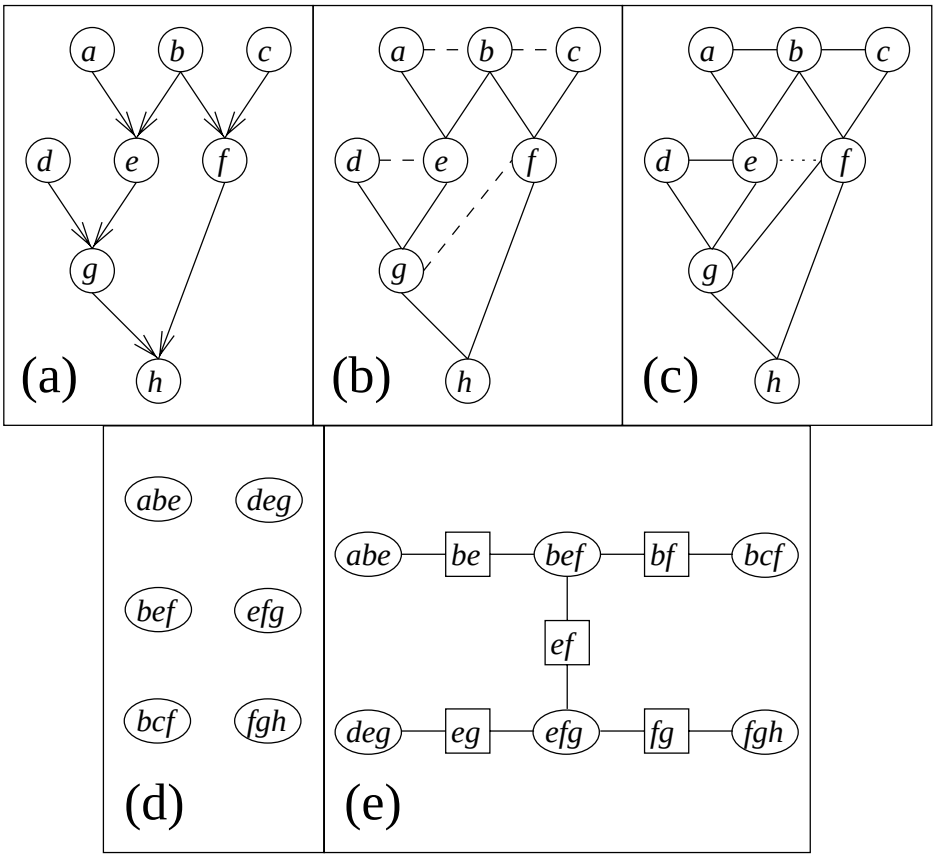


Figure 4.7: The process of turning a Bayesian network into a junction tree; (a) the original Bayesian network, (b) the moralized graph, (c) the triangulated graph, (d) the cliques, and (e) the junction tree. Moral links are dashed, and fill-in links are dotted.

exact methods is that they have storage requirements exponential in the size of the largest clique and the cliques of looped networks may sometimes grow extremely large. Thus, it is very important to keep the clique sizes as small as possible.

However, many elimination sequences can be found for the same network and it has been shown that it is NP-hard to find an optimal one (Arnborg, Corneil & Proskurowski 1987). Kjærulff (1990) has provided a few heuristics that often work well in practice. These heuristic methods are based on various criteria for selecting the next variable in the elimination sequence. They are all local methods, i.e., they only look one step ahead when selecting the next variable to eliminate. This clearly cannot always produce an optimal elimination sequence. The heuristics attempt to obtain elimination sequences that are not necessarily optimal wrt. the storage requirements of the junction tree, but optimal wrt. other criteria. However, the fulfillment of these other criteria in most cases leads to relatively good elimination sequences wrt. the storage requirements.

Minimum fill-in links: this approach selects the variable resulting in the

least number of new fill-in links. This approach clearly attempts to minimize the number of fill-in links which is not the same as minimizing the junction tree storage requirements. However, in most cases this criterion will result in near-optimal elimination sequences also in this respect.

Minimum clique size: this approach selects the variable having the least number of neighbours. If a variable is eliminated, links are added between its neighbours, and they end up in the same clique, i.e., this approach attempts to minimize the size of new cliques.

Minimum clique weight: this approach selects the variable where the variable and the neighbours have the smallest joint state space. Obviously, this heuristic attempts to minimize the table size of new cliques.

Minimum fill-in weight: this approach selects the variable resulting in the smallest sum of fill-in weights of its links. The *fill-in weight* of a link is defined as the product of the number of states of the two variables it links together. This way it is attempted to find an indicator for the increase in state space by adding the new fill-in links, attempting to find a means to minimize the size of the tables of new cliques.

Depending on the structure of the Bayesian network and the number of states of the variables, any of these heuristics may be better than the others. Further, as the heuristic methods often have to choose randomly between several variables with equal fulfillment of the criterion in use, any of them may come up with several different elimination sequences to the same network. In practice, a good triangulation algorithm is to run each of the heuristic methods a number of times, and then simply use the best elimination sequence found.

Cliques: The cliques can be extracted from the triangulated graph by selecting all maximal sets of variables that are all pairwise linked. This has been done in Figure 4.7(d). The storage requirements of the cliques can now be examined by computing the sizes of state spaces. If the storage requirements are too large, there are two possibilities. Either, another attempt may be made to find a better triangulation resulting in smaller cliques, or if this seems impossible, the Bayesian network can not be handled by exact inference methods.

Junction tree: When the cliques have been found, the junction tree can be constructed by linking the cliques such that a tree is formed. They have to be linked in such a way that the junction tree property holds, however. This operation has been performed in Figure 4.7(e).

There exists a number of simple algorithms for the construction, e.g., Prim's (Gibbons 1985) or Kruskal's (Aho, Hopcroft & Ullman 1974).

Defining the *weight* of a link in a junction tree to be the number of variables in the corresponding separator, the idea of Kruskal's algorithm is to successively insert links of maximal weight between cliques unless a cycle is created. This approach leads to a tree that fulfills the junction tree property.

In the expert-system shell HUGIN step three and four of the above are implemented using the fast *maximum cardinality search* algorithm of Tarjan & Yannakakis (1984) that simultaneously checks whether the graph is triangulated and, if it is, constructs a junction tree.

In general the task of exact inference in Bayesian networks is NP-hard (Cooper 1990). As mentioned earlier, it is NP-hard to find an optimal triangulation. Thus, it is time-consuming to find near-optimal solutions to this problem. However, the HUGIN propagation method makes it unnecessary to attempt to find a near-optimal

solution to this problem every time evidence is propagated. Instead, this problem is handled only once in the junction tree construction phase where much effort can be devoted to it.

As previously mentioned, the cliques of the junction tree may grow to astronomical sizes making exact inference practically impossible. Such a large clique may be created, for instance, by a variable with many parents. It is fortunate that these complexities can be estimated prior to actual processing, because, when the estimates exceed reasonable bounds, we can switch to a stochastic method such as blocking Gibbs sampling.

4.6 Conditioning

However, it is possible to trade the storage requirements of cliques with time requirements of the exact inference method. Thus, by spending much more time in the process, we can lower the storage requirements somewhat, and still obtain exact results. We can do this by using the idea of *conditioning* by Pearl (1986a) and later described in the book of Pearl (1988). By using conditioning, we can sometimes handle a problem that requires, say, 10 or 100 times more storage than we have available. But if our problem is larger than that, we still have to turn to stochastic methods.

The benefits of conditioning can be explained as follows. Consider Figure 4.8. In Figure 4.8(a), a simple Bayesian network can be seen. Variables c and h have been observed. With this information, the network in Figure 4.8(a) can be turned into the equivalent but different looking network in Figure 4.8(c). Given the evidence, the modified network contains the same conditional independences as the original one (see Section 4.7). If a variable, v, is observed, it can be replaced by a number of *clones*, one connected with the parents, and one for each of the offspring. This clearly makes no difference, as the clones of v have the same effect on other variables as v had. Following this, the Bayesian network in Figure 4.8(c) is obtained in two steps. First, as c is observed, the loops between the parents of c and the offspring of c going through c can be broken, due to the simple fact that if c is instantiated to any value, the parents and offspring of c become independent and each of the offspring of c becomes independent of the other offspring. To illustrate this, a number of clones of c replace c, one connected to the parents of c (c_1), and one for each of the offspring (c_2, c_3, c_4). Second, as h is observed, the same operation can be carried through for h, replacing it with three clones.

In the book of (Pearl 1988), conditioning was used as a method of reducing complexity of a Bayesian network such that exact inference becomes possible. A variable, v, connecting many parts of the network and thus present in many large cliques, and thus being responsible for much of the storage requirements, is selected. Then v is instantiated with each of its states, one at a time. With v instantiated, it can be replaced by a number of clones, thus effectively breaking loops in the Bayesian network. For each of the states of v, exact inference is performed in the Bayesian network, obtaining marginal beliefs on all other variables with v instantiated. Finally, the true marginal beliefs can be found by summing over the states of v. Here is a formal definition of conditioning :

Definition 4.6.1 (Conditioning). We have a universe, $U = \{v_1, \ldots, v_n\}$, and a Bayesian network, BN, representing U. On our computer we have a storage capability of S_C. The junction tree corresponding to BN has storage requirements of $S_{JT} > S_C$. We select variables, $A \subseteq U$, such that by conditioning upon them and replacing them with clones, the storage requirements of the corresponding junction

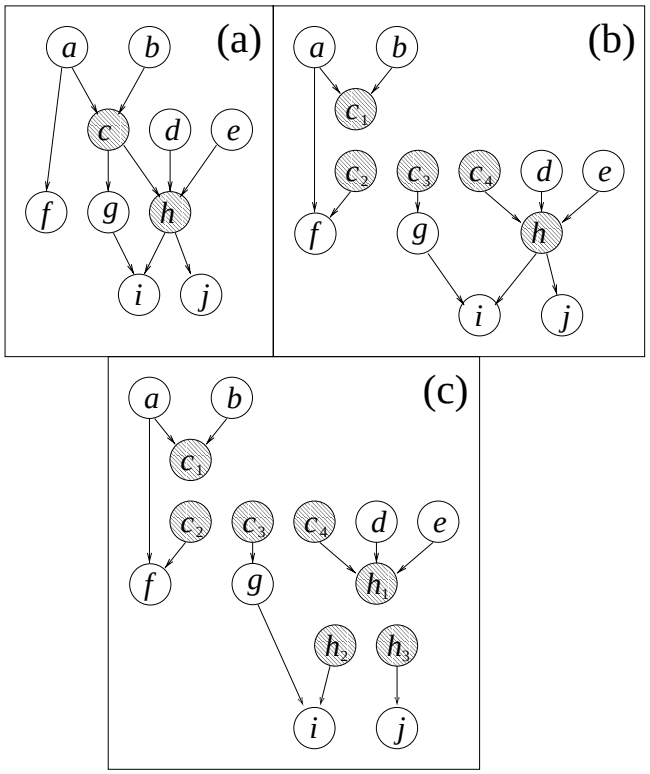


Figure 4.8: Conditioning upon variables c and h, turns the initial Bayesian network in (a) into the broken-up network in (c). The dark circles represent variables that have been observed.

tree is reduced to S_C or less. The reduced Bayesian network is denoted BN_r, and the corresponding reduced junction tree is denoted JT_r. Thus, in the reduced versions of the network and junction tree, the variables in A have been replaced by their clones.

Assume, we have evidence e entered in JT_r, and we want to know the marginal conditional probability of some variable, v^*, $P(v^*|e)$. Then, for each of the configurations, c_i, of the variables in A, insert c_i as evidence in their clones, perform exact inference (e.g., a HUGIN propagation), and find $P(v^*|c_i, e)$. Finally,

$$P(v^*|e) = \sum_{c_i} P(v^*|c_i, e).$$

The complexity of conditioning depends on the size of the joint state space of the variables in A, as for each configuration, c_i, of the variables in A, a propagation is performed. If there are many variables in A, the joint state space will be astronomical, thus rendering conditioning impractical.

Imagine that A contains 10 variables, each with 3 states, then $3^{10} = 59049$ propagations must be performed. For 20 variables with 3 states each, we need more than 3 billion propagations. It is thus easily seen that the method of conditioning is only practical in the borderline cases where exact inference is impractical due to the storage requirements of the junction tree that are slightly too large, and can be made practical by conditioning upon just a few variables.

In this thesis, we take advantage of conditioning in several respects. First, conditioning is a general method that can be used when we have observed variables. Consider a Bayesian network where a set of variables, $A \subseteq U$, is observed. Thus, before compiling the network into the corresponding junction tree, we can condition upon these variables and replace them with clones. This reduces the storage requirements of the junction tree, and we are able to handle larger and more complex networks. For instance, in sampling schemes where propagations are performed in the same junction tree a lot of times, it is important to condition upon variables that are always observed. However, if different variables are observed each time a HUGIN propagation is performed, it is not efficient to break their loops, as we then have to recompile the network at each propagation, and as previously mentioned, compiling Bayesian networks requires finding near-optimal "solutions" to an NP-hard problem which can be time-consuming. In the blocking Gibbs sampler this technique is used, as throughout the duration of each run, the same variables are observed each time a block is visited.

Second, conditioning is also used in the algorithm for finding a legal starting configuration, described in Chapter 8.

4.7 Independence Properties in Bayesian Networks

In this section, criteria for conditional independence between two groups of variables, given a third group of variables are presented for both directed and undirected graphs. These are denoted the directed and undirected global Markov properties. The properties are presented using the theory of Markov fields. The interested reader can refer to Lauritzen, Dawid, Larsen & Leimer (1990) for a thorough exposition of the area.

4.7.1 Markov Fields over Undirected Graphs

For an undirected graph G over universe $U = \{v_1, \ldots, v_n\}$ with variables $v_1, \ldots, v_n$, we let $A \perp\!\!\!\perp B|C$ denote "A is conditionally independent of B given C", where $A, B, C \subseteq U$. Furthermore, we define :

Definition 4.7.1 (Boundary, bd). The boundary of a set of variables, $A \subseteq U$, is the set of variables in $U \setminus A$ that are neighbours to at least one variable in A. The boundary of A is also denoted $\mathrm{bd}(A)$.

Basically, the boundary of a variable, v, in the moralized graph corresponding to a Bayesian network, equals the parents, offspring and spouses of v (as these are linked with v during the moralization).

Definition 4.7.2 (Closure, cl). The closure of a set of variables, $A \subseteq U$, is the set of variables $\mathrm{cl}(A) = A \cup \mathrm{bd}(A)$.

Then, a probability distribution p defined on $H = \mathrm{Sp}(v_1) \times \cdots \times \mathrm{Sp}(v_n)$ can have various Markov properties denoted by the letters, **L**, **P** and **G** :

L: p obeys the local Markov property relative to G, if for any variable $v_i \in U$,

$$v_i \perp\!\!\!\perp U \setminus \mathrm{cl}(v_i)|\mathrm{bd}(v_i).$$

P: p obeys the pairwise Markov property relative to G, if for any pair (v_i, v_j) of non-adjacent variables,

$$v_i \perp\!\!\!\perp v_j | U \setminus \{v_i, v_j\}.$$

G: p obeys the global Markov property relative to G, if for any triple (A, B, C) of disjoint subsets of U such that C separates A from B in G,

$$A \perp\!\!\!\perp B | C.$$

With the above Markov properties, if p is strictly positive, we have that $\mathbf{G} \Rightarrow \mathbf{L} \Rightarrow \mathbf{P}$, but if p is not strictly positive this does not hold in general.

Further, if p factorizes according to G, i.e., the joint probability distribution equals

$$p = \prod_{v \in U} p(v | \mathrm{bd}(v)), \tag{4.2}$$

it can be shown that it also obeys the global Markov property. For a proof, see Lauritzen et al. (1990). If p factorizes according to Eq. (4.2), p is said to *admit recursive factorization*. For a Bayesian network representing a probability distribution, p, p always admits recursive factorization according to the moral graph of the network.

4.7.2 Markov Fields over Directed Graphs

If BN is a Bayesian network over a universe, U, and $A \subseteq U$, then BN_A is a Bayesian network over the variables in A, retaining only the links in BN that connect variables in A.

Consider a DAG, BN, where p admits recursive factorization, abbreviated to **DF** (directed factorization). Then, for any triple (A, B, C) of disjoint subsets of variables of BN such that C separates A from B in $(BN_{A \cup B \cup C \cup \mathrm{an}(A \cup B \cup C)})^m$,

$$A \perp\!\!\!\perp B | C. \tag{4.3}$$

This property is also called the directed global Markov property, abbreviated by **DG**.

When examining the conditional independences between sets A, B, and C, we only consider the reduced Bayesian network $(BN_{A \cup B \cup C \cup \mathrm{an}(A \cup B \cup C)})^m$. This can be explained with the notions of *serial*, *converging*, and *diverging* connections. In Figure 4.9 we have examples of serial, diverging and converging connections. With the serial connection (1), A and C are conditionally independent given B, with the diverging connection (2), the offspring, B and C, are conditionally independent given the parent, A, and with the converging connection (3), the parents A are marginally independent, but may become dependent given the value of C. These three notions lead us to formulate the more complex notion of *d-separation* (where the d stands for *directional*):

Definition 4.7.3 (d-separation). Two variables v_i and v_j in a Bayesian network are d-separated if for all paths between v_i and v_j there is an intermediate variable v_k such that :

- the connection leading through v_k is serial or diverging and the state of v_k is known, or

- the connection leading through v_k is converging and neither v_k nor any of v_k's descendants are observed (have received evidence).

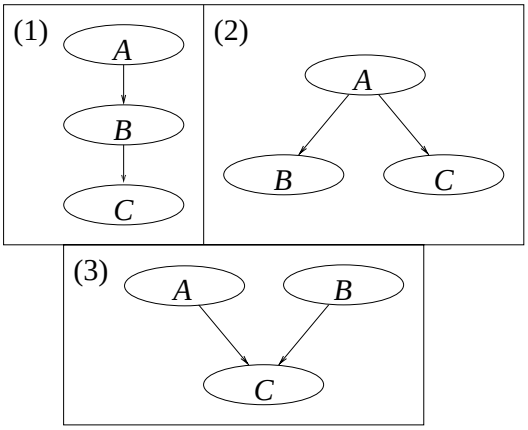


Figure 4.9: Three Bayesian networks representing serial (1), diverging (2) and converging (3) connections.

If, for a variable, v_k, neither v_k nor any of its descendants are observed, we say that v_k is *barren*.

The above definition explains why we only have to consider the ancestors of the variables in question (and the variables themselves) when examining conditional independences. If we want to examine $A \perp\!\!\!\perp B|C$ for three disjoint sets of variables, it is assumed that only the values of the variables in C are given, thus no descendants of C are observed. This means that any path between a variable in A and a variable in B not leading through any variable in C, but leading through a variable not included in $A \cup B \cup C \cup \text{an}(A \cup B \cup C)$, must go through at least one variable, v^*, that is not observed (and have no observed descendants), and is placed in a converging connection. The sets A and B are thus *d-separated* according to Def. (4.7.3).

Following this, if we want to examine the conditional independences between sets of variables in a Bayesian network, we moralize the graph and examine the dependences in the ancestral set of the involved variables. For instance, if we want to investigate whether $\{B, E\} \perp\!\!\!\perp \{D, F\}|\{H, I\}$ in the Bayesian network in Figure 4.10, we examine the corresponding moralized graph in Figure 4.11. Since the ancestors of I include the entire DAG, we have to take the entire DAG in Figure 4.10 into consideration. Then we have to examine whether there is a path from $\{B, E\}$ to $\{D, F\}$ without passing $\{H, I\}$. Due to the moralization, there is a link between B and F. Therefore, $\{B, E\} \perp\!\!\!\perp \{D, F\}|\{H, I\}$ is false.

Of course, when examining $A \perp\!\!\!\perp B|C$, and there are observed variables, D, outside of C, then the variables in D and their ancestors have to be included in the investigation.

Also, it can be shown (Lauritzen et al. 1990) that C d-separates A from B if and only if C separates A from B in $(BN_{A \cup B \cup C \cup \text{an}(A \cup B \cup C)})^m$.

Another Markov property is the directed local Markov property, (**DL**), stating that a variable is conditionally independent of its nondescendants, given its parents,

$$v \perp\!\!\!\perp (\text{nd}(v) \setminus \text{pa}(v))|\text{pa}(v). \tag{4.4}$$

It can be shown that **DL**, **DG**, and **DF** are equivalent assuming existence of a density p that admits recursive factorization. These results will be used in some of the subsequent chapters.

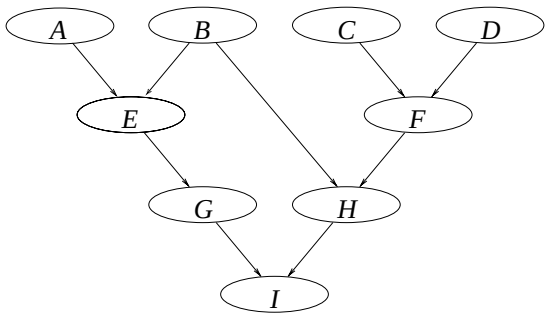


Figure 4.10: A Bayesian network.

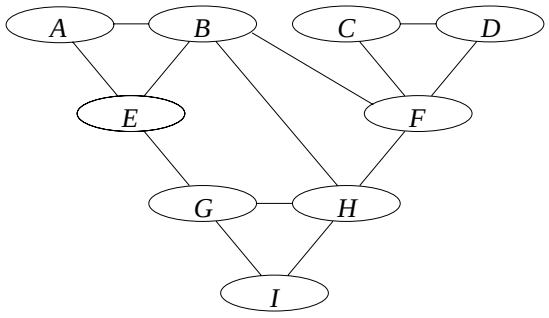


Figure 4.11: The moralized graph of the network in Figure 4.10.

Chapter 5

Markov Chain Monte Carlo Methods

As mentioned in the previous chapter, exact inference in Bayesian networks is not practical in general, as the complexity involved with the processing (the size of the largest clique) may reach astronomical figures. In these cases, stochastic (*Monte Carlo*) methods can be applied to obtain results within reasonable bounds of precision.

Markov chain Monte Carlo (MCMC) methods are a subset of Monte Carlo methods that are applicable to a very wide range of problems. Generating samples from the complex distributions encountered in artificial intelligence applications is often not easy. Sampling methods based on *Markov chains* often alleviate this problem as they allow and encourage drawing from conditional distributions which is often much simpler. Furthermore, sampling methods based on Markov chains are incorporated into a framework where it can be proved that the correct distribution is generated, at least in the limit, as the length of the chain grows. Even with very complicated problems, it is often possible to design a Markov chain with the wanted equilibrium distribution.

Typically, the Markov chain explores the space in a "local" fashion, e.g., $x^{(t)}$ differs from $x^{(t-1)}$ in only one variable. This is the case, for instance, with almost all implementations of the *Gibbs sampler*, thus denoted the *single-site* Gibbs sampler. The blocking Gibbs sampler, presented in the next chapter, explores the space in a more "global" fashion, updating many variables jointly in each step.

If the Markov chain is *irreducible*, the averages of the realizations of the sampler converge to the equilibrium distribution as the sample size goes to infinity, but if the chain is mixing slowly it may require huge sample sizes to get precise results. *Slow mixing* occurs particularly when problems are of high dimension, have very correlated variables, and the sampler updates one variable at a time, such as is the case for the single-site Gibbs sampler.

Although, the Gibbs sampler is only one of many variants of MCMC methods, it has become one of the most prominent for a number of reasons, with its intuitive explanation and simple implementation probably being the most important. Here, the many other MCMC methods, like the Metropolis and Hastings algorithms (Metropolis et al. 1953, Hastings 1970) will not be discussed. For recent reviews, see (Gelfand & Smith 1990, Thomas et al. 1992, Gelman & Rubin 1992, Geyer 1992, Smith & Roberts 1993, Besag, Green, Higdon & Mengersen 1995, Gilks, Richardson & Spiegelhalter 1996).

The Gibbs sampler has been known in the physics literature as the *heat bath algorithm* (Creutz 1979, Ripley 1979), and it was later brought into wider recognition by Geman & Geman (1984) which applied it to the problem of image restoration. Geman & Geman (1984) denoted it the *Gibbs sampler* as they used it for analyzing Gibbs distributions on lattices. Since then, the Gibbs sampler has been applied to many problems which where then considered unsolvable, and has been the subject of massive theoretical investigation. Fishman (1996) and Roberts & Sahu (1997) have investigated the difference between visitation schemes for the Gibbs sampler (random, fixed, etc.), and have furthermore obtained some preliminary theoretical results on the benefits of block updating. Convergence issues have been dealt with by many researchers (Amit, Grenander & Piccioni 1991, Rosenthal 1992, Tierney 1994, Polson 1995, Roberts & Tweedie 1995, Mengersen & Tweedie 1996) and it has been possible to establish theoretical bounds on rates of convergence for some Markov chains. However, these theoretical bounds are often much too high to be useful in practice. As a result, an assortment of convergence diagnostics based on the output of single or multiple chains are used in practice to detect convergence. For examples, see the review articles by Brooks & Roberts (1995) and Cowles & Carlin (1995). These diagnostics provide important information about the behaviour of the Markov chain, but they cannot be used to *prove* convergence.

Many extensions to the Gibbs sampler have been proposed during the last five years. A promising new direction is in the construction of new algorithms based on auxiliary variables and processes.

The most successful auxiliary variables method has been the Swendsen & Wang (1987) algorithm for sampling in continuous Markov random fields, e.g., the Ising model. The algorithm is able to handle severely multimodal distributions, but it does not seem applicable in the case of discrete Bayesian networks.

The state space expanding MCMC method, *simulated tempering* (Geyer 1991, Marinari & Parisi 1992, Geyer & Thompson 1995), is a more general method based on auxiliary processes. The algorithm has adopted its name from simulated annealing (Kirkpatrick, Gelatt & Vecchi 1983) which is a method for optimization that starts with a "heated" version of the problem, then "cools" it down until a solution is found. Simulated tempering maintains a hierarchy of Markov chains ranging from the most "heated" chain that is easy to sample from, to the "coldest" chain that induces the distribution of interest and likely suffers from reducibility and/or multimodality. The simulated tempering algorithm samples from one of the chains in a period, then proposes a move from the current chain to one of its adjacent chains. Moves are proposed such that the individual limit distributions are maintained. At the "coldest" chain we can collect information on the target distribution. At the "hottest" chain we are able to move around with much faster mixing, and we allow the sampler to jump between different modes. There are some problems with this approach, however. It seems that the computational overhead of running all the non-target chains may be large, and also it seems that the construction of such a hierarchy of "heated" chains may be a difficult task in practice.

A more direct approach for handling the problems of high-dimensional reducible/multimodal problems is *blocking Gibbs sampling* (Jensen et al. 1995) described in Chapter 6. This algorithm is based on the substantial amount of research in the area of exact inference in Bayesian networks (see Chapter 3) and a particular exact scheme, the *junction tree propagation method* (Lauritzen & Spiegelhalter 1988, Jensen et al. 1990) described in Chapter 4. The algorithm effectively makes it possible to sample sets (*blocks*) of variables simultaneously, with the blocks usually containing more than 90% of the variables. Due to the joint updating of the majority of variables, this sampling scheme can have very fast mixing and avoid problems of reducibility and multimodality. The method has been used successfully with

very hard applications in genetics, see (Jensen et al. 1995, Jensen & Kong 1996) and Chapters 16 and 17. However, there are still some issues that must be settled before blocking Gibbs becomes a truly general method for inference in very large and complex Bayesian networks. The most important is the selection of blocks, see Chapter 9. As of yet there is no general method for the construction of blocks that guarantee irreducibility. This problem corresponds to finding the noncommunicating classes of an MCMC sampler discussed by Lin et al. (1994) and more recently by Jensen & Sheehan (1997), see Chapter 18.

In the next section, elementary Markov chain theory will be introduced with a definition of many central notions used throughout this thesis. In the final section of this chapter, the general Gibbs sampler will be defined and the requirements for its convergence explained.

5.1 Markov Chain Theory

This section covers elementary Markov chain theory necessary for proving the convergence of Markov chain Monte Carlo methods, and in particular the Gibbs sampler. The section presents central notions in the area but does not delve deeply into theoretical matters. Also here, proofs of theorems are left out. Interested readers can refer to the large amount of material available on the theory of Markov chains for these proofs, e.g., a general book on the topic by Feller (1968), a simple proof of the ergodicity of the Gibbs sampler by York (1992), and a walkthrough of basic Markov chain theory by Neal (1993).

We start out with some basic definitions. In the following, x denotes an instantiation of X:

Definition 5.1.1 (Markov chain). A Markov chain is a series of random variables, $X^{(0)}, X^{(1)}, \ldots$, in which $X^{(t+1)}$ is conditionally independent of $X^{(0)}$, $X^{(1)}$, $\ldots$, $X^{(t-1)}$ given $X^{(t)}$. This basic Markov property can also be formulated more formally,

$$P(x^{(t+1)}|x^{(0)}, x^{(1)}, \ldots, x^{(t)}) = P(x^{(t+1)}|x^{(t)}).$$

The indices, $t = 0, 1, 2, \ldots$, are viewed as representing successive "time points". The $X^{(t)}$ have a common range, the *state space* of the Markov chain. The state space is assumed to be finite in this context, but it may also be infinite.

To start off the Markov chain, the *initial probabilities*, i.e., the marginal distribution for $X^{(0)}$, and the *transition probabilities*, i.e., the conditional distributions for $X^{(t+1)}$ given $X^{(t)}$, must be specified. Here, the initial probability of state x will be written as $p_0(x)$, and the transition probability for state x' at time $t+1$ given state x at time t as $T_t(x, x')$. In cases, where the transition probabilities do not depend on time, the Markov chain is denoted *stationary* (or *homogeneous*) and the transition probabilities will be written as $T(x, x')$. As we will only cover stationary Markov chains in this thesis, the latter notation will be used in the remainder of this section.

With these definitions, it is now possible to specify formally the probability of a state x occurring at time $t+1$, i.e., $p_{t+1}(x)$ in terms of the probabilities at time t:

$$p_{t+1}(x) = \sum_{x'} p_t(x')T(x', x) \tag{5.1}$$

We can also write the probabilities at time t as a row vector, $\mathbf{p}_t$, and the transition probabilities as a matrix, $\mathbf{T}$. $\mathbf{T}$ is an example of a *stochastic matrix* which is

a matrix with all elements non-negative and rows summing to one. Eq. (5.1) can then be written as :

$$\mathbf{p}_{t+1} = \mathbf{p}_t \mathbf{T} \tag{5.2}$$

Furthermore, $\mathbf{T}^k$ (the k-th power of the matrix $\mathbf{T}$) yields the "k-step" transition probabilities. With similar notation, we also write the "k-step" transition probabilities of a state x following a state x' as $T^k(x, x')$. We can then also write :

$$\mathbf{p}_t = \mathbf{p}_0 \mathbf{T}^t \tag{5.3}$$

Definition 5.1.2 (Invariant distribution). An invariant distribution for a Markov chain is one that persists forever once it is reached. More formally, the distribution $\pi(x)$ is invariant wrt. a Markov chain with transition probabilities $T(x, x')$, if,

$$\pi(x) = \sum_{x'} \pi(x') T(x', x)$$

If we let π represent a distribution as a vector, then it is invariant if and only if $\pi = \pi \mathbf{T}$. A Markov chain can have more than one invariant distribution, but it always has at least one.

Definition 5.1.3 (Detailed balance). A Markov chain satisfies the condition of detailed balance, if, when a transition occurs from a state picked according to a distribution π, then the probability of that transition being from x to x' is the same as the probability of it being from state x' to x. More formally, for all x:

$$\pi(x) T(x, x') = \pi(x') T(x', x)$$

For a Markov chain satisfying *detailed balance* (also labelled *time reversible*), π must be an invariant distribution, as :

$$\sum_{x'} \pi(x') T(x', x) = \sum_{x'} \pi(x) T(x, x') = \pi(x) \sum_{x'} T(x, x') = \pi(x). \tag{5.4}$$

However, though detailed balance implies the existence of an invariant distribution, it is possible for a distribution to be invariant without detailed balance holding.

When designing a Markov chain, it is not sufficient to find a Markov chain wrt. which the distribution we wish to sample from is invariant. We also have some further requirements but before we can state those we need a few definitions.

Definition 5.1.4 (Communicating states). Two states in the Markov chain are said to communicate if it is possible for the Markov chain to move from either state to the other one, i.e., for states x' and x,

$$\exists k_1, k_2 : \quad T^{k_1}(x, x') > 0 \text{ and } T^{k_2}(x', x) > 0.$$

The sets of states that communicate with each other forms an equivalence class, as the equivalence relation, *x communicates with x'*, is reflexive, transitive and symmetric.

Definition 5.1.5 (Irreducibility). If all the states of a Markov chain is in one equivalence class, i.e., all the states of the chain communicate pairwise, then the chain is said to be irreducible. If the chain is not irreducible, it is said to be *reducible*.

A sufficient condition for irreducibility is the *positivity* condition introduced by Besag (1974). Basically, positivity means that although the variables are dependent, any joint configuration of the variables are logically possible. This is violated in many settings (see Chapters 16 and 17) and often weaker requirements suffice to prove irreducibility. In pedigree analysis it has been proved by Sheehan & Thomas (1993) that in some cases irreducibility is fulfilled though positivity is not.

Another way to state some of the above requirements is through the definition of recurrence :

Definition 5.1.6 (Recurrent state). A state x is said to be recurrent, if and only if, starting from state x, eventual return of the Markov chain to this state is certain.

If a state is not *recurrent*, it is said to be *transient*. We can then define :

$$N(x, x') = \min\{t | X^{(t)} = x', \text{ given } X^{(0)} = x\}, \tag{5.5}$$

which means that $N(x, x')$ is the earliest time t, for which $X^{(t)} = x'$, given that the Markov chain was started with $X^{(0)} = x$. The probability of the earliest time being t, is :

$$f^{(t)}(x, x') = P(N(x, x') = t). \tag{5.6}$$

Then we can define the probability that the Markov chain enters the state x in a finite number of steps, when started in the same state :

$$f^*(x, x) = \sum_{t=1}^{\infty} f^{(t)}(x, x) = P(N(x, x) < \infty), \tag{5.7}$$

and the mean value of $N(x, x)$:

$$\mu(x, x) = E[N(x, x)]. \tag{5.8}$$

Then, we say that state x is *recurrent* if $f^*(x, x) = 1$, and *transient* if $f^*(x, x) < 1$. Also, we call x *null-recurrent*, if $\mu(x, x) = \infty$, and *positive recurrent* , if $\mu(x, x) < \infty$. A Markov chain is said to be positive recurrent, if all the states of the Markov chain is positive recurrent. It can be shown that a recurrent finite Markov chain is positive recurrent if and only if a proper invariant distribution exists, and any of these conditions may be easier to show in a given situation.

A stationary, irreducible, Markov chain on a finite state space with an invariant distribution (i.e., the chain is positive recurrent) is said to be *ergodic*. An ergodic Markov chain has the property that the probabilities at time t converge towards the invariant distribution as $t \to \infty$. For a proof of this result also called *the ergodic theorem*, see, e.g., (Tierney 1994). Obviously, an ergodic Markov chain can only have one invariant distribution which is also denoted its *equilibrium distribution*.

An ergodic Markov chain is furthermore said to be *geometrically ergodic* if there exists $0 < \lambda < 1$ and a function $f(t) > 1$ such that :

$$\sum_{x'} |T^t(x, x') - \pi(x')| \leq f(x)\lambda^t \tag{5.9}$$

for all x. The smallest λ for which there exists a function f satisfying Eq. (5.9) is denoted the *rate of convergence*. A geometrically ergodic Markov chain is said to converge to the joint density π at a geometric rate in t.

Further, we can define the concept of *warm-up*. When starting an ergodic Markov chain in a state not selected from the equilibrium distribution, it will take some time for the Markov chain to converge to the equilibrium distribution. This period of time is denoted the *warm-up* or alternatively the *burn-in*. If the chain is started in a state selected from the equilibrium distribution, no warm-up phase is required. In practice, it is often difficult to select a starting configuration for a Markov chain, and in particular to draw one from the equilibrium distribution. If this was possible, it would not be necessary to use a Markov chain in the first place. In Chapter 8, this problem is discussed further.

Finally, we will define an often used notion used about Markov chains :

Definition 5.1.7 (Mixing). The rate of mixing refers to the long-term correlations between states of a Markov chain, i.e., how far from an i.i.d. sample the state of the chain is. If they are highly correlated, the chain is said to be *mixing slowly*, and if they are close to independent, the chain is said to be *mixing rapidly*.

5.2 Gibbs Sampling

The Gibbs sampler is one of the simplest of the Markov chain Monte Carlo methods, but it has recently been the subject of much research, starting with Geman & Geman (1984) and Gelfand & Smith (1990). It can be used in many problems where the variables have conditional distributions that can easily be sampled from.

Definition 5.2.1 (Gibbs sampling algorithm). We wish to sample from the joint distribution for $\mathbf{X} = \{X_1, \ldots, X_n\}$ given by $P(x_1, \ldots, x_n\}$ where the range of the X_i's may be either continuous or discrete. The Gibbs sampler proceeds by repeatedly replacing the value of each component with a value picked from its distribution conditional on the current values of all other components. This process generates a Markov chain built from a set of transition probabilities, T_k, for $k = 1, \ldots, n$, with :

$$T_k(x, x') = P(x'_k | x_{-k}),$$

where x_{-k} denotes the set $\{x_i | i \neq k\}$, i.e., the step using T_k only changes the value of the component x_k and leaves the remaining components unchanged. The T_k are applied in sequence, ensuring that all components are visited. This sequence can be fixed, e.g., $T_1, T_2, \ldots, T_n$, or random. In the case where the sequence is fixed, the Gibbs sampler generates a stationary Markov chain with transition probabilities :

$$T(x, x') = T_1(x, x^{(1)}) T_2(x^{(1)}, x^{(2)}) \cdots T_{n-1}(x^{(n-1)}, x^{(n)}) T_n(x^{(n)}, x'),$$

or simpler :

$$\mathbf{T} = \mathbf{T_1 T_2} \cdots \mathbf{T_n}.$$

We can also explain the workings of the Gibbs sampler better by elaborating the steps of a fixed-scan sampler. The procedure for generating $X^{(t)}$ from $X^{(t-1)}$ can be described in n steps applying each of the previously defined transition probabilities, T_i:

1 Pick $x_1^{(t)}$ from the distribution for X_1 given $x_2^{(t-1)}$, $x_3^{(t-1)}$, ..., $x_n^{(t-1)}$.

2 Pick $x_2^{(t)}$ from the distribution for X_2 given $x_1^{(t)}$, $x_3^{(t-1)}$, $x_4^{(t-1)}$, ..., $x_n^{(t-1)}$.

$\vdots$

i Pick $x_i^{(t)}$ from the distribution for X_i given $x_1^{(t)}$, $x_2^{(t)}$, ..., $x_{i-1}^{(t)}$, $x_{i+1}^{(t-1)}$, ..., $x_n^{(t-1)}$.

$\vdots$

n Pick $x_n^{(t)}$ from the distribution for X_n given $x_1^{(t)}$, $x_2^{(t)}$, ..., $x_{n-1}^{(t)}$.

The above steps thus specify one *iteration* of the Gibbs sampler.

5.2.1 Ergodicity of the Gibbs Sampler

The many results from the theory of Markov chains described in Section 5.1 can be applied to the Gibbs sampler to determine whether a Gibbs sampler for a specific problem works, i.e., is ergodic.

First, we must verify that all the T_k leave the desired distribution invariant. Clearly, this is the case. T_k leaves the components x_i, for which $i \neq k$ unchanged, so for these components the desired marginal distribution is certainly invariant. Further, T_k draws the new state for x_k given the other components from the conditional distribution that is defined to be that which is desired (P). Together this ensures that the joint distribution of the x_i remains the same after all the T_k have been applied. This is equivalent to showing positive recurrence of the chain.

The other necessary criterion is to show that the Markov chain induced by the Gibbs sampler and built from the T_k's is irreducible. If this is the case, we are guaranteed that the Gibbs sampler is able to move from any state to any other state in a finite number of steps. If all the conditional probabilities used to define T_k are non-zero, we clearly have irreducibility, as it is possible to get from any state to any other state in only n steps where each step changes one component to that of the desired state. However, in general this is not true, and in particular it is not true for the genetics applications of this thesis. In general each individual case must be investigated to see whether irreducibility hold.

If the previous criteria are fulfilled, the Gibbs sampler induces a stationary, irreducible Markov chain with an invariant distribution, and is thus *ergodic*. Further, it has been shown that if the Gibbs sampler is ergodic, it is also geometrically ergodic as defined in Eq. (5.9). In practice, even though ergodicity holds for a Gibbs sampler, the Gibbs sampler may still be useless. This is due to the fact, that the Gibbs sampler can be *almost* reducible, i.e., its sample space consists of a number of areas that is very difficult to move between. An example of such a sample space can be seen in Figure 5.1. A Gibbs sampler with a thinly connected sample space like that in Figure 5.1 is said to be *multimodal* (or *near-reducible*). Multimodality is a common problem in many applications, also in particular with the genetics applications covered in this thesis. Multimodal Gibbs samplers, though ergodic, may be useless in practice, as they sometimes require astronomical sample sizes to reach adequate precision of the estimate. The problem of multimodality is covered in more detail in Chapter 15.

Def. (5.2.1) defines the general Gibbs sampling algorithm where each component may consist of several simple stochastic variables. However, with the version of Gibbs sampling usually implemented and used in practice, each component only consists of one variable. For this reason we call this algorithm the *single-site Gibbs sampler*. The single-site Gibbs sampler is very easy to implement, as each basic step consists of the drawing of one stochastic variable given the remaining variables, an operation that is usually very easy to implement. On the other hand, if each component consists of several variables, the conditional drawing of these variables given the remaining variables, requires an exact updating method like the one described in Chapter 4. This is, in fact, what blocking Gibbs is based on, see Chapter 6.

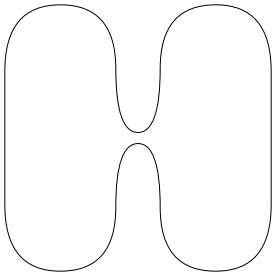


Figure 5.1: The sample space of a multimodal Gibbs sampler.

5.2.2 Gibbs Sampling in Bayesian Networks

When the Gibbs sampling algorithm is used for iterative stochastic inference in Bayesian networks, the basic step of drawing one component conditional on others can be simplified significantly by making use of the basic Markov properties of Bayesian networks, see Section 4.7.

When applying the single-site Gibbs sampler to the Bayesian network in Figure 5.2, we at some point in the sampling process have to draw variable G conditional on the remaining variables. We can then make use of the *boundary*, Def. (4.7.1) on page 25, and the local Markov property defined on page 25, to make this step simpler. Basically, the local Markov property states that the variable G is independent of the remaining variables given its boundary. In the moralized graph, the boundary of a variable equals the set of neighbours, i.e., the boundary of a variable when considering the Bayesian network equals its parents, its offspring, and its spouses (as the spouses are married with the variable when moralizing). The boundary is also denoted the *Markov blanket*. Thus, in Figure 5.2, the boundary of G is represented by the grey variables.

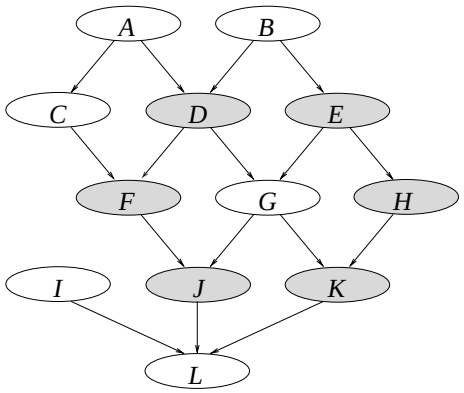


Figure 5.2: A simple Bayesian network with 12 variables. The boundary of G is represented by the grey variables.

This means that when drawing G conditional on the remaining variables, it is only necessary to take the boundary of G into account, making the sampling step much simpler, according to the following theorem :

Theorem 5.2.1. *The probability distribution of each variable v in the network conditional on all other variables, is given by the product :*

$$p(v|U \setminus v) = \alpha p(v|pa(v)) \prod_{w \in ch(v)} p(w|pa(w)),$$

where α is a normalization constant, independent of v.

Proof: Assume that BN is a Bayesian network over a universe $U = \{v_1, \ldots, v_n\}$ representing a joint distribution $P(U)$. Then, the joint distribution of U can be written as :

$$P(U) = P(v_1, \ldots, v_n) = \prod_{i=1}^{n} P(v_i|\mathrm{pa}(v_i)),$$

as P admits recursive factorization.

Now consider a variable $v \in U$, having s children. v appears in exactly $s+1$ factors of the above product; once in $P(v|\mathrm{pa}(v))$ and once in $P(w|\mathrm{pa}(w))$ for each $w \in \mathrm{ch}(v)$. Therefore :

$$P(U) = P(v, U \setminus \{v\}) = P(v|\mathrm{pa}(v)) \prod_{w \in \mathrm{ch}(v)} P(w|\mathrm{pa}(w)) \prod_{u \in U \setminus (v \cup \mathrm{ch}(v))} P(u|\mathrm{pa}(u)).$$

Since v does not appear in the rightmost product (over u), this product can be regarded as a constant α' relative to v, i.e., we can write :

$$P(v, U \setminus \{v\}) = \alpha' P(v|\mathrm{pa}(v)) \prod_{w \in \mathrm{ch}(v)} P(w|\mathrm{pa}(w)).$$

Further, since $P(U \setminus \{v\}) = \sum_v P(v, U \setminus \{v\})$ is also a constant relative to v, we have :

$$P(v|U \setminus \{v\}) = \frac{P(v, U \setminus \{v\})}{P(U \setminus \{v\})} = \alpha P(v|\mathrm{pa}(v)) \prod_{w \in \mathrm{ch}(v)} P(w|\mathrm{pa}(w)),$$

where $\alpha = \frac{\alpha'}{P(v, U \setminus \{v\})}$ is a constant wrt. v. This proves the theorem. □

Thus, when using single-site Gibbs sampling on a Bayesian network, we can use the formula given in Thm. (5.2.1) to draw the new values of variables. All the conditional distributions used in the formula are specified when the Bayesian network is constructed, and all the variables in the boundary of v are observed, when v is drawn, thus we can easily find the marginal distribution of v conditional on the values of the remaining variables. From this marginal distribution, it is easy to draw a new value for v.

5.2.3 Empirical and Mixture Estimates

The objective of the Gibbs sampler is to obtain accurate estimates of the marginal distributions of the unobserved variables conditional on the observed variables. These estimates can be obtained in two different ways.

Empirical estimate: the probability that a variable v is in a certain state s conditional on the observed variables can be estimated by simply counting the number of times v attains the state s, and dividing by the number of iterations. This estimate of the probability is called the *empirical estimate* as it is

constructed based on the *empirical* distribution (i.e., the histogram of data), and it can be expressed formally as follows :

$$\widehat{P}_e(v = s) = \frac{1}{N} \sum_{i=1}^{N} \phi(v^{(i)}, s), \tag{5.10}$$

where N is the number of iterations, $v^{(i)}$ is the value of v drawn at iteration i, and ϕ is defined as :

$$\phi(v^{(t)}, s) = \begin{cases} 1 & \text{if } v^{(t)} = s \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

Mixture estimate: the probability can be estimated by averaging over the probability that the variable v attains a certain state s. This is called the *mixture estimate* as in general it is the average (mixing) of a number of conditional distributions, however, in this case there is only a single distribution. This can be expressed formally :

$$\widehat{P}_m(v = s) = \frac{1}{N} \sum_{i=1}^{N} P(v = s | \, (\mathrm{bd}(v))^{(i)}), \tag{5.12}$$

where $(\mathrm{bd}(v))^{(i)}$ denotes the observed values of the boundary of v at iteration i.

The mixture estimate has been shown by Kalos & Whitlock (1986) to always be the best. Similar points have been made by Pearl (1987), Gelfand & Smith (1990) and Liu, Wong & Kong (1994). Further, empirical results have shown that in some cases, the mixture estimate performs much better than the empirical estimate. This happens in situations where the probability distribution of a variable is partly or completely determined by its neighbours. Imagine that all the neighbours of a variable v are observed, and the conditional distribution of v is $\{0.5, 0.5\}$. With the mixture estimate, this distribution would be obtained immediately in iteration 1, and never deviated from. With the empirical estimate, the sampler would draw state 1 half the times, and state 2 half the times, thus converging towards the correct distribution and requiring several iterations to obtain acceptable accuracy.

Needless to say, the mixture estimate is used in the blocking Gibbs sampler, introduced in the next chapter.

5.2.4 Visitation Schemes

As previously mentioned, one has to choose between different visitation schemes for the Gibbs sampler. These fall into two categories, *random* and *fixed*. Random visitation schemes are, as the name suggests, based on some randomized method for selecting the next component. Fixed visitation schemes follow some fixed ordering of the components.

Furthermore, some visitation schemes fulfill the criterion of detailed balance in Def. (5.1.3) (time-reversibility). If this is the case, much stronger and cleaner theoretical results are available on the convergence rates and efficiency of the sampler. However, lack of time-reversibility does not normally deteriorate the performance of the sampler.

First, we will describe some of the most common visitation schemes :

Deterministic: the most commonly applied scheme. The components are visited in a fixed order, i.e., $T_1, T_2, \ldots, T_n$. A Gibbs sampler following this updating scheme is not reversible.

Reversible deterministic: the components are visited first in forwards ordering, $T_1, \ldots, T_n$, and then in backwards ordering, $T_n, \ldots, T_1$. This visitation scheme induces a reversible Markov chain.

Random sweep: at each update, a component is selected at random and then visited. This, the second most commonly applied scheme, induces a reversible Markov chain.

Random sweep with distinct successive components: this scheme is similar to the *random sweep* visitation scheme, except that successive updated components are required to be distinct. This scheme is also reversible.

Random permutation: at each iteration, a random permutation of the numbers $\{1, 2, \ldots, n\}$ is selected, and the components are visited in this order. This scheme is also reversible.

Random forward-backward sweep: in each iteration, a component is selected at random, excluding the last one updated. Then the sampler scans forward or backward in numerical order, choosing the directions with equal probabilities. This scheme is also reversible.

When choosing between these visitation schemes, one can consider various theoretical results. Roberts & Sahu (1997) have shown that the simple random sweep updating scheme takes approximately twice as many iterations as the simple deterministic updating scheme will take to achieve the same level of accuracy, at least for fairly slow converging problems. The reason for this may be that the random sweep updating scheme may update a component before all its neighbouring components have been updated. This leads to more dependence between the updates. The deterministic updating scheme is better at avoiding this and may often even be tailored to avoid it completely.

Besag et al. (1995) mention that one demerit of the deterministic sweep updating schemes is the potential for significant artificial "drift" among the variables, which may in some situations hinder the mixing of the chain and produce visible directional effects in spatial problems when the order of visiting the components follows their spatial arrangement.

Some of the above visitation schemes have considerable computational overhead, e.g., finding a random permutation of n numbers for large n.

With the blocking Gibbs sampler, the simple deterministic visitation scheme has been selected, as it has very little computational overhead and it has been shown both empirically and theoretically to perform well in most situations. The problems with "drift" in spatial applications are assumed not to be present in the applications of blocking Gibbs in Chapters 16 and 17, and indeed, to the knowledge of the author, have not been reported in any Bayesian network applications.

Chapter 6

The Blocking Gibbs Algorithm

In this chapter, an overview of the blocking Gibbs sampler is given. The components of the sampler are introduced, and will be described carefully in later chapters.

6.1 Outline of the Algorithm

The blocking Gibbs algorithm is basically the same as the general Gibbs sampler. We wish to sample from the joint distribution for $\mathbf{X} = \{X_1, \ldots, X_n\}$ given by $P(x_1, \ldots, x_n\}$ where the domain of the X_i is discrete. With the general Gibbs sampler, each of the components, X_i, can consist of more than one variable, and the blocking Gibbs sampler is indeed a version of the Gibbs sampler where more than one variable is contained in each component.

In the blocking Gibbs terminology, the components are called *blocks*, and we refer to them as B_i instead of X_i. These blocks can be selected using various *block selection* algorithms, presented in Chapter 9. One property, that renders the blocking Gibbs sampler different from any other present version of the Gibbs sampler, is, that the blocks may overlap. This means, that if $x \in B_i$, we can also have $x \in B_j$, i.e., the variable, x, is sampled at least twice in each iteration where one iteration is taken as a complete round of sampling $B_1, \ldots, B_n$.

The blocks are selected to be as large as possible, meaning that the blocks can overlap to a large extent. The joint sampling of many variables is generally thought to be very beneficial, and have the potential of solving crucial problems, i.e., slow mixing and reducibility. This will be discussed further in Chapters 15 and 18. As discussed in the previous chapter, joint sampling of many variables requires that the exact inference method (junction tree propagation) described in Chapter 4 can be applied to the block. Thus the storage requirements of the corresponding junction tree must be sufficiently low.

The actual method for selecting the blocks is based on observing which variables are responsible for the largest reduction of storage requirements for the junction tree when they are conditioned upon and replaced with clones. By conditioning upon these variables (often referred to as the "optimal" variables) and replacing them with clones as described in Section 4.6, we reduce the storage requirements of the junction tree corresponding with the block. The blocks are thus created from the initial junction tree by conditioning upon a sufficient amount of variables, such that exact inference becomes feasible. However, this will be described in great detail in

Chapter 9.

The result of the block selection algorithm, usually results in blocks containing more than 90% of the variables, thus making block updating extremely efficient with very rapid mixing. The performance of blocking Gibbs sampling has been investigated empirically by Jensen et al. (1995) and Jensen & Kong (1996), see Chapters 16 and 17.

Before we give a formal definition of the algorithm, we will provide a few simple examples :

Example 1: The algorithm can be understood informally by Figure 6.1. The figure represents the set of variables in a Bayesian network with the area covered by each set being proportional to the number of variables in each set. The variables in sets 1–6 correspond to the "optimal" variables in this example. A typical number of sets have been shown, showing a realistic construction of blocks; block 1 consists of sets 1 and 7, block 2 consists of sets 2 and 7, ..., block 6 consists of sets 6 and 7. Thus, it is seen that each of the blocks contains the majority of the variables, and that the variables in set 7 are contained in all blocks. Thus, these variables are sampled 6 times in each iteration while the variables in sets 1–6 are only sampled once. The algorithm then proceeds by sampling the blocks, i, one at a time conditional on the remaining variables (the union of sets $1, \ldots, i-1, i+1, \ldots, 6$).

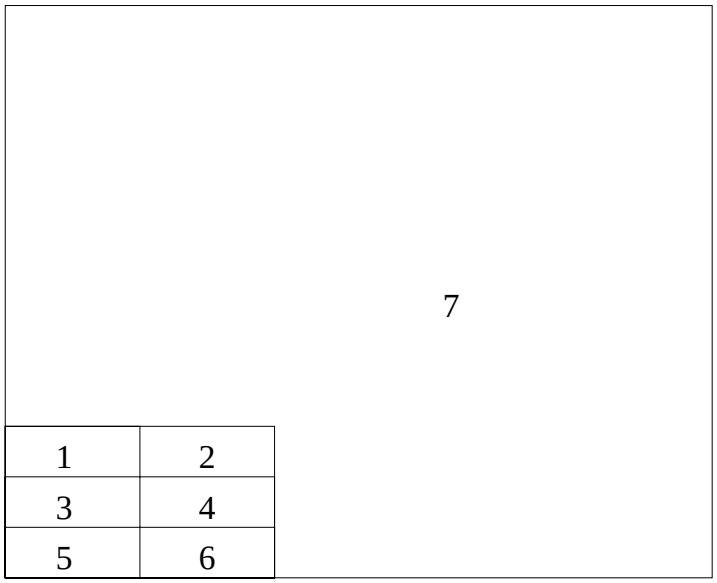


Figure 6.1: The set of variables split into 6 blocks. Block 1 consists of sets 1 and 7, block 2 consists of sets 2 and 7, ..., and block 6 consists of sets 6 and 7.

Example 2: Consider the Bayesian network in Figure 6.2(a). As explained previously we create the blocks by conditioning upon variables that are responsible for a large proportion of the storage requirements of the junction tree, shown in Figure 6.2(b). These variables we can then replace with clones and thus break the connections leading through them as described in Section 4.6. If we assume that the network in Figure 6.2(a) requires too much storage when represented as the junction tree in (b), we have to decide on the variables to

condition upon. In this network, we have one main loop, between e, b, f, and g. In the junction tree, we can see that these four variables are also the variables causing the most complexity in the junction tree, as one or more of them are present in all cliques and separators. Thus, it would be optimal to select these four variables for being conditioned upon. A suggestion for blocks would thus be (with U denoting the entire set of variables):

1. $U \setminus \{e\}$,
2. $U \setminus \{b\}$,
3. $U \setminus \{f\}$, and
4. $U \setminus \{g\}$.

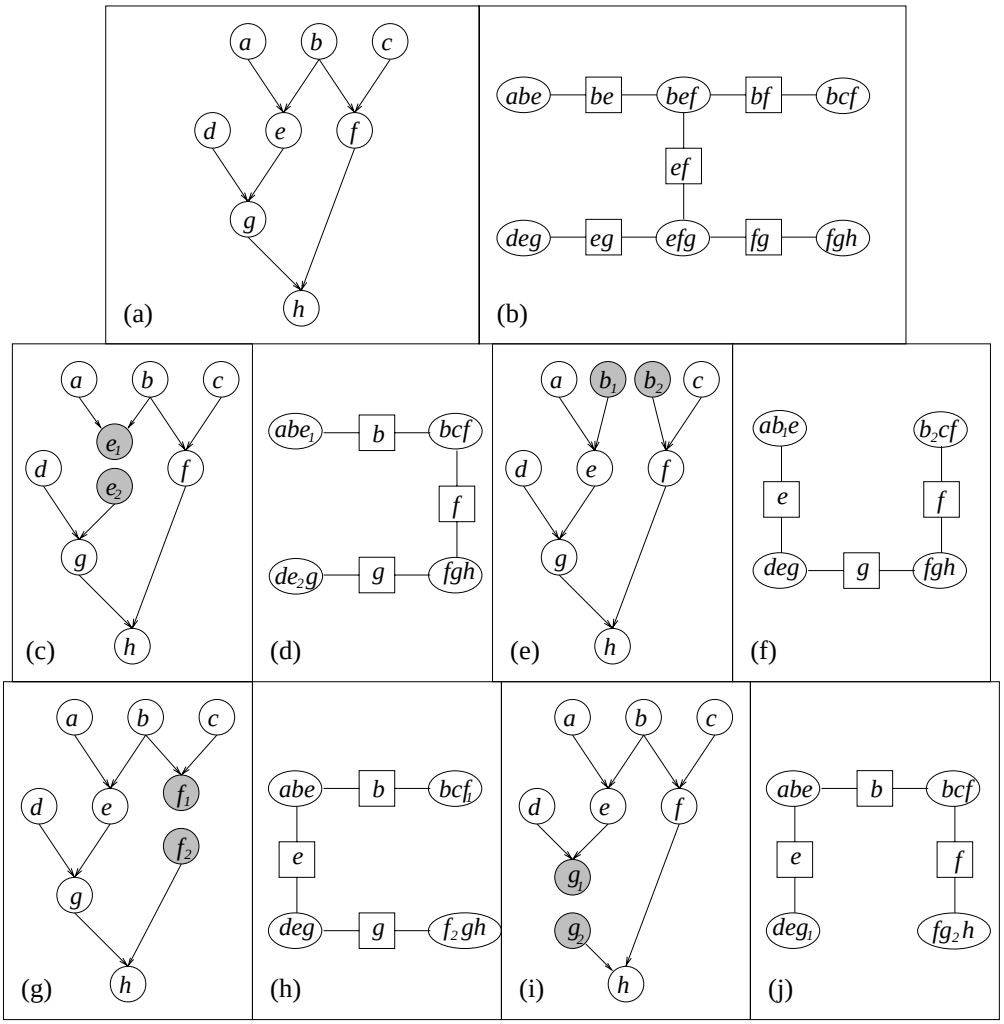


Figure 6.2: The constructed blocks in an example network. (a) shows the original network and (b) its corresponding junction tree. (c)–(j) show the four blocks and their corresponding junction trees. In (c)–(d), (e)–(f), (g)–(h) and (i)–(j), the loops of respectively e, b, f, and g have been broken. The clones of variables that have had their loops broken are grey to denote that they will always be observed when the block is used.

Thus, for instance, with the third block, we can condition upon f and replace it with clones breaking the connections leading through it. Thus, the storage requirements of the junction tree corresponding to each block clearly becomes smaller than that of the initial junction tree. The blocks created by conditioning upon e, b, f, and g, have been shown respectively in Figure 6.2(c)–(d), (e)–(f), (g)–(h), and (i)–(j) as Bayesian network and junction tree. An iteration, t, of the blocking Gibbs sampler with the above blocks thus consists of the following steps :

1. Pick a configuration for $U \setminus \{e\}$ from $P(U \setminus \{e\}|e^{(t-1,3)})$,
2. Pick a configuration for $U \setminus \{b\}$ from $P(U \setminus \{b\}|b^{(t,1)})$,
3. Pick a configuration for $U \setminus \{f\}$ from $P(U \setminus \{f\}|f^{(t,2)})$,
4. Pick a configuration for $U \setminus \{g\}$ from $P(U \setminus \{g\}|g^{(t,3)})$,

where $v^{(t,i)}$ indicates the ith sampled value for v at iteration t. This notation is introduced to indicate that e, b, f, and g, are sampled three times in each iteration. The remaining variables, a, c, d, and h are sampled in each block, i.e., four times in each iteration. So, when we write, e.g., "Pick a configuration for $U \setminus \{f\}$ ..." we really should have written "Pick $a^{(t,3)}$, $b^{(t,2)}$, $c^{(t,3)}$, $d^{(t,3)}$, $e^{(t,2)}$, $g^{(t,3)}$, and $h^{(t,3)}$ jointly ...".

In general, however, the block construction is much more complicated, and along with this, some other preliminary steps must be completed before the sampler can begin. The full algorithm can be defined formally as follows (where b denotes an instantiation of a set of variables B):

Definition 6.1.1 (Blocking Gibbs sampling algorithm). Assume that we wish to sample from the joint distribution for $U = \{v_1, \ldots, v_n\}$ given by $P(v_1, \ldots, v_n)$ where the domain of v_i is discrete. The universe, U, and its distribution P is furthermore represented by a Bayesian network, BN.

The algorithm proceeds as follows :

a. We condition upon the variables with initial evidence, A_{evidence} (see Section 4.6) replacing them with clones and breaking the connections leading through them, thus resulting in the reduced network, BN'.

b. The barren variables, A_{barren}, are removed (see Chapter 10) resulting in the further reduced network, BN''. The reduced universe is called $U'' = U \setminus A_{\text{barren}}$.

c. k blocks, $B_1, \ldots, B_k$, are selected in BN'', such that $B_1 \cup \ldots \cup B_k = U''$, and such that each block, B_i, contains as many variables as possible and allows joint updating (see Chapter 9).

d. A legal starting configuration is found for $U'' \setminus (A_{\text{evidence}} \cup B_1)$ henceforth denoted as A_1, i.e., $a_1^{(0)}$ (see Chapter 8).

e. A number of warm-up iterations are performed, similar to the real iterations, except that samples are not used for estimation (see Section 5.1).

f. The initial evidence is inserted in the clones of the variables in A_{evidence}.

g. In each iteration t, for $t = 1, \ldots, N$, do :

1. Visit block 1:

- If this is the first iteration ($t = 1$), let $a_1^{(t)}$ be given by $a_1^{(0)}$ found in step **d**, otherwise let $a_1^{(t)}$ be given by the instantiation of the last visited block, $a_k^{(t-1)} \cup b_k^{(t-1)}$.
- Pick $b_1^{(t)}$ from the conditional distribution for B_1 given $a_1^{(t)}$.
- The barren variables, A_{barren}, are forward sampled, obtaining $a_{\text{barren}}^{(t,1)}$, see Chapter 10.

2. Visit block 2:

- Let $a_2^{(t)}$ be given by the instantiation of the last visited block, $a_1^{(t)} \cup b_1^{(t)}$.
- Pick $b_2^{(t)}$ from the distribution for B_2 given $a_2^{(t)}$.
- The barren variables, A_{barren}, are forward sampled, obtaining $a_{\text{barren}}^{(t,2)}$.

⋮

k. Visit block k :

- Let $a_k^{(t)}$ be given by the instantiation of the last visited block, $a_{k-1}^{(t)} \cup b_{k-1}^{(t)}$.
- Pick $b_k^{(t)}$ from the distribution for B_k given $a_k^{(t)}$.
- The barren variables, A_{barren}, are forward sampled, obtaining $a_{\text{barren}}^{(t,k)}$.

The objective of the blocking Gibbs sampler is to obtain estimates of the marginal probabilities of the variables in the network conditional on the evidence. As described in Section 5.2.3, we can estimate these marginals using either the *empirical* or *mixture* estimates. As the mixture estimate is the optimal, it is used in the blocking Gibbs sampler. Thus, when estimating the conditional probability of a variable, v, being in a state, s, i.e., $P(v = s)$, we average over the probability that the variable v is in the state s in each iteration. When we draw each block, $b_i^{(t)}$ from $P(B_i|a_i^{(t)})$, and $v \in B_i$, we can get the conditional probability that $v = s$ at iteration t from $P(B_i|a_i^{(t)})$ by marginalizing.

Liu et al. (1994) have shown that the gain from using the mixture estimate is small if the chain is mixing slowly, and the gain can be very substantial if the chain is mixing fast. This was shown in the setting with two non-overlapping blocks and is probably not always true in a general setting with multiple overlapping blocks. However, as seen in Chapter 16, the blocking Gibbs sampler mixes faster than the single-site Gibbs sampler in the examined cases and it can be expected that the gain of using the mixture estimate is substantial.

As variables are present in different numbers of blocks, we get an unequal amount of samples for them. Considering Figure 6.1, some variables are sampled 6 times in each iteration and others only once. In Chapter 7 it is shown that we can use all samples for all variables when estimating probabilities, i.e., we do not have to limit ourselves to using one sample for each variable in each iteration.

From Def. (6.1.1) it is seen that the barren variables, A_{barren}, are sampled once for each block, i.e., k times in each iteration. This should be done, if the algorithm for sampling these variables, *forward sampling*, is fast compared with the exact inference method used for the joint updating of the blocks, and further, if most of the leaf variables connected with the barren variables are sampled each time a block is visited. This happens to be the case in the implementation of the blocking Gibbs algorithm used in this thesis, see Chapter 10. If this was not true, it would be more optimal only to sample the barren variables once in each iteration, e.g., after step **g**.

Also, we see from the definition that the same blocks are used over and over again in each iteration. This is one of the basic benefits of blocking Gibbs, as the very time-consuming block construction is performed only once, and, which cannot be seen in the definition, the time-consuming compilation of each block into its corresponding junction tree is also performed only once, before sampling begins. The latter is of course possible as the structure of the block remains the same throughout the run, with different evidence being inserted in the clones of variables in each iteration, transferred from the last block visited.

According to Def. (6.1.1), the blocking Gibbs algorithm basically consists of two parts, first, a block selection part where the blocks are constructed in an optimal way, and second, the sampling part where the blocking Gibbs sampler samples one block at a time conditional on the remaining variables. The various steps will not be described in detail in this chapter, but will each be covered in later chapters.

Chapter 7

Irreducibility of Blocking Gibbs

In this chapter it will be shown that whenever the single-site Gibbs sampler is ergodic, so is the blocking Gibbs sampler.

The single-site Gibbs sampler consists of a large number of transition probabilities, $\mathbf{T_1}, \ldots, \mathbf{T_n}$, one for each variable. Then, the Markov chain with transition probabilities $\mathbf{T} = \mathbf{T_1}\mathbf{T_2}\cdots\mathbf{T_n}$ is stationary with an invariant distribution. Furthermore, if this Markov chain is irreducible, then the single-site Gibbs sampler is ergodic. Whether the single-site Gibbs sampler is irreducible is a question that must be answered in each individual problem. For instance, in the area of pedigree analysis it has been proven by Sheehan & Thomas (1993) that a subclass of problems that do not fulfill the positivity condition, are irreducible, see Chapters 16, 17 and 18.

In this chapter, however, we will assume that the single-site Gibbs sampler has been proven irreducible and thus ergodic. First, if the single-site Gibbs sampler is irreducible, this means that it is possible to get from any state to any other state in a finite number of steps where each step results in a change of state in only a single variable. Clearly, if more than one variable is updated jointly in each step, the chain is still irreducible. Thus, if the single-site Gibbs sampler is irreducible, the blocking Gibbs sampler will also be.

If we denote the transition probabilities of the blocking Gibbs sampler with k blocks, $\mathbf{T}_1, \ldots, \mathbf{T}_k$, then we also have to show that each of the $\mathbf{T}_i$ leaves the desired distribution invariant. This is as simple as for single-site Gibbs sampling. In $\mathbf{T}_i$ some of the variables, B_i (the block), are updated, and some of the variables, $A_i = U \setminus B_i$, are left unchanged. Thus, for the variables in A_i the desired marginal distribution is certainly invariant. Further, $\mathbf{T}_i$ draws the new states for the variables in B_i jointly given A_i from the conditional distribution that is defined to be that which is desired (P). Together this ensures that the joint distribution of the variables remains the same after all the $\mathbf{T}_i$ have been applied.

Finally, if we construct a Markov chain with transition probabilities, $\mathbf{T} = \mathbf{T}_1 \cdots \mathbf{T}_k$, this chain is clearly stationary. Thus, as the blocking Gibbs sampler induces a stationary, irreducible Markov chain with an invariant distribution, it is *ergodic*. Of course, this only holds in the case where the single-site Gibbs sampler is also irreducible. We will later in Chapter 16 show that the blocking Gibbs sampler is rendered irreducible in many cases where the single-site Gibbs sampler is not.

For the single-site Gibbs sampler specified with transition probabilities $\mathbf{T} =$

$\mathbf{T}_1 \cdots \mathbf{T}_k$ we can use the sample in each iteration to estimate the probabilities of the variables. In each iteration, we basically get one new value (empirical estimate) or conditional probability distribution (mixture estimate) for each variable to use for the estimation.

When doing a single run with the blocking Gibbs sampler with k blocks, this can be viewed as k overlapping stationary chains running simultaneously with transition probabilities :

1. $\mathbf{T}_{\text{chain } 1} = \mathbf{T}_1 \cdots \mathbf{T}_k$,
2. $\mathbf{T}_{\text{chain } 2} = \mathbf{T}_2 \cdots \mathbf{T}_k \mathbf{T}_1$,
3. $\mathbf{T}_{\text{chain } 3} = \mathbf{T}_3 \cdots \mathbf{T}_k \mathbf{T}_1 \mathbf{T}_2$,

$\vdots$

$k-1$. $\mathbf{T}_{\text{chain } k-1} = \mathbf{T}_{k-1} \mathbf{T}_k \mathbf{T}_1 \cdots \mathbf{T}_{k-2}$,

k. $\mathbf{T}_{\text{chain } k} = \mathbf{T}_k \mathbf{T}_1 \cdots \mathbf{T}_{k-1}$.

For each of these stationary chains we can use the sample at each iteration to estimate the probabilities of the variables, and, as with single-site Gibbs, we get one new value for each variable at each iteration. Thus, when estimating the probability distribution of a variable v, we can for instance use chain i, and thus use the value at each iteration to estimate the conditional distribution for v. However, this is wasteful as v may be sampled several times during each iteration in the intermediate $\mathbf{T}_i$'s.

When estimating v, it is legal to combine the values obtained at each iteration in each of the k overlapping chains, as this basically corresponds to combining the results from k different runs. Thus, we can use all the samples obtained for each variable in the intermediate steps $\mathbf{T}_1, \ldots, \mathbf{T}_k$ for the estimates. As some variables are not sampled in each step, $\mathbf{T}_i$, there is no purpose in using the samples of these variables at the steps where they are not sampled. Obviously, we will get better estimates for these variables by leaving out the values of the steps, $\mathbf{T}_i$, where the variable has not been sampled.

Thus, in this chapter we have shown that if the single-site Gibbs sampler is ergodic, so is the blocking Gibbs sampler, and further, we can use all samples of the variables for estimation purposes.

However, if the single-site Gibbs sampler is not irreducible, we will not immediately have any guarantee that the blocking Gibbs sampler is. This entirely depends on the constructed blocks, and is further discussed in Section 9.4.

Chapter 8

Finding the Starting Configuration

This chapter corresponds with (Jensen 1996a) and will deal with the general problem of finding a legal configuration in very large and complex Bayesian networks which corresponds to the specific problem of determining a legal starting configuration for an MCMC method. As previously explained, the storage requirements for exact inference in Bayesian networks can grow to astronomical amounts. For the Bayesian network in Figure 8.1, the storage requirements are more than 10^{160} MB. This network represents a pedigree of a population of Greenland Eskimos collected by Gilberg, Gilberg, Gilberg & Holm (1978).

Thus, it is often not possible to find a legal configuration using exact methods. Markov chain Monte Carlo methods cannot be used to solve this problem, as they themselves require a legal configuration to get started.

For the blocking Gibbs sampling algorithm described in Chapter 6, the problem is not as acute as with other MCMC methods, as the block selection algorithm described in Chapter 9 is usually able to place more than 90% of the variables in each block. This means, that for the blocking Gibbs sampler, we usually only have to find a legal starting configuration for less than 10% of the variables, i.e., the variables that we are drawing the block conditional on. Further, these variables are usually located in different parts of the network, and are thus only rarely related, making it easier to select a legal configuration. However, still, with the very large networks that blocking Gibbs is aimed at, it is problematic to find a starting configuration.

Very little research has been done on this problem so far. However, it is relatively simple to show that finding a legal configuration in a general Bayesian network with evidence inserted on some variables is NP-hard, as this corresponds to the general *satisfiability problem* in the propositional calculus. In (Jensen 1996b), this is considered in Exercises 3.16 and 4.14.

Until now, researchers have applied nondeterministic methods such as forward sampling (see Chapter 10) or the importance sampler of Sheehan & Thomas (1993). If the Bayesian network contains many observed variables, the forward sampler may run for years without coming up with a legal configuration. A simple calculation can quickly illustrate this.

Imagine, in Figure 8.1, that 100 bottom level variables have been observed and that each of these have three states. Now, the forward sampler has a probability of $\left(\frac{1}{3}\right)^{100} = 1.9 \cdot 10^{-48}$ of finding a legal configuration in one iteration. Another small calculation shows that on a fast (at the present level of technology) computer the

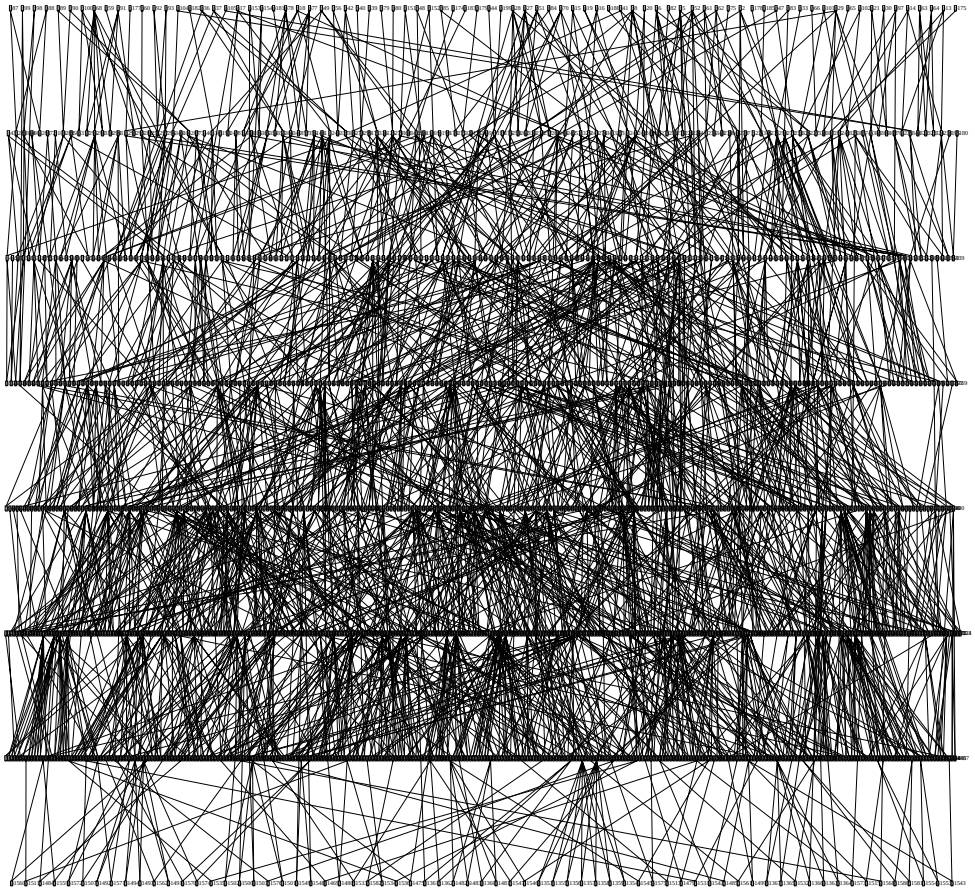


Figure 8.1: A Bayesian network representation of a large pedigree. The pedigree contains 1614 individuals of the Greenland Eskimo population. The pedigree was pretty printed using methods described by Thomas (1988) and implemented by the author.

forward sampler must run for more than 10^{30} years to have a realistic chance of finding a legal configuration.

The rejection sampler of Sheehan & Thomas (1993) has been used successfully for finding legal configurations in large complex Bayesian networks but the generality of the method is not known.

These methods for finding a legal configuration are both nondeterministic. In the following, an almost deterministic algorithm will be presented. The algorithm will not be proven deterministic but empirical results will show that in practice this seems to be the case. Furthermore, the usefulness of the algorithm is high, as it is simple and very easily implemented. Finally, the complexity of the algorithm will be outlined, and a discussion will conclude with directions for further research.

8.1 The Algorithm

The underlying idea of the algorithm is to take advantage of the idea of *conditioning* to break the Bayesian network into manageable pieces. Conditioning has been

described in detail by Pearl (1988) and is also outlined in Section 4.6.

In this algorithm, the idea of conditioning is applied in a quite different fashion as that intended by Pearl (1988) and one of the basic assumptions of conditioning is violated. To split a variable into a number of clones to break loops is only legal if the variable is observed. In this algorithm, we will perform this splitting into clones even if the variable is only partly observed, i.e., contains *soft evidence*. Using soft evidence to break loops is essentially illegal as this type of evidence does not render the parents independent of the offspring, or one offspring independent of another. However, as we will see, the algorithm attempts to maintain their dependency by transferring beliefs between independent variables.

We operate with two Bayesian networks in the algorithm. BN_0 is the original network that we want to find a legal configuration for. BN_e is an "exploded" version in which all variables are assumed observed and replaced by their clones. BN_e thus consists of a large number of *nuclear families*[1] as seen in Figure 8.2 which depicts the "exploded" version of the network in Figure 4.8a. It follows that exact inference in BN_e is always possible for arbitrary BN_0. The *feasible* states of a variable v in BN_0 are the states that are currently possible with the current evidence in the system (soft evidence on clones of v, and initial evidence on other clones in the network). The goal of the algorithm is to narrow down the feasible states of all variables sufficiently that no states illegal due to evidence are among them, then selecting one of these states and moving on to the next variable :

1. BN_0 is "exploded" into BN_e.
2. Evidence is inserted into all clones, i.e., if v in BN_0 is observed, this evidence is inserted into all clones of v in BN_e.
3. For each variable, v, in BN_0, perform substeps (a), (b) and (c) :
   (a) While the feasible states of any variable in BN_0 can be narrowed further down, perform substeps (i) and (ii) :
      i. For each variable, w, in BN_0, perform substeps (A) and (B) :
         A. Multiply the marginal beliefs of the clones of w together.
         B. Insert the normalized resulting belief table (the feasible states of w) as soft evidence in each of w's clones.
      ii. Perform exact inference in BN_e (for instance with the junction tree propagation method), propagating the effects of the soft evidence of last step further away.
   (b) Multiply the marginal beliefs for v's clones together to find the feasible states of v.
   (c) One of the feasible states of v is selected and inserted as hard evidence in each of v's clones.

This is not the complete algorithm, however. It may occur, despite the continuing narrowing down of feasible states that an illegal state (illegal given the initial evidence) is inserted at step 3c. This will later be detected at step 3(a)iA where an all zero belief table will show up for some variable. In this case, an extra backtracking step must be added to the algorithm after this step :

Backtrack: if an all zero belief table is found, then step back to the previously instantiated variable and try its next feasible state. If all feasible states for this variable have already been "tried", then backtrack to the previous, etc.

[1] Here, offspring can have one or more parents.

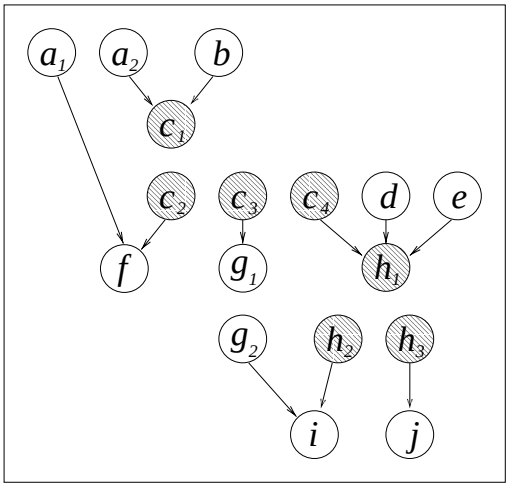


Figure 8.2: The "exploded" version of the Bayesian network in Figure 4.8a.

Thus we have to remember the order in which the variables are determined in step 3c, and in addition the feasible states that have been "tried" for each variable. Usually only one state is "tried" for each variable. However, if backtracking has occurred, more than one of v's states have been "tried". This information must be stored such that if backtracking at v ever occurs again, the algorithm will know which states have already been "tried".

8.2 Results

In this section an empirical investigation of the algorithm of the previous section will be performed. We will attempt to disclose the complexity of the algorithm, along with some statistics on the occurrence of backtracking.

First, we have applied the algorithm to a number of linkage analysis problems in various sizes and of various types, some constructed and some real-world problems. These linkage analysis problems are highly complex and difficult to handle for any inference algorithm as they often contain a large number of variables and loops. Furthermore, the conditional probability tables of the variables contain a large number of zeros, invalidating samplers updating one variable at a time, due to the large number of noncommunicating classes (see Section 9.4 for a definition). The goal of linkage analysis is to establish *linkage* between a marker (a gene of known location) and a disease gene, and thus effectively find the location of the disease gene. This can be done by representing the entire pedigree along with variables for the marker and disease genes and their recombination in a Bayesian network, and then applying some MCMC method, e.g., a Gibbs sampler, see Chapter 17.

The first pedigree (Figure 8.3) is a highly inbred human pedigree affected with a rare heart-disease (the LQT syndrome) originating from professor Brian Suarez. The pedigree (henceforth termed Problem 1) contains 73 individuals, but the Bayesian network representation of the linkage problem contains 903 variables (see (Kong 1991, Jensen & Kong 1996) and Section 13.2 for descriptions of this representation), many of which are observed. Similarly, in the remaining examples, the Bayesian network that the algorithm is applied to, contains several times more variables than the number of individuals in the pedigree, see Table 8.1. For further information

on the genetics terminology used in the chapter, see Chapter 12.

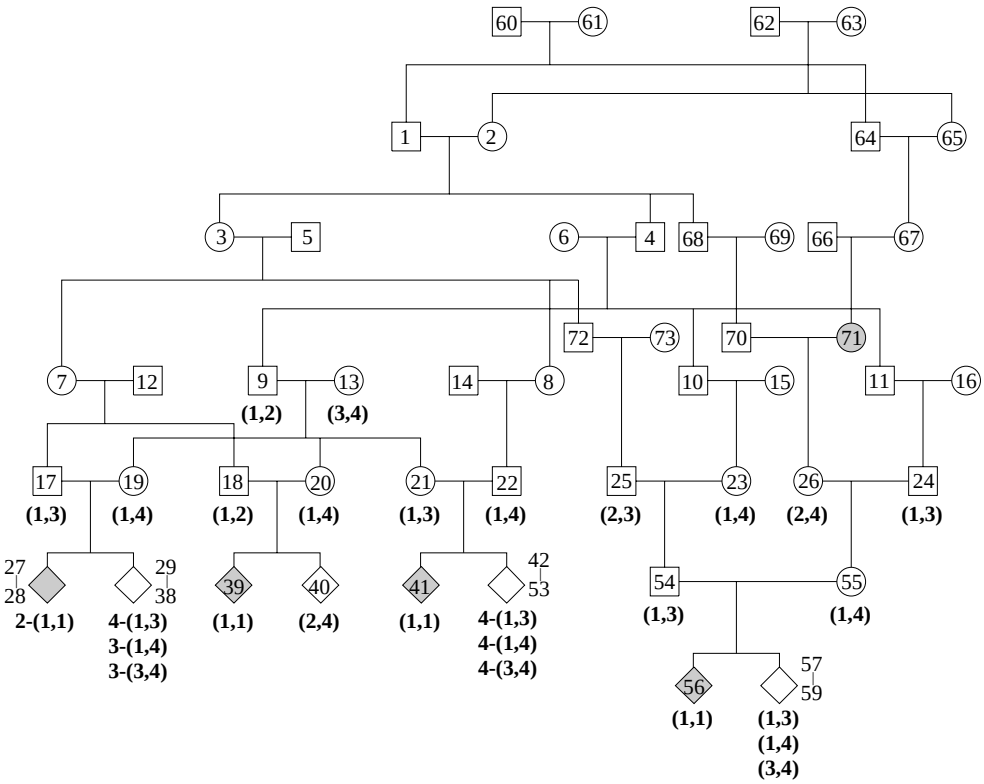


Figure 8.3: The LQT pedigree. The special marriage node graph representation is used in the figure. Marriages and individuals are each depicted by nodes on the graph and the individual nodes are square for males, circular for females and diamond-shaped for unmarried offspring of unknown sex. Individuals are shaded if they have data. In this case, marker genotypes are in brackets, and darker diamonds denote affected offspring. Some diamonds represent several individuals, e.g., 29–38 represent the 10 individuals $29, \ldots, 38$, four of which have the genotype $(1, 3)$, three of which have the genotype $(1, 4)$ and three of which have the genotype $(3, 4)$.

Problem 2 is shown in Figure 8.1. In this problem it is attempted to establish linkage between the ABO and MN bloodgroups. The pedigree contains 1614 individuals, but the Bayesian network representation of the linkage analysis problem contains 19762 variables, many of which are observed. In both Problem 1 and 2, evidence is located all over the pedigree with the majority in the lower part.

Problems 3 and 4 are reduced versions of Problem 2. In Problem 3, the top-most and bottom-most generations of Problem 2 have been removed. In Problem 4, the top-most and bottom-most generations of Problem 3 have been removed.

Problem 5 is shown in Figure 18.4 on page 131. It is a highly inbred pedigree with problems of reducibility, and it was constructed particularly to highlight problems with MCMC methods in pedigree analysis, see (Jensen & Sheehan 1997) and Chapter 18. Noncommunicating classes for MCMC methods updating one variable at a time are created on individuals 2 and 3, and on 11 and 12. The homozygous

individuals 7 and 10 forces their respective offspring, 11 and 12, to carry an A-allele each. And, the common offspring of 11 and 12, individual 13, forces them to carry a B and a C allele. Thus, 11 and 12 only have two legal configurations, $(g_{11} = AB, g_{12} = AC)$ and $(g_{11} = AC, g_{12} = AB)$. Similarly, the noncommunicating configurations of 2 and 3 can be found. It will be impossible for any single variable updating scheme to jump between these configurations. It is assumed that the presence of the noncommunicating classes will cause the algorithm of Section 8.1 trouble, forcing it to backtrack more often.

Problem 6 is shown in Figure 18.6 on page 132. This pedigree is not inbred like the one in Problem 5, but still there are problems with noncommunicating classes. Like Problem 5, Problem 6 is a constructed pedigree. Problem 6 is also discussed in the context of determining the noncommunicating classes by Jensen & Sheehan (1997).

Problem 7 is shown in Figure 8.4. This inbred pedigree is also constructed.

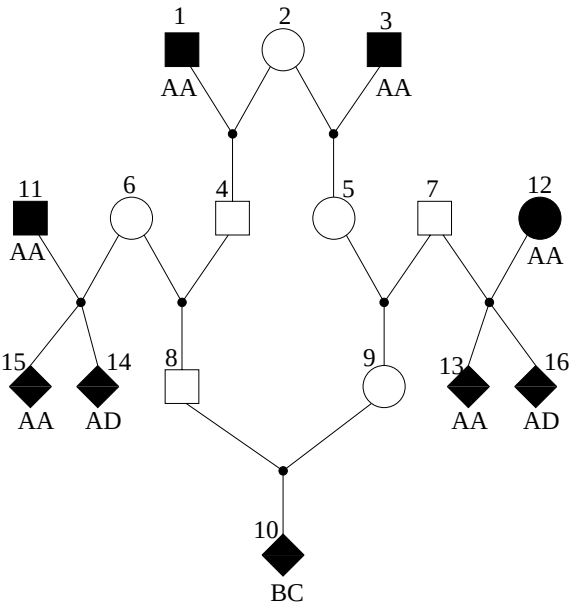


Figure 8.4: Another small inbred pedigree.

A simple implementation of the algorithm described in the previous section has been implemented and run on the seven test problems. The results are shown in Table 8.1.

We can make a number of observations from these results. First, in the real world examples there are no backtracking. This occurs only in the constructed Bayesian networks. This is a vague indication that backtracking may occur rarely in real world networks.

Second, we can see that the most backtracking occurred in Problem 5 (Figure 18.4 on page 131) which is actually the smallest network with only 121 variables. However, this pedigree has been constructed especially to cause problems for the algorithm. We will now analyze a situation in which backtracking occurs.

Due to the structure of this pedigree, configurations are forced upon individuals due to the information on their relatives. Here, individuals 2 and 3 must each carry one A-allele, enforced by their respective homozygous offspring, 5 and 6.

Problem	#Variables	Complexity	Time	Time/Variable	#Backtracks
1	903	$2.2 \cdot 10^8$	2.83 m	0.19 s	0
2	19762	$7.0 \cdot 10^{159}$	32.37 h	5.90 s	0
3	7829	$9.8 \cdot 10^{55}$	3.96 h	1.8 s	0
4	3513	$2.9 \cdot 10^{12}$	51.1 m	0.87 s	0
5	121	$4.0 \cdot 10^5$	5.8 s	0.048 s	8
6	149	$4.0 \cdot 10^6$	14.5 s	0.097 s	2
7	153	$4.7 \cdot 10^5$	9.5 s	0.062 s	3

Table 8.1: Results of the algorithm for the seven linkage problems. *Complexity* is the amount of storage required to handle this problem exactly. *Time* is the time it took for the algorithm to find a legal configuration. All results are averages over 10 runs.

Furthermore, the two alleles B and C, carried by individual 13 must originate from 2 and 3, each of which can carry only one of them. This forces 2 and 3 to be in one of two legal configurations : $(g_2 = AB, g_3 = AC)$ or $(g_2 = AC, g_3 = AB)$, where g_i denotes the genotype of i.

Imagine that the algorithm visits individual 2 and sets this variable to genotype AB. Now, even with this information the algorithm is not able to narrow down 3's genotype to AC which is its only legal state. It is then possible for 3 to be set to, e.g., AA, causing problems for the subsequent narrowing down. The algorithm thus has to backtrack. If, in another run, individual 2 was first set to genotype AA. Then, the subsequent narrowing down would not detect the mistake and would continue setting new variables until a situation arises where it has to backtrack. In this situation, the algorithm may have to backtrack through several variables to get into a legal configuration with $g_2 = AB$ or $g_2 = AC$.

The problem of backtracking is likely to occur whenever there are noncommunicating classes in the Bayesian network such as the above described. It is the author's belief that this occurs rarely in networks representing real-world problems and even when it occurs, it is possible to modify the algorithm such that it handles the situation better. This can be done by handling the variables in a specific order instead of a random, such as outlined in the description of the algorithm. The variables could be ordered such that variables that are connected in the Bayesian network are treated in succession. This would ensure that the algorithm would rarely have to backtrack far to get into a legal configuration.

The above is of course based on the belief that the effects of an observed variable usually do not travel very far from the variable itself. Of course, the beliefs of all variables in the network may be modified slightly based on the new observation, but states are usually only rendered impossible for variables in the local neighbourhood. In all types of Bayesian networks where this property holds, the algorithm will be efficient.

8.3 Complexity of the Algorithm

The complexity of the algorithm can be analyzed by looking at the description in Section 8.1. It is :

$$\mathcal{O}(sn^2), \tag{8.1}$$

where n is the number of variables in the network, and s is the average number of states. The first n is for the main loop which performs its steps for each variable in the network. $s \cdot n$ is for the next loop which performs its internal steps until no feasible states for any variable can be narrowed further. As there are $s \cdot n$ states in all, in the limit this loop can run $s \cdot n$ times. Each time one narrowing down step has been performed, exact inference is performed in the "exploded" network. This operation has complexity $\mathcal{O}(n)$ as the network consists of nuclear families only. Thus, the complexity is $\mathcal{O}(n(sn+n))$ which reduces to $\mathcal{O}(sn^2)$.

In Figure 8.5, the results from Table 8.1 are shown as a graph. The x-axis represents the number of variables and the y-axis represents the time measured in seconds. Both axes are represented in log-scale to determine whether 8.1 holds approximately. If the algorithm has polynomial complexity $(a \cdot n^b)$, the points in Figure 8.5 will be oriented on a line. As we see, this is in fact the case. The slope of the fitted line is approximately 2, meaning that for the examined cases, the complexity was very close to that of Eq. (8.1).

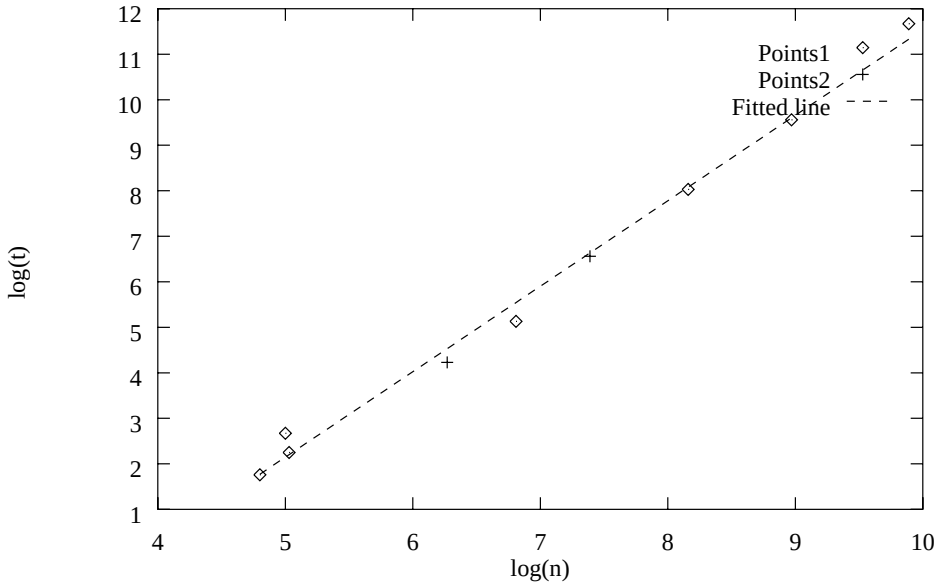


Figure 8.5: The points from Table 8.1 are plotted in the above figure. The x-axis represents the number of variables, and the y-axis represents the time measured in seconds. Both axes are in log-scale. The best fit to the points is shown as a line. The diamond points (*Points1*) are taken from Table 8.1 and the cross points (*Points2*) are some extra results, included to get some more basis for the fitting of the line.

8.4 Summary

We have shown with empirical results that the outlined algorithm is indeed quite efficient with a complexity of $\mathcal{O}(sn^2)$. Furthermore, the algorithm is easily understandable and implemented. One of the weak points of the algorithm is that the amount of backtracking that it will have to do in a general setting is uncertain. Here, we can only provide vague indications that it will rarely have to backtrack much in real-world problems, as the noncommunicating classes that enforce back-

tracking, seem to occur rarely. Furthermore, the algorithm can easily be modified to counter this by handling the variables in a more intelligent order.

The implementation used to document the complexity of the algorithm on the example Bayesian networks was very inefficient, thus causing the large running times shown in Table 8.1. However, the algorithm could quite easily be optimized many-fold to provide legal configurations for most networks fast. Indeed, Heath (1997) independently developed a similar algorithm in the setting of pedigree analysis and peeling which was implemented far more efficiently.

Another way to optimize the algorithm would be to handle larger parts of the network exactly. It is perfectly possible and not hard to implement an algorithm that only "explodes" a minimal proportion of the Bayesian network, a proportion just sufficient to be able to handle the partly "exploded" network with an exact inference method. This will lead to even less backtracking as many variables would be sampled jointly. However, the algorithm would probably become slower in most "easy" networks where little backtracking is necessary.

As a future investigation, the author would like to look further into the situations in which the algorithm has to backtrack. It would be interesting to identify this class of Bayesian networks.

Chapter 9

Block Selection

In this chapter, the principles and details of *block selection* will be described. As mentioned in Chapter 6, before the blocking Gibbs algorithm can start sampling, it must go through the process of selecting the blocks. We want to be sure that the blocks contain as many variables as possible and fulfill some other criteria in an optimal way. In the following, these criteria and some block selection principles will be presented.

9.1 Criteria of Block Selection

We have various criteria that we want the blocks to fulfill :

1. The blocks should contain as many variables as possible. Their size is governed by memory capacity. The larger the blocks, the faster blocking Gibbs will mix. In the limit where the entire network is included in the same block, we get exact simulation. In the opposite limit where only one variable is included in each block, we get single-site Gibbs. This criterion will be discussed further in Section 9.2.

2. We want all variables to be sampled approximately equally often. We can construct blocks such that some variables are sampled rarely, and others much more often which is probably not optimal wrt. the mixing rate. Section 9.3 addresses this problem.

3. The blocking Gibbs sampler should be irreducible. This can be obtained by ensuring that all states in the induced Markov chain *communicate*, thus requiring certain highly correlated variables to be sampled jointly. This problem is outlined in Section 9.4.

9.2 Criterion 1: Large Blocks

To construct a block that contains as many variables as possible, we use the fact that conditioning on certain variables and replacing them with clones (Section 4.6) breaks loops in the network, creating a network with fewer loops. This network, given that enough variables are being conditioned upon and thus enough loops are broken, then becomes feasible for exact inference, allowing us to sample the variables of the block jointly (Section 4.4). In the following we will refer to the operation of conditioning upon a variable and replacing it with clones thus blocking

the connections between its parents and offspring leading through the variable, as described in Section 4.6, as simply *conditioning upon the variable*.

In the following, we will present a method for block selection based on selecting the blocks by conditioning upon the variables causing large reductions in storage requirements in the initial junction tree and then updating the junction tree correspondingly, see Section 9.2.1. Then, in Section 9.2.2, the method for calculating the reduction in storage requirements of variables is presented.

9.2.1 Block Selection Method

In the following, the terms *junction tree* and *block* are used interchangeably. With this method, a block is created from the initial junction tree by conditioning upon variables until the storage requirements have been reduced sufficiently. Strictly, this means that the block is a junction tree where a number of variables have been conditioned upon.

We construct a block B by first letting it be equal to the initial junction tree, and then conditioning upon the variables A that should not be part of the block, replacing them with clones. Thus, the junction tree representing the block contains the variables in B and the clones of the variables in A. When sampling B we then condition upon the variables in A by inserting evidence in their clones, thus enabling us to break a sufficient number of loops as described in Section 4.6. The variables in B can then be sampled jointly using the *random propagation* method described in Section 4.4.

As the same variables in the block are conditioned upon in each iteration we can initially replace these variables with clones, breaking some loops, and then use the reduced junction tree in all subsequent iterations by just inserting new evidence in the clones each time the block is visited.

However, how should we select the variables that result in the greatest reduction in storage requirements when they are conditioned upon? In Section 9.2.2 a method for doing this is presented.

The block selection strategy of the iterative selection of variables yielding large reductions in storage requirements when conditioned upon can be described by means of pseudo code. In the simple program in Figure 9.1, S is the storage capacity of the computer (RAM, hard disk, etc.). In this program, we iteratively find the optimal variable, v^*, wrt. reduction in storage requirements, and condition upon v^* in the junction tree. Subsequent reductions in the junction tree necessary for maintaining the junction tree property and keeping the tree minimal are also described in Section 9.2.2. Finally, the reduced junction tree is assigned to the block, B.

This is not perfectly optimal, however, as the algorithm is local, i.e., it only looks one step ahead when selecting the next variable. If, for example, v_1 and v_2 are the first two variables selected by the algorithm, it is possible that another combination of variables may yield a larger reduction in storage requirements. To create a completely optimal algorithm, we would thus have to consider all possible orderings of the variables. As it is NP-complete to find the best of these orderings, obviously, this is completely infeasible.

There is one other reason why this algorithm is not perfectly optimal. When updating the junction tree after conditioning upon a variable, this is done by considering only the junction tree. Ideally, the conditioning upon the variable and the subsequent replacement with clones and breaking of loops should be performed in the corresponding Bayesian network and then, at each step, the junction tree should be created from this network through finding a new triangulation. This junction tree will in most cases have smaller storage requirements than the one obtained

```
JT ← initial junction tree
find variable, v*, with largest reduction in storage
    requirements for JT
while storage_req(JT) > S do:
        condition upon v* in JT and update JT correspondingly
        find variable, v*, with largest reduction in storage
            requirements for JT
end while
B ← JT.
```

Figure 9.1: The algorithm for selecting a block, B.

```
while at least one block is too large, do:
    for each block, B_i, for i = 1, ..., k, do:
        find variable, v*, with largest reduction in storage
            requirements for B_i
        ensure that v* has not been conditioned upon in all other blocks
        condition upon v* in B_i and update B_i correspondingly
    end do.
end while.
```

Figure 9.2: The algorithm for selecting k blocks.

by updating the junction tree directly, as a new triangulation is performed at each step. However, to perform a triangulation at each step of the algorithm in Figure 9.1 would be extremely time-consuming, and the method for simulating the conditioning upon a variable within the junction tree described in Section 9.2.2 should be adequate.

However, we also need to select more than one block. We must attempt to make all the blocks optimal, forcing them to share the optimal variables. If conditioning upon a variable, v, causes a large reduction in storage requirements, it is optimal wrt. storage requirements to condition upon this variable in all blocks. However, then the variable will never be sampled. We thus have the requirement, that each variable must be contained in at least one block. To enforce this and to ensure that the blocks are all created optimally, i.e., containing as many variables as possible and requiring minimal storage, we have to devise a more advanced scheme, see Figure 9.2. This scheme basically visits the blocks one at a time, and selects an optimal variable for being conditioned upon in each block. Of course, we have to ensure that the same variable is not conditioned upon in all blocks.

While running the algorithm in Figure 9.2, we have to maintain a separate junction tree for each of the blocks, representing the remaining variables, B_i, the variables that have been conditioned upon, A_i, and the correct reductions in storage requirements of the remaining variables. This means that when conditioning upon the optimal variable, v^*, we also have to update the junction tree corresponding to B_i as described in Section 9.2.2. Further, we have to update the reductions in storage requirements for a number of variables to reflect the changes in the junction tree. The method for doing this is outlined in Section 9.2.2 and here it suffices to say that it makes up for a significant part of the computations. Obviously, it is important to avoid performing this computation more often than necessary.

Variants of block selection method

Three variants of the selection method have been designed to offer flexibility with speed and quality of blocks. As mentioned in the previous section, the reason for this is that computing the reductions in storage requirements of a set of variables can be a very time-consuming operation. Thus, the three variants call the operation of Section 9.2.2 for updating the reductions in storage requirements of variables at different time intervals :

1. Compute the reductions in storage requirements at each step. Clearly this variant creates blocks with the lowest storage requirements of the three, however, for large networks it can be very time-consuming.

2. Compute the reductions in storage requirements when necessary. This variant initially computes the reductions in storage requirements for all variables, and then runs for a while without recomputing them. After a while, the values for reductions in storage requirements become less and less accurate, and finally when some criterion is fulfilled, they are recomputed. This criterion could for instance be that the current estimates of the reductions in storage requirements have very low values, and by recomputing them, it is possible that variables which actually have larger reductions are found. Another and simpler criterion might be to recompute reductions in storage requirements whenever a prespecified period of time has passed.

   This variant offers a compromise between speed and quality of selected blocks. Often, the blocks selected here are of sufficient quality, and the method is significantly faster than the first one.

3. Compute the reductions in storage requirements only once, at the beginning. Then it keeps selecting variables, until the remaining ones have too low reductions in storage requirements to be worth removing. This variant is very fast, but often creates blocks that contain too few variables and require too much storage.

Block visitation schemes

In practice, using the algorithm in Figure 9.2 is problematic. Due to the fixed order of visiting the blocks when selecting the optimal variables, it is possible for some of the blocks (the first few) to become smaller than the last ones. This is probably due to the fact that the first blocks are always given the best opportunity to select the most optimal variables, at least in the beginning. Also, by coincidence it is possible that multiple times the same blocks are allowed to select the best variables, and others are left with variables with lower reduction capabilities. One way to remedy this is, after each round, to assign *priorities* to the blocks and visit them according to these. The priority of a block could be computed in many ways, but should provide a measure of the quality of the block. Thus, in the next round we can select an optimal variable for removal from the block of lowest quality first. In the current implementation of blocking Gibbs (see Appendix A), this measure is computed as follows,

$$M_i = \log(\text{storage\_req}(B_i)) * (N_{LC} - N_{VR}), \tag{9.1}$$

where storage_req(B_i) is the current storage requirements of block i, N_{LC} is the number of variables in the largest clique, and N_{VR} is the number of variables in the largest clique that it is still possible to condition upon (i.e., if a variable has been conditioned upon in all other blocks, it cannot be conditioned upon in this block).

M_i provides a measure of block i's need to condition upon an optimal variable. Obviously, the larger the storage requirements of the block, i.e., storage_req(B_i), the greater this need, and also, the fewer variables that can be conditioned upon in the largest clique, i.e., N_{VR}, the greater this need. Only the largest clique is considered, as this clique makes up the largest proportion of the storage requirements of all cliques, and thus it is of the highest importance to reduce the size of this clique.

The selection algorithm of Figure 9.2 is modified to include steps computing these priorities for the blocks, and visiting them in the order indicated by them.

9.2.2 Reduction in Storage Requirements

In the following, the method for computing the reductions in storage requirements of variables is presented. The purpose of the method is to estimate the reduction in storage requirements obtained by conditioning upon a variable and replacing it with clones, by performing local operations directly on the junction tree, and avoiding performing a new triangulation. The reduction in storage requirements is thus found by performing a number of operations on the junction tree as outlined in Figure 9.3, and then subtracting the storage requirements of the reduced junction tree from the initial.

In practice, only the cliques of the junction tree are considered as these amount for most of the storage requirements. The separators amount for only a smaller part of the storage requirements, and leaving the separators out of the computation makes it unnecessary for the algorithm to spend time reconnecting new and reduced cliques with separators. Strictly, the algorithm is thus not operating on a junction tree, but a collection of cliques. Obviously, the resulting reduced collection of cliques can always be connected in a junction tree.

The first loop of the algorithm in Figure 9.3 assures that all instances of the variable in question, v, in the junction tree are replaced with all the clones, $v_0, \ldots, v_k$. The number of clones, $k+1$, equals the number of offspring plus one if there are any parents, see Section 4.6 and Figure 9.6. Replacing v with all the clones in the cliques implies that links are placed between the clones and the former neighbours of v, and between the clones, as all these variables now appear in the same cliques. When the replacement is performed, the location of each clone will be known, i.e., which clone is connected with the parents, which is connected with the first offspring, etc., see Figure 9.6. This means that we know which of the new links are fill-in links and which are not.

To illustrate that new fill-in links are created when replacing v with clones, as all variables in a clique must be pairwise connected, consider the subgraph in Figure 9.7(a) which is responsible for three cliques in the junction tree, see (b). When V is replaced with its two clones, V_0 and V_1, in these cliques, we obtain the cliques in (c). These cliques correspond with the subgraph in (d) where several new fill-in links have been inserted. For instance, a fill-in link has been placed between V_0 and W, due to the new clique $\{U, V_0, V_1, W\}$ that enforces that U, V_0, V_1, and W are pairwise linked.

When the variable is replaced with clones, it cannot be known whether any of the new fill-in links can be avoided. Therefore, the only safe conduct is to link each clone with all the neighbours of the original variable, and with each of the other clones. As the location of each clone is known, and it is only connected with either the parents, or one of the offspring in the Bayesian network, the other links must be fill-in links as any other type of link would introduce changes in the underlying network. This again is seen in Figure 9.7 where V_0 is connected to S with a causal link, and to all other neighbours with fill-in links. V_1 on the other hand is connected

```
for each clique C containing v do:
    replace v with all clones v_0, ... , v_k in C
end for.

find redundant fill-in link, α–β in G, see Figure 9.4

more_links ← true
while more_links = true, do:
    remove α–β from G
    split C = {α, β, v_1, ... , v_k} into two cliques, C_1 = {α, v_1, ... , v_k}
          and C_2 = {β, v_1, ... , v_k}
    remove C_1 if subset of any other clique
    remove C_2 if subset of any other clique

    find redundant fill-in link, δ–γ, leading from either α or β and
          set found correspondingly, see Figure 9.5
    α–β ← δ–γ

    if not found do:
        more_links ← false
    end if.
end while.
```

Figure 9.3: The algorithm for updating a junction tree corresponding to conditioning upon a variable and replacing it with clones. Subroutines can be found in Figures 9.4 and 9.5.

```
found ← false
α–β ← first fill-in link in G
while not found and more fill-in links, do:
    if exists only one clique, C, containing α, β, do:
        found ← true
    else
        α–β ← next fill-in link in G
    end if.
end while.
```

Figure 9.4: The algorithm for finding a redundant fill-in link in a triangulated graph, G. It is called from the main algorithm in Figure 9.3.

to U with a moral link and to W with a causal link, and to all other neighbours with fill-in links.

However, when performing the operation of connecting all clones to each other and to the neighbours of v we have to guarantee that the graph is still triangulated. This can be shown formally as follows.

Theorem 9.2.1. *Let $G = (V, E)$ be a triangulated graph, $v \in V$, and $G' = (V', E')$, where $V' = V \setminus \{v\} \cup \{v_0, \ldots, v_n\}$, $E' = E \cup \{v_i - v_j | i \neq j, i = 0, \ldots, k\} \cup \{v_i - u | u \in$*

```
found ← false
α–γ ← first fill-in link starting from α, γ ≠ β
while not found and more fill-in links α–γ, do:
    if exists only one clique, C, containing α, γ, do:
        found ← true
    else
        α–γ ← next fill-in link starting from α, γ ≠ β
    end if.
end while.

if found=true do:
    δ–γ ← α–γ
end if.

if not found do:
    β–γ ← first fill-in link starting from β, γ ≠ α
    while not found and more fill-in links, β–γ, do:
        if exists only one clique, C, containing β, γ, do:
            found ← true
        else
            β–γ ← next fill-in link starting from β, γ ≠ α
        end if.
    end while.

    if found=true do:
        δ–γ ← β–γ
    end if.
end if.
```

Figure 9.5: The algorithm for finding a new redundant fill-in link δ–γ starting from the previous redundant link α–β. If no link is found, *found* will be set to false.

$nb(v)\}$[1], *and* $v_0, \ldots, v_n \notin V$. *Then* G' *is triangulated.*

Proof: Consider Figure 9.8. Any chordless n-cycle, $n > 3$, in G' must (by construction of G') include at least two variables x and y from $\mathrm{nb}(v)$ and two variables u and w from $\{v_0, \ldots, v_k\}$. Thus, the cycle must be of form $< \ldots, x, u, \ldots, w, y, \ldots >$. However, since x is connected with w (and u is connected with y) and G is triangulated, G' must also be triangulated. □

After replacing v with the clones in the first loop of Figure 9.3, it is possible that some fill-in links have been rendered *redundant*. A redundant fill-in link is one that can be removed without creating cycles of length greater than three without chords, i.e., the graph will still be triangulated after removing the link. Following Thm. (2) of Kjærulff (1993), redundant fill-in links are links α–β where $\{\alpha, \beta\}$ is the subset of exactly one clique in the junction tree. This also corresponds with the requirement that $\{\alpha, \beta\}$ is not a subset of any separator in the junction tree. If any of these equivalent requirements hold, the clique can be split into two such that if the clique contains $\{\alpha, \beta, v_1, \ldots, v_n\}$, the first will contain $\{\alpha, v_1, \ldots, v_n\}$ and the

[1] $\mathrm{nb}(v)$ is the neighbours of v.

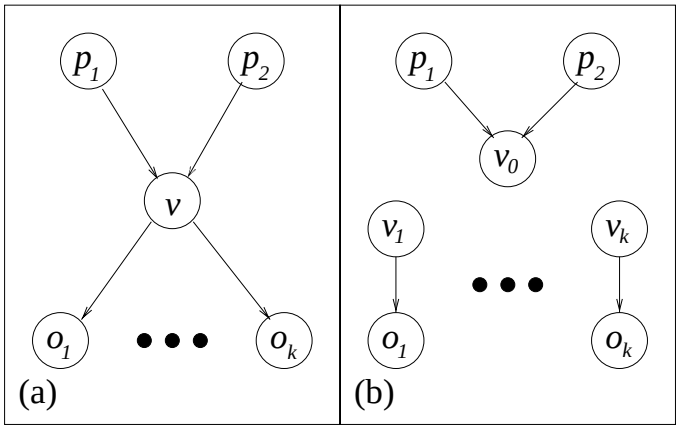


Figure 9.6: In a small Bayesian network (a) with two parents and k offspring, a variable v is conditioned upon and replaced with clones, v_0, $v_1, \ldots, v_k$.

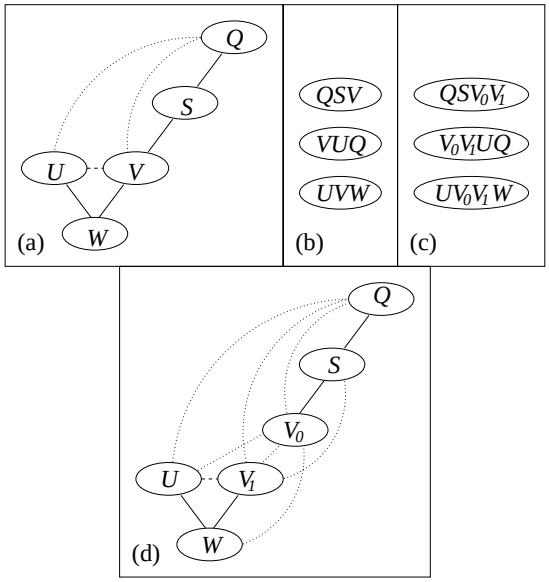


Figure 9.7: A subgraph of the moralized and triangulated graph in Figure 9.11(a) is shown in (a). In (b), are shown the cliques corresponding with this subgraph, and in (c) are shown the cliques resulting from the first loop of Figure 9.3 where V has been replaced with clones, V_1 and V_2. Thus, in (d) is shown the subgraph with new fill-in links corresponding to the new cliques in (c).

second $\{\beta, v_1, \ldots, v_n\}$. Thus, the redundant link is no longer represented in any clique.

As mentioned by Kjærulff (1993), the redundant fill-in links can be discovered in a local manner, once the first has been discovered. In the algorithm of Figure 9.3, the first redundant fill-in link is discovered by the subroutine of Figure 9.4 that simply loops through the set of fill-in links until one is discovered. Once the first redundant

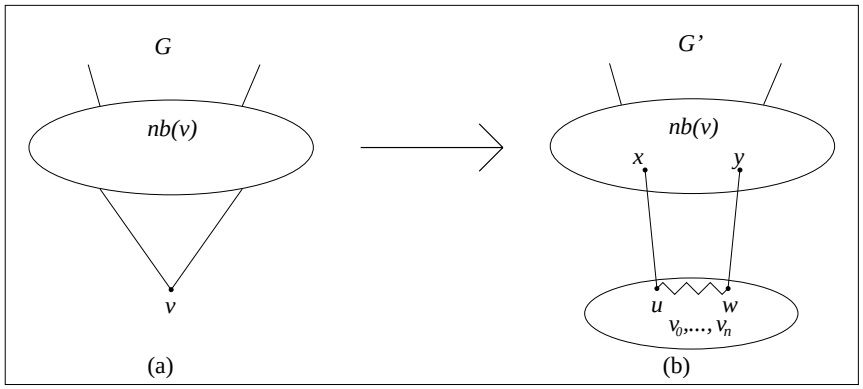


Figure 9.8: In (a), v and its neighbours in the graph, G, are shown. In (b), $v_0, \ldots, v_k$ have replaced v, and are connected with the neighbours of v and each other. Thus, $u, w \in \{v_0, \ldots, v_k\}$ and $x, y \in \mathrm{nb}(v)$.

link α–β has been removed and the cliques have been reduced correspondingly, the next is discovered by the subroutine in Figure 9.5 where all fill-in links leading from either α or β are examined.

When the junction tree has been reduced by replacing a clique with two smaller cliques, it must be checked whether any of these cliques are subsets of any other cliques in the junction tree. If this is the case, the new clique can be removed, as it will still be contained elsewhere in the junction tree.

We can exemplify the removal of redundant fill-in links with the last loop of Figure 9.3 by considering the example in Figure 9.7. The removal of redundant fill-in links for this example will proceed as follows, also illustrated in Figure 9.9 :

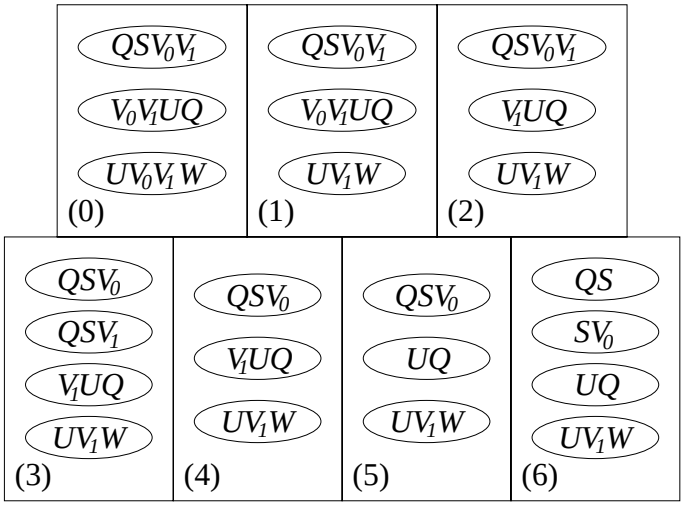


Figure 9.9: Removal of redundant fill-in links from the cliques in Figure 9.7(c). The original cliques are shown in (0). One by one, the redundant links are removed, in (1), V_0–W, in (2), V_0–U, in (3), V_0–V_1, in (4), V_1–S, in (5), V_1–Q, and in (6), Q–V_0.

1. V_0–W is redundant, as $\{V_0, W\}$ is a subset of only $\{U, V_0, V_1, W\}$. Thus, this clique is split into two smaller cliques $\{U, V_0, V_1\}$ and $\{U, V_1, W\}$. Further, $\{U, V_0, V_1\}$ can be removed as it is the subset of another clique. The resulting cliques are shown in Figure 9.9(1).

2. V_0–U is redundant, as $\{V_0, U\}$ is a subset of only $\{V_0, V_1, U, Q\}$. This clique is replaced by $\{V_0, V_1, Q\}$ and $\{V_1, U, Q\}$, where the first can be removed as it is the subset of $\{Q, S, V_0, V_1\}$. Again, the resulting cliques are shown in Figure 9.9(2).

3. V_0–V_1 is redundant, as $\{V_0, V_1\}$ is a subset of only $\{Q, S, V_0, V_1\}$. This clique is then replaced by $\{Q, S, V_0\}$ and $\{Q, S, V_1\}$, none of which can be removed, see (3).

4. V_1–S is redundant, as $\{V_1, S\}$ is a subset of only $\{Q, S, V_1\}$. $\{Q, S, V_1\}$ is replaced by $\{Q, S\}$ and $\{Q, V_1\}$. Both of these smaller cliques are subsets of other cliques and can be removed, see (4).

5. V_1–Q is redundant, as $\{V_1, Q\}$ is a subset of only $\{V_1, U, Q\}$. $\{V_1, U, Q\}$ is replaced by $\{V_1, U\}$ and $\{U, Q\}$ where the first can be removed as it is a subset of $\{U, V_1, W\}$, see (5).

6. Finally, Q–V_0 is redundant, as $\{Q, V_0\}$ is a subset of only $\{Q, S, V_0\}$. This clique is replaced with $\{Q, S\}$ and $\{S, V_0\}$, none of which can be removed.

The end result can thus be seen in Figure 9.9(6). It is seen that of the original six fill-in links, only U–Q remains. However, when considering the larger junction tree that the cliques in (6) are part of, it will also be possible to remove this link, etc.

Examples

We will illustrate the algorithm with a few examples. First, in Figure 9.10, some examples showing the removal of different variables from a junction tree are given. The original Bayesian network and the corresponding junction tree is shown in (a) and (b).

a: In Figure 9.10(c), a is conditioned upon and replaced with a single clone, a_0. Thus, $\{a, b, e\}$ is changed to $\{a_0, b, e\}$ as shown in (d). No further reductions can be performed as no fill-in links are rendered redundant. The resulting junction tree is shown in (e).

f: In Figure 9.10(f), f is conditioned upon and replaced with clones, f_0 and f_1. Thus, f is replaced with the clones in the four cliques containing f, as shown in (g). After removal of redundant fill-in links, these cliques are reduced, leaving only $\{b, c, f_0\}$ and $\{f_1, g, h\}$. The resulting junction tree is thus shown in (h).

b: In Figure 9.10(i), b is conditioned upon and replaced with clones, b_0 and b_1. b is replaced with b_0 and b_1 in the three cliques containing b, as shown in (j). These can be reduced when removing redundant fill-in links, resulting in the junction tree in (k).

We will also illustrate the algorithm of Figure 9.3 with a larger example shown in Figure 9.11. In Figure 9.12, a junction tree corresponding to this network is

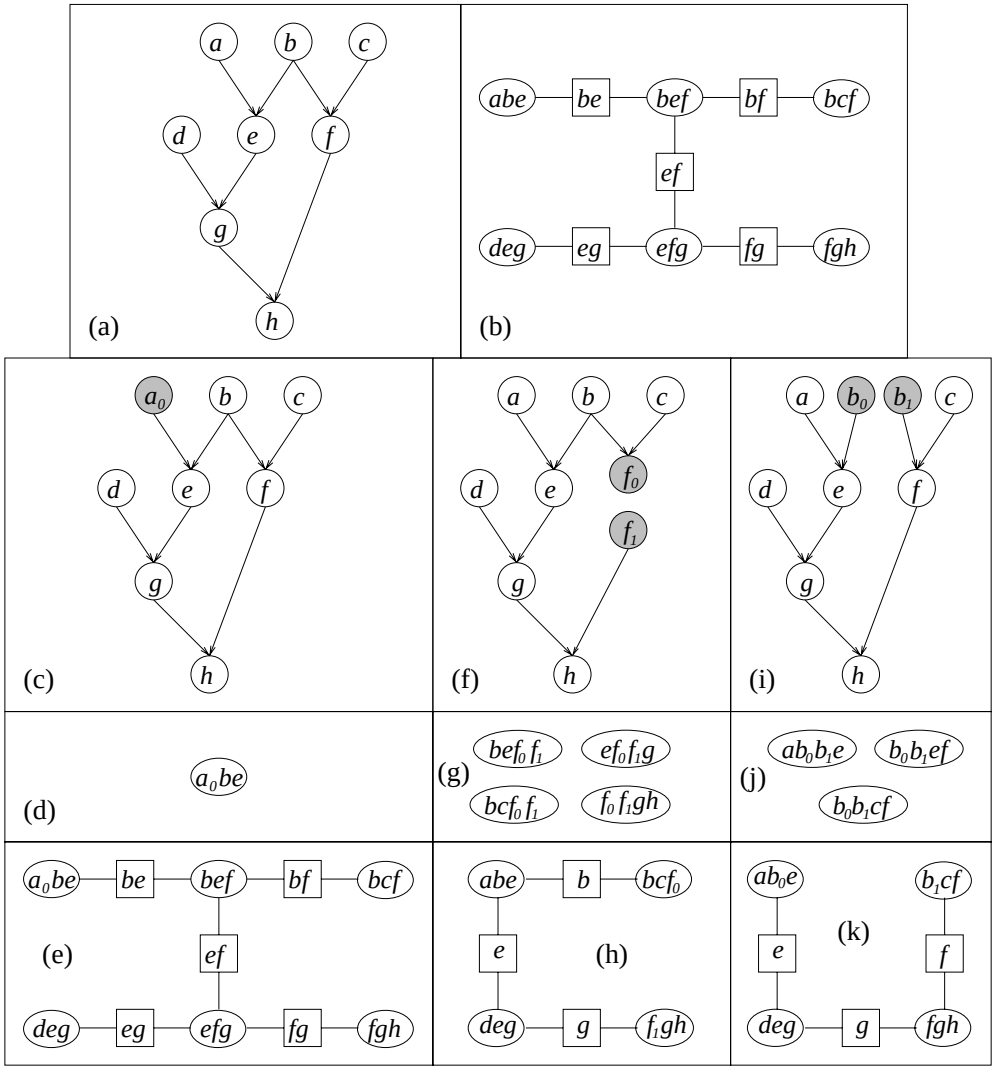


Figure 9.10: Some examples showing the reductions in a junction tree corresponding to conditioning upon various variables, as performed by the algorithm in Figure 9.3. In (a), the original network is shown, and in (b), the original junction tree. In (c), a is conditioned upon, in (f), f is conditioned upon, and in (i), b is conditioned upon. The cliques that are created when replacing the variable in question with clones are shown in respectively (d), (g), and (j). The reduced junction trees are shown in respectively (d), (h), and (k).

shown. Conditioning upon S causes the clique $\{S, V, Q\}$ to be replaced by the clique, $\{S_0, S_1, V, Q\}$.

Further reductions are possible as removing S renders many fill-in links redundant when the cycle S–V–U–R–P–J–F–D–G–L–Q–S is broken. Immediately, the fill-in link S_1–Q only appears in one clique, enabling it to be removed. Initially, this is the only redundant link, but once it has been removed and the junction tree has been reduced accordingly, the link S_1–S_2 becomes redundant, etc. Altogether, 12

fill-in links can be removed, resulting in the substantially reduced junction tree in Figure 9.13. The reduction in clique and separator sizes can be seen in Table 9.1. Obviously, in Figure 9.13 there are much fewer large cliques and separators than in the original junction tree. In Table 9.2, the storage requirements of the junction trees in the three figures are shown, given that all the variables have a specific number of states, s.

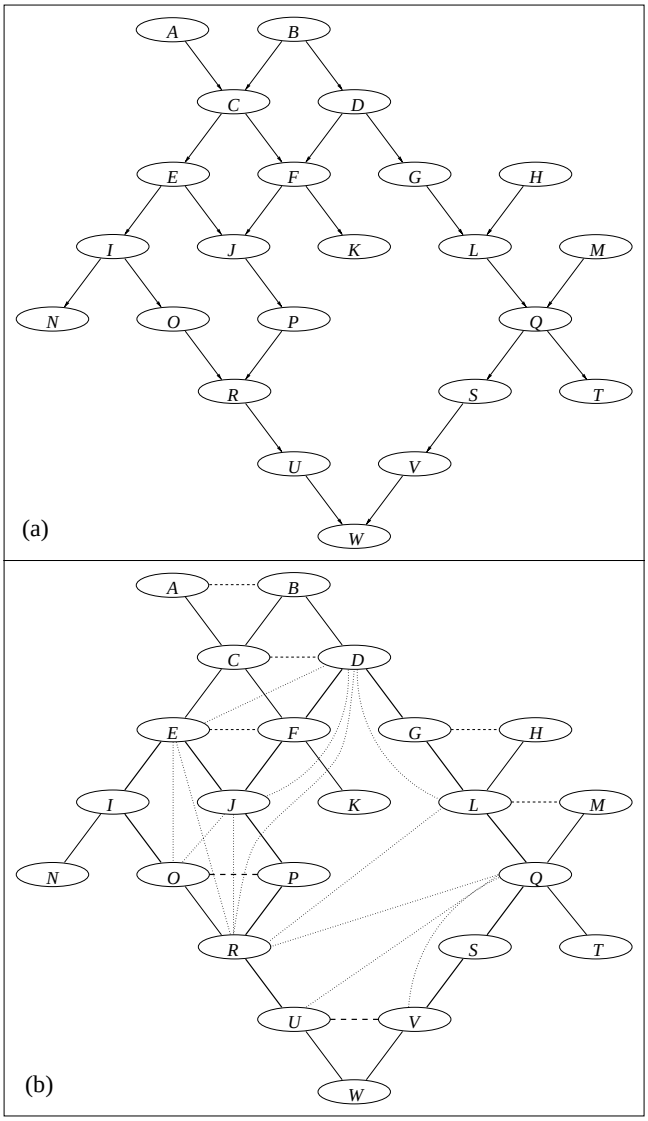


Figure 9.11: A large example network in (a). In (b), the moralized and triangulated graph is shown. Moral links are dashed, and fill-in links added during the triangulation are dotted.

The method for finding the reduction in storage requirements outlined with the algorithm in Figure 9.1 requires in each step an update of the junction tree of the current block when conditioning upon a variable with the algorithm in Figure 9.3. Further, when the junction tree has been reduced corresponding to conditioning upon the variable, the previously computed reductions in storage requirements for

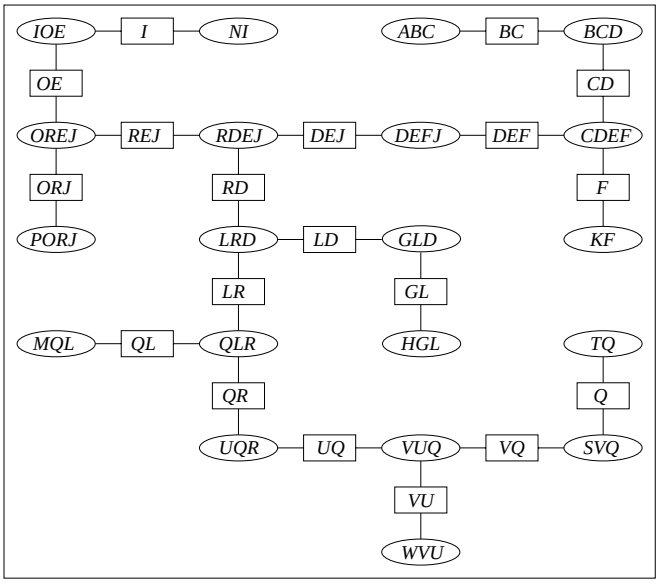


Figure 9.12: The junction tree of the Bayesian network in Figure 9.11.

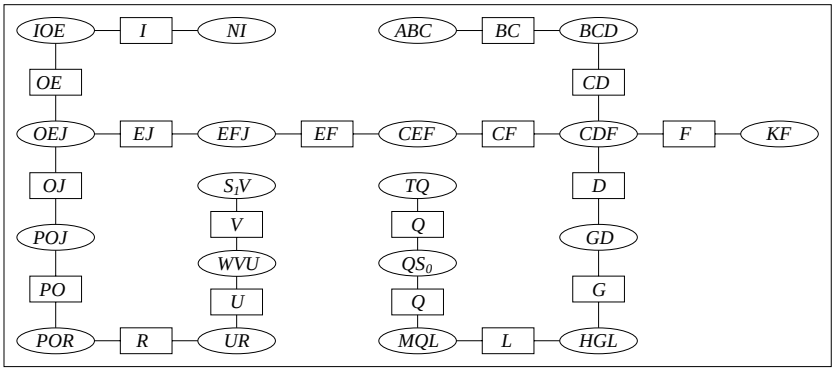


Figure 9.13: The junction tree has been further reduced by iteratively removing redundant fill-in links and reducing the tree accordingly.

many other variables will no longer be correct. Therefore, it is necessary to update these reductions for all variables that are affected by the previous changes in the junction tree, i.e., all variables present in the cliques containing the optimal variable v^* and the cliques affected by the removal of redundant fill-in links. As many variables may be affected, this is a time-consuming operation with a complexity that is difficult to express exactly, but it makes up a significant part of the computations. This paves the ground for introducing the variants of the block selection method in Section 9.2.1 where this operation can be executed more or less often.

The method presented here has not yet been implemented and was thus not used in the experiments of the thesis. The method that was used, simply removed the variable in question from the junction tree, resulting in reductions in storage requirements approximately like the correct. The used method did not remove redundant fill-in links suggesting that the one presented here should yield better performance.

	Figure	
	9.12	9.13
4-cliques	5	0
3-cliques	12	12
3-separators	4	0
2-cliques	3	7
2-separators	12	8
1-separators	3	10

Table 9.1: The distribution of different sized cliques and separators in the two junction trees of Figures 9.12 and 9.13.

	Figure	
s	9.12	9.13
2	274	176 (36%)
3	981	489 (50%)
4	2556	1048 (59%)

Table 9.2: The storage requirements of the junction trees of the two figures, given that all variables have s states. The reductions in storage requirements in percentage are given in parentheses.

Though the one presented here supposedly is better, it is shown through empirical investigations in Chapters 16 and 17 that the implemented method also performs quite well.

9.3 Criterion 2 : Sampling Variables Equally Often

We want variables to be sampled approximately equally often to ensure that we do not get situations where some variables are sampled many times in each iteration, and others are sampled only once. However, Criterion 1 urges us to condition upon the optimal (wrt. reduction of storage requirements) variables in most of the blocks, and as a result these variables are then sampled only one or a few times in each iteration, whereas the majority of variables are included in all blocks and thus sampled much more often. It seems that in most cases we *have* to use the optimal variables for reducing storage requirements to obtain blocks with sufficiently low storage requirements, letting Criterion 1 take priority over Criterion 2.

However, the optimal variables may also be important in the network as they are located in many loops, i.e., it may lower the mixing rate of the blocking Gibbs sampler to only sample these important variables relatively rarely. Empirical results indicate that this is the case. Considering Table 9.3 showing the M.S.E. (mean squared error) of results obtained with blocking Gibbs runs having either the requirement that variables must be sampled in at least half the blocks (method 1) or that variables must be sampled in at least one block (method 2). The last requirement is the minimal possible requirement forcing each variable to be sampled at least once in each iteration. The runs have been performed on Pedigree B introduced in Section 16.2, and as in Chapter 16 the mean squared errors are obtained

Iterations	M.S.E. (1)	M.S.E. (2)	$\sqrt{\text{M.S.E. (1)}}$	$\sqrt{\text{M.S.E. (2)}}$
100	$1.31 \cdot 10^{-10}$	$3.50 \cdot 10^{-9}$	$1.14 \cdot 10^{-5}$	$5.91 \cdot 10^{-5}$
1,000	$1.36 \cdot 10^{-11}$	$3.66 \cdot 10^{-11}$	$3.69 \cdot 10^{-6}$	$6.05 \cdot 10^{-6}$
10,000	$9.40 \cdot 10^{-13}$	$3.90 \cdot 10^{-12}$	$9.69 \cdot 10^{-7}$	$1.97 \cdot 10^{-6}$
100,000	$8.93 \cdot 10^{-14}$	$5.04 \cdot 10^{-13}$	$2.99 \cdot 10^{-7}$	$7.10 \cdot 10^{-7}$
1,000,000	$1.61 \cdot 10^{-14}$	$3.25 \cdot 10^{-14}$	$1.27 \cdot 10^{-7}$	$1.80 \cdot 10^{-7}$

Table 9.3: Here, the M.S.E. is shown for runs having either the requirement that variables must be sampled in half the blocks (method 1) or that variables must be sampled in at least one block (method 2). For each method, runs of various lengths are performed. The blocking Gibbs runs were performed on Pedigree B introduced in Section 16.2.

```
for i = 1 to k do:
    B_i ← initial junction tree
end for.

while storage_req(B_1) > S ∨ ... ∨ storage_req(B_k) > S do:
    for i = 1 to k do:
        v_j ← variable in B_i with largest reduction
            of storage requirements for which c_j < γ
        c_j ← c_j + 1
        condition upon v_j in B_i and update correspondingly
    end for.
end while.
```

Figure 9.14: Selection loop ensuring that variables are not conditioned upon in more than γ blocks.

by comparing the blocking Gibbs estimates with near exact results obtained by forward sampling for a very long time. Table 9.3 clearly indicates that better precision is gained when optimal variables are sampled in more blocks. However, in many cases it is not possible to construct blocks with reasonable storage requirements when requiring variables to be sampled in at least half the blocks, and in these cases we must resort to the minimal requirement, though causing slightly slower convergence.

To ensure that the blocks have approximately the same size, we can use a simple algorithm, here outlined with a piece of pseudo code in Figure 9.14. We want to construct k blocks, called $B_1, \ldots, B_k$. A counter, c_i, is associated with each variable, v_i. c_i counts how many times v_i has been conditioned upon in any block (i.e., how many times in each iteration v_i is *not* sampled). We have tentatively chosen that c_i must not exceed $\gamma = \lfloor \frac{k}{2} \rfloor$ for any variable, thus forcing the optimal variables to be included in at least half the blocks, thus they are sampled at least $\lfloor \frac{k}{2} \rfloor$ times in each iteration. With the implemented version of blocking Gibbs, different values for γ can be used, however. For instance, if the network is very large and complex, γ can be set as high as $k-1$, i.e., the block selection algorithm allows variables to be placed in only one block, thus sampling them only once in each iteration.

9.4 Criterion 3 : Blocks for Irreducibility

As has been mentioned in previous sections, irreducibility can often not be guaranteed in general applications. In this section, we will explain what causes this problem and provide some simple examples.

As defined in Section 5.1, a Markov chain is irreducible if all its states communicate, meaning that it should be possible to get from any state to any other state in a finite number of steps. With the single-site Gibbs sampler this corresponds to the requirements that by updating one variable at a time conditional on the remaining, it should be possible to get from one legal configuration of the entire network to any other. This is clearly not always the case.

Consider the Bayesian network in Figure 9.15 that models the logical `xor` function. The conditional probabilities of C can be seen in the figure. If C is observed to the value 1, then A and B can be in two possible configurations, $(A = 0, B = 1)$ or $(A = 1, B = 0)$. As the single-site Gibbs sampler can only update one variable at a time, it will get stuck with this network. If the starting configuration is selected to be, e.g., without loss of generality, $(A = 0, B = 1, C = 1)$, there are three possibilities depending on which variable we sample first,

A: the single-site Gibbs sampler can only draw the same value for A, as this is the only legal value given the values of B and C,

B: similarly, the sampler cannot change the value of B, as this is the only legal value given the values of A and C,

C: finally, the sampler must sample C to 1 again, as this is the only legal value given the values of A and B.

Thus, the single-site Gibbs sampler will be stuck forever in the initial configuration. If it was started in another configuration, e.g., $(A = 0, B = 0, C = 0)$, the sampler would be equally stuck. When the sampler is not able to traverse the configuration space freely, we say that the sampler is *reducible*. It is then clear that the Markov chain induced by the Gibbs sampler will not have an equilibrium distribution. Instead it will have several invariant distributions, one for each of the noncommunicating subsets of the sample space. These noncommunicating subsets will be denoted *the noncommunicating classes* or, indiscriminately, *the noncommunicating sets*.

As is seen from the example in Figure 9.15, the noncommunicating classes arise because of the single-site updating, i.e., if more than one variable are updated jointly, the problem can be avoided. For instance, in Figure 9.15, if variables A and B are sampled jointly, the Gibbs sampler would be able to jump between the configurations, $(A = 0, B = 1, C = 1)$ and $(A = 1, B = 0, C = 1)$. However, if C is still updated by itself conditional on A and B, the sampler will not be able to jump between, e.g., $(A = 0, B = 1, C = 1)$ and $(A = 0, B = 0, C = 0)$. To obtain a completely irreducible Gibbs sampler in this case, we have to update all three variables jointly.

The above principles have been used in the blocking Gibbs sampling algorithm. The blocking Gibbs sampler would easily be able to sample the above three variables jointly, and induce an irreducible and thus ergodic Markov chain. However, in general, it is difficult to establish the required blocks to obtain an irreducible chain. In smaller problems it is easy to locate the variables that must be updated jointly by hand, but in large problems with many variables we need a general method.

Such a general method has not yet been found. In fact, it is believed that the general problem of constructing blocks for irreducibility, i.e., locating the noncom-

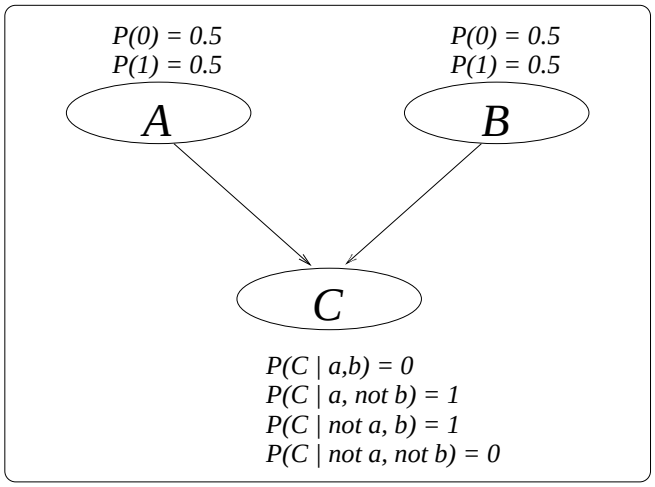


Figure 9.15: A small Bayesian network modeling logical `xor`.

municating classes, is NP-hard. However, in specific classes of problems, it may be feasible, e.g., in genetics.

However, even though it is very difficult to locate the variables that must be blocked for the blocking Gibbs sampler to become irreducible, in practice, as the blocking Gibbs sampler is usually able to select blocks containing more than 90% of the variables, irreducibility is obtained. Empirical results confirm that the blocking Gibbs sampler often becomes irreducible in cases where other MCMC methods would have been reducible, simply by selecting these large blocks. However, this is of course not a guarantee that the blocking Gibbs sampler will become irreducible in the general case. In each case it must be checked whether the selected blocks induce an irreducible chain, and if possible, a general method for selecting blocks for irreducibility must be found. For genetics applications it is easy to check whether the blocking Gibbs sampler is irreducible, see Section 16.7.

In Chapters 16 and 17, some examples of reducible problems in genetics and methods for constructing blocks for them are given. In Chapter 15, near-reducibility in genetics applications is discussed. Near-reducibility implies that the sampler is almost but not quite reducible and occurs for completely different reasons. Finally, in Chapter 18, the limitations of a published algorithm (Lin et al. 1994) for locating the noncommunicating classes in genetics problems are outlined, and pointers towards a general algorithm are provided.

Chapter 10

Forward Sampling Barren Variables

As mentioned in Chapter 6, the *barren* variables in the Bayesian network are forward sampled by the blocking Gibbs sampler. A barren variable is an unobserved variable with no observed descendants. In this section, the forward sampling algorithm will be introduced and it will be explained why it is beneficial to apply it to barren variables.

10.1 The Forward Sampling Algorithm

The Monte Carlo sampling scheme denoted *forward sampling* is an efficient way of obtaining independent samples in Bayesian networks where observations are not available for variables with unobserved predecessors. It was proposed by Henrion (1988) in the expert systems literature but a similar algorithm known as *gene dropping* was introduced years before in genetics by Edwards (1968) and later generalized by MacCluer, Vandeburg, Read & Ryder (1986).

Let p be a probability function on $\mathrm{Sp}(U)$ where U is a set of discrete variables such that it factorizes according to the DAG of a Bayesian network, BN. A sample can be obtained from p with the forward sampling algorithm as follows. Let $U_0 \subseteq U$ such that $\mathrm{pa}(v) = \emptyset$ for all $v \in U_0$ (i.e., variables in U_0 have no parents). Since p obeys the directed global Markov property wrt. BN (see Section 4.7.2), the variables in U_0 can be sampled independently. Let $U_1 \subseteq U \setminus U_0$ such that $\mathrm{pa}(v) \setminus U_0 = \emptyset$ for all $v \in U_1$ (i.e., the parents of variables in U_1 are in U_0). Now, since U_1 is a set of conditionally independent variables given U_0, they can be sampled independently. Continuing in this fashion until all variables have been sampled, we obtain a sample of the joint distribution. Thus, an approximation, $\widehat{p}$, of p can be obtained by creating n samples, $x^{(1)}, \ldots, x^{(n)}$, from p and estimating $\widehat{p}(v_i = s)$ by either the empirical or the mixture estimate as described in Section 5.2.3. This sampling procedure is known as *forward sampling*.

The basic forward sampling procedure works well even in the case where evidence is available for a subset $A \subseteq U$ such that $\mathrm{pa}(v) \subset A$ for all $v \in A$. But if there is at least one $v \in A$ for which $\mathrm{pa}(v) \not\subset A$ (i.e., it has unobserved parents), then only the samples for which the sampled value for A is identical to the evidence on A should be taken into account. This variant of forward sampling which has been investigated by Henrion (1988) under the name of *logic sampling* gets increasingly inefficient as the probability of the evidence on variables with non-observed parents

decreases. A more general forward sampling procedure which is capable of handling non-categorical evidence is known under the name of *importance sampling* (Yiannoutsos & Gelfand 1994). However, importance sampling is also inefficient when evidence is present on variables with non-observed parents.

As mentioned in Chapter 8, if 100 bottom level variables in Figure 8.1 on page 49 have been observed and each of them has three states, then the forward sampler has a probability of approximately $1.9 \cdot 10^{-48}$ of drawing a configuration corresponding with the observed variables. Thus, using a fast computer, on the average it will require more than 10^{30} years to obtain a usable joint sample of the variables.

10.2 Barren Variables

Clearly, the forward sampling algorithm can be applied to the barren variables of a Bayesian network. However, as mentioned in the previous section, the forward sampling scheme works even in the case where some of the top level variables are observed. This means that we can apply the forward sampling scheme to the barren variables including their parents. We denote this set of variables, A_{barren}. Knowing that the forward sampler is always applied to A_{barren} when the blocking Gibbs sampler has obtained a complete configuration for the remaining variables, see Chapter 6, the top variables of A_{barren} are always observed. Thus, the forward sampler obtains a joint sample of the barren variables that is legal wrt. the remaining variables.

It is beneficial to apply the forward sampler to the barren variables instead of letting the blocking Gibbs sampler handle them, for three reasons :

1. The forward sampler is very fast compared with the blocking Gibbs sampler, as it is a much simpler operation to draw a variable conditional on its parents, than drawing large sets of variables jointly.

2. The forward sampler obtains independent samples, rather than the dependent samples obtained with the blocking Gibbs sampler. Thus, the forward sampler has the maximal possible rate of convergence.

3. In many applications the barren variables constitute a substantial part of the set of variables (sometimes more than 50%), thus forward sampling the barren variables significantly reduces the size of the network handled by blocking Gibbs. This reduces the storage requirements of exact inference on the network, and thus makes it easier to select blocks with smaller storage requirements.

The forward sampler is applied each time a block has been visited by the blocking Gibbs algorithm to obtain a joint sample of the barren variables. As this step is so fast compared with the blocking Gibbs updating, it is even practical to make several draws with the forward sampler for each blocking Gibbs step.

Part II

Genetics Applications

Chapter 11

Introduction to Part II

The computation of probabilities on pedigrees is an essential component in any analysis of genetic data on groups of related individuals. Such computations are relevant to applications in several areas such as, genetic counseling, selective animal breeding, inference on the genetic nature of a disease, analysis of surviving genes in an endangered species and linkage analysis, to name but a few. The exact method for computing such probabilities on pedigrees in which at least one of every parent pair is a founder proposed by Elston & Stewart (1971), was extended by Lange & Elston (1975) and finally generalized to include arbitrarily complex pedigrees and genetic models by Cannings et al. (1978). This method has become known in the statistical genetics literature as *peeling*. Interestingly, it was re-invented several years later in the expert systems literature (Lauritzen & Spiegelhalter 1988) as a means of calculating posterior probabilities on general Bayesian networks, see Chapter 4. While, in theory, every pedigree can be peeled, in practice, due to the enormous storage requirements of the method, exact calculations are very often infeasible. The computational problems arise when the pedigree, viewed as a graph, has too many cycles, or *loops*. Two common types of loops that can make a pedigree complex are *inbreeding* loops which are formed when individuals marry near relatives and *marriage chains* which arise when several individuals are interrelated by marriage. We note that we are using the term "marriage" in the pedigree analysis sense: individuals are said to be married only if they have a common offspring in the pedigree. Calculations are trivial if the graph has no loops—whatever its size.

Simulation, as a technique, is not new to the area of pedigree analysis. Random sampling of ancestral lines was used by Wright & McPhee (1925) to estimate coefficients of kinship and inbreeding. Simulation of genes flowing through a pedigree was used to address the same problem by Edwards (1968). This was developed into the method of *gene dropping* (MacCluer et al. 1986) which, in the absence of phenotypic data, can be very effective in calculating probabilities of gene survival and genetic variability on pedigrees too complex for peeling. Again, the method was re-invented several years later in the expert systems literature by Henrion (1988) as *forward sampling*, see Chapter 10. When phenotypic data are available on some individuals in the pedigree, the backward simulation problem of generating a configuration of genotypes that is consistent with the observed phenotypes is more difficult. The number of possible genotypic configurations on a large pedigree can be huge whereas the proportion compatible with the observed data can be minute. Gene dropping subject to such constraints must involve rejection of illegal configurations which is not practical since, with any substantial amount of data, rejection rates can be close to 100%. The method described in Chapter 8 is another approach that can be used for generating a configuration of genotypes consistent with the observed

phenotypes. Several authors (Kong 1989, Ott 1989, Ploughman & Boehnke 1989) have used simulation to address questions in the area of linkage analysis. In each of these methods, it is necessary to be able to perform a peeling-type calculation at one of the loci considered, and no phenotypic data are assumed at the other. However, peeling even for a diallelic trait, is not always possible in a highly complex pedigree.

From the simple observation that the genetic model is Markovian, a neighbourhood system can be defined on a pedigree whereby, conditional on the neighbours, the genotypes of individuals are independent, see Section 4.7. This local dependency makes Markov chain Monte Carlo methods, such as the Gibbs sampler, very easy to implement and thus provides a means of estimating required posterior probabilities which cannot be calculated exactly (Sheehan 1990). The Gibbs sampler is a sampling scheme which visits each pedigree member in turn and updates the current estimate of genotype by sampling from the conditional distribution of genotype given the current configuration elsewhere on the graph and the phenotypic data, see Section 5.2 for a general description. These conditional distributions are very easy to calculate because of the local dependences induced by the neighbourhood system. The underlying Markov chain is irreducible for most traits determined by a diallelic locus and good estimates of ancestral probabilities were obtained using the Gibbs sampler on a highly complex pedigree of Greenland Eskimos by Sheehan (1992), see Figure 8.1.

However, irreducibility usually does not hold when a multiallelic locus is involved. This means that we cannot then sample properly from the true posterior distribution of genotype, given phenotype, and the resulting estimates for probabilities of interest are unreliable. Even when the chain is irreducible it may be *nearly* reducible so that although it is possible to sample the entire space, the time required to do so can be prohibitive. This latter situation is caused by the fact that some configurations linking one part of the search space to another occur with extremely small probability, see Chapter 15. Reducibility is a particularly serious problem in linkage analysis, for example, where highly polymorphic markers are preferred because they are more informative.

Several ways of getting around this problem have been proposed by various researchers over the last few years. Most of these involve relaxing the genetic model by assigning some positive probability to impossible configurations in order to facilitate movement between different areas of the search space. Importance sampling with weights of zero and one, proposed by Sheehan & Thomas (1993) is the most primitive of these. Basically, the transition laws are completely relaxed and samples from some incorrect distribution are obtained, with rejection of all configurations that do not agree with the true genetic model, until a consistent configuration is arrived at. This method is frequently used to arrive at a starting configuration for a Markov chain Monte Carlo application on a complex pedigree. Various other "heated chain" methods have been proposed which enable the sampler to jump from one noncommunicating class to another. These include the creating of individual specific bridging states between identifiable noncommunicating classes (Lin 1993), the companion chain method of Lin, Thompson & Wijsman (1993) when the classes cannot all be identified, the use of Metropolis-coupled samplers (Geyer 1991), and the annealing-type samplers of Geyer & Thompson (1995).

One problem with these algorithms is that they produce what can sometimes be a very large amount of useless samples just to get from one legal configuration to the next. This inefficiency has been avoided in the *blocking Gibbs* sampler described in Part I of this thesis which can be applied to general Bayesian networks and uses stochastic variables as basic unit of information instead of individuals, see Chapter 6. In a pedigree analysis application, a variable would typically represent one of

the genes of an individual, but it could also represent the phenotype or genotype, see Chapter 13. This method allows the blocking of large sets of correlated variables which are then sampled jointly using the exact method of Lauritzen & Spiegelhalter (1988), see Chapter 4. This exact method was used as it is computationally more advanced than the peeling method, enabling the marginal probabilities of all variables to be obtained in a single propagation. The blocks are constructed in such a way that variables taking part in many loops and thus causing the enormous storage requirements of exact methods are left out of as many blocks as possible, see Chapter 9. This way, the blocks comprise as large a part of the pedigree as can possibly be sampled jointly, conditional on the remaining variables. Empirical results show that usually more than 90% of variables in the pedigree will be sampled jointly, thus yielding very fast mixing and alleviating problems of reducibility and near reducibility. Recently, Jensen & Kong (1996) successfully applied the blocking Gibbs sampler to a complex linkage problem, see Chapter 17.

However, in all of these methods, even when the required irreducibility condition has been provided, there is no real guarantee that the sampler is mixing quickly enough and there is no useful diagnostic indicating when to stop. For how long must the sampler be run in order to yield good estimates from the true posterior distribution? Unfortunately, this will be heavily dependent on the size and topology of the pedigree, as well as on the nature and position of genetic data. This is a general Markov chain Monte Carlo problem and is not peculiar to pedigrees. In the Bayesian networks setting, Hrycej (1990) recommends a stopping rule for the Gibbs sampler based on the stability of the estimates over successive cycles. Given a minimum number of cycles that must first be completed, the process can be stopped when successive estimates are sufficiently close. This is misleading as the random process generating these estimates could exhibit local stability without being close to its mean.

An algorithm such as that of Lin et al. (1994) which attempts to determine the noncommunicating classes of the underlying Markov chain , or "islands", examples of which can be seen below, would be useful. Once these classes can be identified, a sampler, such as that in (Lin 1995) can be designed that is guaranteed to visit each one at some stage during the run. Also, the results of such an algorithm could be used to identify the variables that should be blocked to obtain an irreducible Markov chain with the blocking Gibbs method. The above algorithm, however, is not a general method for finding these noncommunicating classes. It fails to correctly detect these classes in a number of simple cases, see Chapter 18.

This part of the thesis applies the blocking Gibbs sampler to a number of complex problems in genetics. First, in the next chapter, a brief introduction to the basic genetics concepts and terminology used in this thesis will be given. In Chapter 13, the representation of pedigrees as Bayesian networks is described. Then, in Chapters 14 and 15, it will be discussed how the blocking Gibbs sampler can solve problems of reducibility and multimodality (near-reducibility) in genetics problems by blocking the correct variables. In Chapter 16, the blocking Gibbs method has been applied to pedigree analysis, i.e., estimating marginal conditional probabilities in a pedigree of 20,000 pigs. In Chapter 17, the blocking Gibbs method has been applied to a complex linkage analysis problem. In Chapter 18, the algorithm for finding the noncommunicating classes by Lin et al. (1994) will be described, its limitations will be examined, and some suggestions towards a general algorithm will be provided.

Chapter 12

Basic Genetics

In this section, a brief introduction to the basic genetics concepts and terminology used in the thesis will be given. For a good book on pedigree analysis, consult (Thompson 1986). For a good book on linkage analysis, consult (Ott 1991).

Genetics is an area within science that deals with the topics associated with genes, what they are, what processes are involved with their expression and inheritance, etc. A gene is seen as a fundamental building block of living organisms, affecting in some way the structure and composition of the organism. A gene can have several types depending on its effect on observed traits. The types of a gene affecting the same trait are known as *alleles*. Genes usually come in pairs (in almost all living organisms), denoted *genotypes*. A genotype thus consists of two genes.

Consider a gene with two alleles, B and R, affecting the colour of a flower, B tending to produce blue flowers, and R tending to produce red flowers. Thus, every flower now has a genotype consisting of two genes, controlling its colour. In this case, there are three possible genotypes, BB, RR, and RB (the genotypes BR and RB are equivalent, thus usually only one of them is shown). In general, if there are n possible alleles, there are $\frac{n\cdot(n-1)}{2}$ possible genotypes. Further, individuals with genotypes consisting of two identical alleles (e.g., BB), are denoted *homozygotes*, and individuals with two different alleles (e.g., RB) are denoted *heterozygotes*. In this case, the genotype BB would produce a blue flower, RR would produce a red flower, and RB, might produce either a blue, red, or mixed colour flower, depending on the *dominance* of the alleles, as described later.

Further, the genetic material of a living organism consists of many genes; the entire set of which is denoted the *genome*. They are organized in chromosomes which are strings of genes. In a human being, every normal cell contains 46 chromosomes. 44 of these are denoted *autosomes*, arranged in 22 pairs. They are denoted such as one of each pair is received from the mother, and one is received from the father. The remaining two chromosomes are the *sex chromosomes*. These are denoted such as they differ depending on the sex. For females, they are a pair of autosomes, denoted *X chromosomes*, and for males, they are one X chromosome and one *Y chromosome*, received from the father.

The genes are located somewhere on the chromosomes; this location is denoted the *locus* of the gene. We thus, for instance, talk of a *diallelic locus*, if the gene at the locus has two alleles. In some situations we also state that a certain locus corresponds to a specific trait, e.g., the flower colour trait discussed above is controlled by a gene at a specific locus. Traits that are controlled by only a single locus, are naturally denoted *single-locus traits*, but we may also have traits that are affected by genes at several loci.

As mentioned earlier, genes are said to *segregate* describing the process of their inheritance from parents to offspring. *Segregation* occurs in the following way according to Mendel's First Law (Mendel 1866) (thus denoted *Mendelian segregation*). Each individual carries two genes, and when a male individual mates with a female, the offspring receives one gene from the male parent (denoted the *paternal gene*), and one gene from the female parent (denoted the *maternal gene*). Further, when an offspring is formed, a random one of the two genes is passed (i.e., *segregates*) to the offspring. The segregation occurring when forming multiple offspring is assumed to be independent. This is illustrated in Figure 12.1, where there are two parents, A and B, with the genotypes 12 and 34. With these parental genotypes, there are four possible genotypes for the offspring, illustrated with the offspring in Figure 12.1. The probability of each of these offspring resulting from the mating is $\frac{1}{4}$.

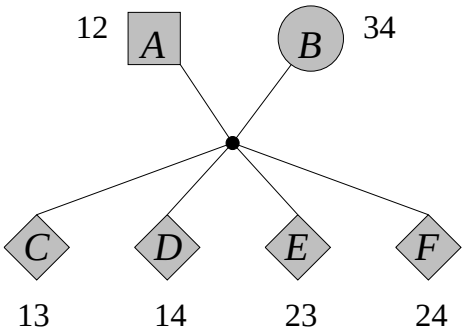


Figure 12.1: A small pedigree illustrating Mendelian segregation and the notation used for drawing pedigrees in the thesis — also denoted the *marriage node graph representation*. Boxes represent males, circles represent females, and diamonds represent individuals of unknown (and irrelevant) sex. Further, individuals are grey if they have been observed.

As previously mentioned, the conjunction of two genes in diallelic organisms is denoted the genotype. For instance, the genotype controlling the previously mentioned flower colour could be RR. In this case, the flower would be red, but still the genotype cannot be directly observed just by considering the colour of the flower. It might also be, e.g., RB if R *dominates* B as explained in the following. The observable characteristic of an individual wrt. some genotype is denoted the *phenotype*. An individual may have several possible phenotypes corresponding to a set of genotypes at a specific locus, the relationship between which can be specified. In the case of diallelic loci, say, with alleles R and B there are three possible genotypes that may correspond to different phenotypes dependent on the type of the alleles. If individuals carrying RB and individuals carrying RR have identical phenotypes (e.g., both flowers are red), the R allele is said to be *dominant* (or dominant to B) and the B allele is said to be *recessive* (or recessive to R). If all three genotypes result in different phenotypes, the alleles are *codominant*. Further, if individuals exhibiting a certain trait must carry a specific allele, R and this allele is dominant, then the trait is said to be dominant. Similarly, if the allele responsible for a specific trait is recessive, we have a recessive trait. Also, we denote individuals who carry the allele responsible for the trait as *carriers* wrt. the trait. The relationship between the genotypes and the expressed phenotypes can be specified using the so-called *penetrance probabilities* that defines the probability of each phenotype given the genotype. Finally, if the genotypes result in different phenotypes, we have

complete penetrance,, otherwise, we have *incomplete penetrance*. In Table 12.1, the penetrance probabilities for the human $AB0$-blood system are shown. As the A and B alleles are codominant, and both dominate the 0 allele, there are only four phenotypes in this system, denoted A, B, AB, and 0, as shown in Table 12.1.

	Phenotype			
Genotype	A	AB	B	0
AA	1	0	0	0
AB	0	1	0	0
$A0$	1	0	0	0
BB	0	0	1	0
$B0$	0	0	1	0
00	0	0	0	1

Table 12.1: The penetrance probabilities for the human $AB0$-blood system are shown, i.e., $P(\text{phenotype}|\text{genotype})$.

Alleles occur with different frequencies in different populations. For instance, the frequencies of the alleles in the $AB0$-blood system varies greatly in different human populations. The allele frequencies are usually specified with the *population allele frequencies* which is the probability that a random chosen gene in the population is a specific allele. The population allele frequencies are thus only valid for a specific population. They allow us to further calculate the *genotype frequencies*, i.e., if with a diallelic locus with alleles, A and B, the frequency of A is p_A, then the genotype frequency for AA is p_A^2, for BB, it is $(1-p_A)^2$, and for AB, it is $2p_A(1-p_A)$ (as AB and BA are equivalent). Further, in a very large population where the individuals mate completely at random, the genotype frequencies will remain the same at each generation. The population is then said to be in *Hardy-Weinberg equilibrium*, meaning that the genotype frequencies in the population depend only on the allele frequencies (Hartl 1988).

Mendel (1866) states as his Second Law that the genes controlling different traits are independently inherited. Unfortunately, this is not true. If an individual with two loci, L_1 and L_2, has paternal gene $g^p_{L_1}$ at locus L_1, paternal gene $g^p_{L_2}$ at locus L_2, maternal gene $g^m_{L_1}$ at locus L_1, and maternal gene $g^m_{L_2}$ at locus L_2, then offspring of this individual will tend to receive either both of the paternal genes, $g^p_{L_1}$ and $g^p_{L_2}$, or both of the maternal genes, $g^m_{L_1}$ and $g^m_{L_2}$. However, the tendency depends on whether the two loci are on the same chromosome, and if so, how close they are. If the two loci are on different chromosomes, there will be equal chances of receiving any two of the four genes. However, even if the two loci are on the same chromosome, it is possible for a *recombination* to occur between them, and thus cause the offspring to receive either $g^p_{L_1}$ and $g^m_{L_2}$, or $g^m_{L_1}$ and $g^p_{L_2}$. The probability of a recombination occurring between two loci depends on the distance between them. The closer they are, the smaller the probability of recombination, the *recombination fraction*,. In this thesis, the recombination fraction will usually be denoted θ. In general, if the two loci are far apart or on two different chromosomes, the recombination fraction will be $\frac{1}{2}$ and Mendel's Second Law holds.

In Figure 12.2, the segregation of two loci are shown for two individuals with four offspring. The four offspring represent the four possible outcomes when one of the parents is homozygous at both loci. B has been made homozygous at both loci to simplify the outcomes, as then the offspring will receive the same combination from B whether recombination occurs or not. Had B been heterozygous at both loci, there would have been eight possible outcomes. Thus, in Figure 12.2, offspring

E and F represent the two possible combinations when a recombination occurs. As the probability of recombination is θ, and the two offspring have equal probability, they must each have a probability of $\frac{\theta}{2}$ of occurring. Similarly, offspring C and D represent the two possible combinations when no recombination occurs. Again, as they have equal probability, they must have probability $\frac{1-\theta}{2}$ each.

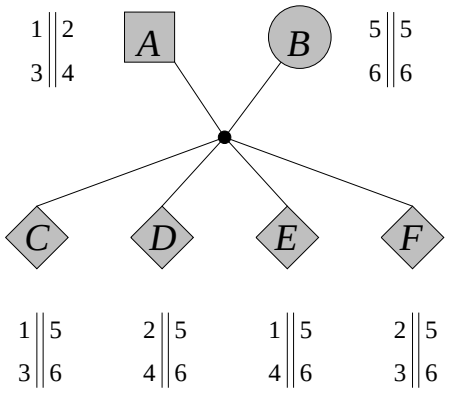


Figure 12.2: This figure shows the four possible combinations when two loci are inherited and one of the parents is homozygous at both loci. We use the following notation; the chromosome is represented with the parallel lines between the genes, and the two top genes are at locus 1, while the two bottom genes are at locus 2. Further, the two left genes are the paternal genes, and the two right genes are the maternal. In this situation, B is homozygous at both loci to make the situation simpler. The probability of each combination being inherited is; C: $\frac{1-\theta}{2}$, D: $\frac{1-\theta}{2}$, E: $\frac{\theta}{2}$, and F: $\frac{\theta}{2}$.

When performing *linkage analysis*, the goal is to estimate the recombination fraction between genes at two loci. If a small recombination fraction is estimated, say, 0.1, we have evidence that the loci are close on the same chromosome. We thus say that *linkage* has been established. Often, when performing linkage analysis one of the two genes affects a specific observable trait, i.e., a disease, and the other gene is a so-called *marker*, i.e., a gene whose location is known (with some uncertainty).

When performing linkage analysis, the process of which will be further explained in Chapter 17, the two basic hypotheses are, no recombination H_0 ($\theta = \frac{1}{2}$) and linkage H_1 ($\theta < \frac{1}{2}$). Usually, the decimal logarithm of the likelihood ratio

$$\mathrm{LOD}(\theta) = \log_{10} \frac{L(\theta)}{L(0.5)} \tag{12.1}$$

is used as the measure for statistical support for linkage. As indicated, Expr. (12.1) is denoted the *lod score*. For example, if we have n observations consisting of k recombinations and $n-k$ nonrecombinations, the lod score can be found as follows (for $\theta > 0$):

$$\begin{aligned}
\mathrm{LOD}(\theta) &= \log_{10}\left(\frac{\theta^k(1-\theta)^{n-k}}{\left(\frac{1}{2}\right)^k\left(\frac{1}{2}\right)^{n-k}}\right) && (12.2)\\
&= \log_{10}\left(2^n\theta^k(1-\theta)^{n-k}\right) && (12.3)\\
&= n\log_{10}(2) + k\log_{10}(\theta) + (n-k)\log_{10}(1-\theta). && (12.4)
\end{aligned}$$

If $\theta = 0$, all the n observations must be nonrecombinations, thus $k = 0$, and as 0^0 is defined to be 1, the lod score is $n \log_{10}(2)$. Thus, positive lod scores indicate evidence for linkage, and negative lod scores indicate absence of linkage. When the lod score exceeds a certain critical value, Z_0, the data are said to convey *significant evidence for linkage*. The critical value generally used is the one originally proposed by Morton (1955), $Z_0 = 3$.

When performing *pedigree analysis* (the general activity of analyzing aspects of a pedigree, including, e.g., linkage analysis), a number of assumptions are usually made, some of which have been mentioned above. First, it is assumed that Hardy-Weinberg equilibrium holds which again is based on the assumption that we have an infinite (or very large) population and that the individuals mate completely at random. Obviously, individuals do not mate completely at random, otherwise there would be no evolution. Another assumption is that the segregation of genes to different offspring is assumed to be independent which also does not hold completely in all cases. Also, usually, Mendelian segregation is assumed, but it has been shown for instance, that disease alleles are, for some conditions, preferentially received from the father or from the mother. Mutation is the process of a random change in alleles when segregating. Mutation occurs very rarely, and usually it is assumed that no mutations has occurred in the considered pedigree data. Finally, when performing linkage analysis, it is a frequent assumption that the recombination fraction is the same for all involved individuals. It has been found, however, that it is different for males and females, that it varies from region to region in the genome, and that further, it is dependent on age. However, all of these assumptions are usually reasonable as all of the above special cases occur rarely and with small significance on results.

Chapter 13

Representation

In this chapter it will be explained how to represent pedigree data with Bayesian networks. Pedigree data will be understood as a pedigree structure containing only information regarding the familial relations, information pertaining to the population as a whole such as population allele frequencies, penetrance probabilities, etc., and data on the individuals such as genotypes, phenotypes, etc. In general, a pedigree is understood as a pedigree structure with any amount of information on population and individuals present.

13.1 Genotype Representation

A pedigree can immediately be represented as a Bayesian network. In Figure 13.1(a), a pedigree is shown using the ordinary marriage notation. In (b), the same pedigree is represented as a Bayesian network. The sex of the individuals are not represented in the Bayesian network, as this information is usually not used when performing pedigree analysis. If necessary, variables representing the sex can easily be introduced. Each variable in the network thus represents the genotype of the corresponding individual in the pedigree. This representation will be denoted the *genotype representation*.

As mentioned in Section 3, we have to specify marginal probabilities for the variables representing the founders, and conditional probability tables for the variables representing their descendants. Assuming that we are considering a diallelic locus in Figure 13.1 with alleles N and n, and N having the population frequency of p_N, we use the population genotype frequencies to specify the marginal probabilities for the founder variables, see Table 13.1, and the Mendelian segregation probabilities to specify the conditional probability tables of descendants, see Table 13.2.

	NN	Nn	nn
a	p_N^2	$2p_N(1-p_N)$	$(1-p_N)^2)$

Table 13.1: The population genotype frequencies and the a priori probability distributions for the genotype of a in Figure 13.1(b). The same distribution is used for b and c.

However, we will most likely also want to model the penetrance of the gene under consideration. Assuming that the trait affected by the gene of Figure 13.1 has two different phenotypes, we can model the relationship between the two phenotypes and

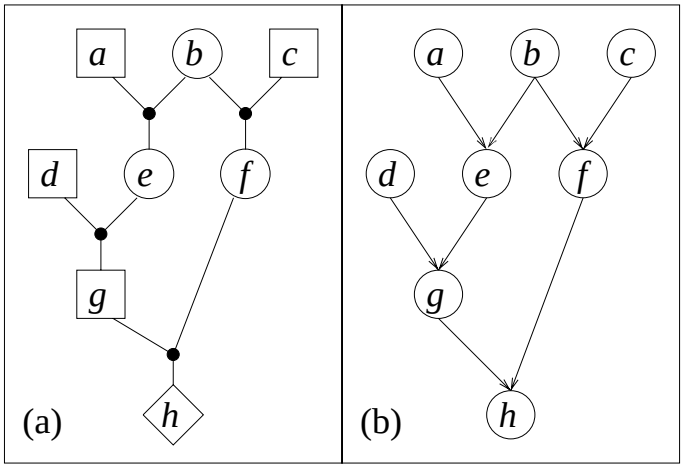


Figure 13.1: The representation of a pedigree as a Bayesian network; the two notations.

a		NN			Nn			nn		
b		NN	Nn	nn	NN	Nn	nn	NN	Nn	nn
e	NN	1	0.5	0	0.5	0.25	0	0	0	0
	Nn	0	0.5	1	0.5	0.5	0.5	1	0.5	0
	nn	0	0	0	0	0.25	0.5	0	0.5	1

Table 13.2: The Mendelian segregation probabilities and the conditional probability distribution of e given the parents a and b in Figure 13.1(b). The same distribution is used for d, f, g, and h.

the three genotypes (NN, Nn, and nn) using the penetrance probabilities. These aspects are incorporated in the extended Bayesian network in Figure 13.2(b), representing the individuals' phenotypes as well. The variables representing phenotypes are simply appended to the network in (a), by placing each of them as an offspring of their corresponding genotype variable. This also agrees with the causality that is believed inherent in the relationship between genotype and phenotype, i.e., the phenotype is usually understood as causally dependent on the genotype.

We also have to specify the conditional probability distributions of the phenotype variables. This is done by using the penetrance probabilities. Assuming that the trait affected by the gene in Figure 13.2 is dominant, and N is the dominant gene, we will have two phenotypes with penetrance probabilities as shown in Table 13.3. These penetrance probabilities are used as the conditional probability distributions for the phenotype variables in Figure 13.2.

13.2 Gene Representation

However, the representation in the previous section does not allow us to represent knowledge of paternal and maternal genes. We can incorporate this knowledge by explicitly representing each gene as an individual variable. This representation is inspired by Kong (1991) and will be denoted the *gene representation*. Figure 13.3

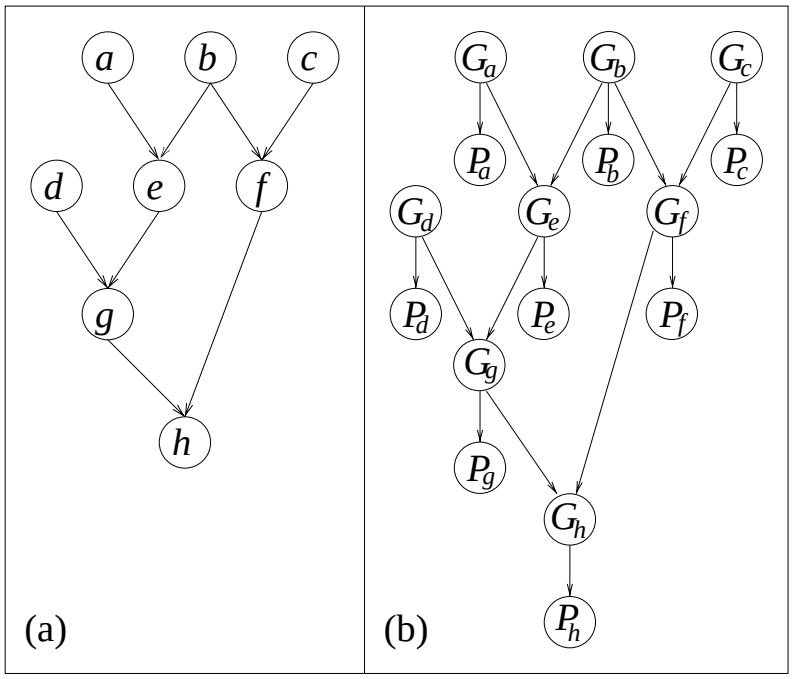


Figure 13.2: In (a), the simple network representation of a pedigree is shown. In (b), the relation between the individuals' genotypes and phenotypes has been represented as well. Also, the notation is expanded, i.e., the variable G_a represents the genotype of individual a, and P_a represents the phenotype of a.

	P_a	
G_a	N	n
NN	1	0
Nn	1	0
nn	0	1

Table 13.3: The penetrance probabilities for the gene in Figure 13.2(b) which are used for the conditional probability distribution of P_a given G_a. The same distribution is used for the other phenotype variables.

illustrates the genotype and gene representations of the pedigree. As seen, the gene representation is a more complete model of the segregation where also the relationship between the genes of the parents and offspring is represented. From Figure 13.1(a), we can see that a is the father of e, and b is the mother. This is represented in Figure 13.3(b) by indicating that the paternal gene of e, e^f, can only originate from a, thus the two links from a^f and a^m to e^f. Similarly, the maternal gene of e, e^m, can only originate from b. Thus, with this representation the genotypes Nn and nN will not be equivalent, as it will explicitly be represented which parent each of the genes originate from.

The conditional probability distributions for the gene variables are simple, see Table 13.4 for a diallelic example. Even though it is necessary to specify more condi-

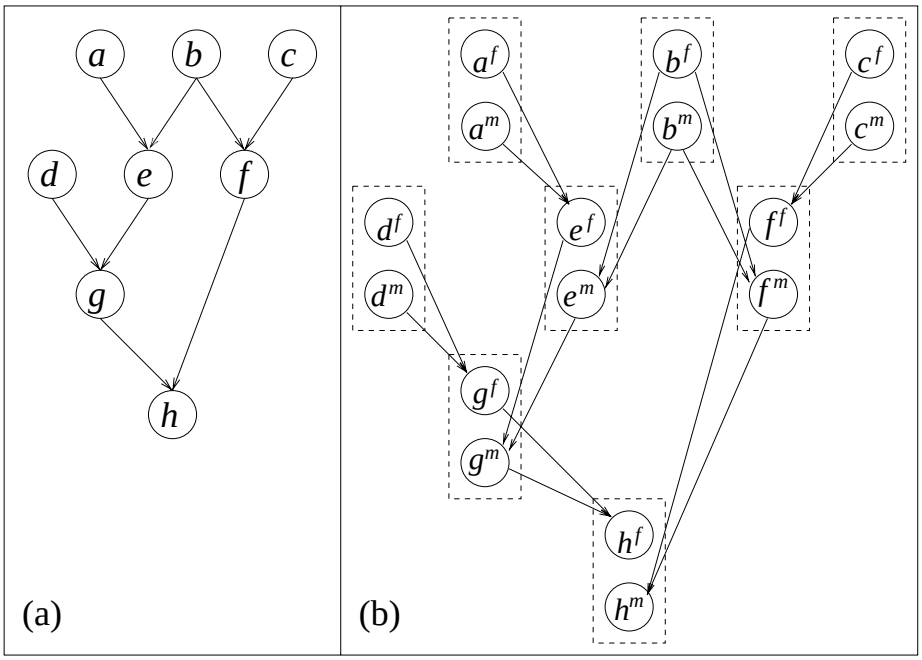


Figure 13.3: This figure illustrates two possible representations of pedigrees with Bayesian networks. In (a), the *genotype representation* is shown, and in (b), the *gene representation* is shown. The notation is as follows; a^f denotes the paternal gene (f for father) of A, and a^m denotes the maternal. Each pair of variables belonging to the same individual is surrounded by a dashed box.

tional probability distributions with the gene representation, they are much smaller than with the genotype representation, such that the overall storage requirements of the gene representation are lower. For example, if the networks in Figure 13.3 model a diallelic trait, the storage requirements of (a) will be 162 (two alleles yields three states for each genotype variable, and when the graph is triangulated, 6 cliques with three members are created, yielding storage requirements $6 \cdot 3^3 = 162$) and for (b) only 64 (each variable has two states, and when the graph is triangulated, 8 cliques with three members are created, yielding storage requirements $8 \cdot 2^3 = 64$). This is also due to the fact that, as seen in Figure 13.3, loops may be broken when using the gene representation, as further independence relations is exploited (the paternal gene is only dependent on the genes of the father, etc.). Furthermore, as previously mentioned this representation takes more information about segregation into account. In Kong (1991), the benefits of the gene representation are further discussed.

When using the gene representation as exemplified in Figure 13.3(b), we do not get an explicit marginal distribution of the genotypes of the individuals. However, this can easily be obtained by appending a genotype variable as an offspring of the corresponding gene variables, as shown in Figure 13.4. However, this removes the advantage of lower storage requirements that the gene representation had over the genotype representation, as it is then not possible to exploit the additional independence relations. The conditional probability distributions for the genotype variables are very simple, as shown in Table 13.5. With the genotype representation,

		e^f	
a^f	a^m	N	n
N	N	1	0
N	n	0.5	0.5
n	N	0.5	0.5
n	n	0	1

Table 13.4: The conditional probability distribution of the paternal gene of e, e^f, given the genes of the father, a^f, a^m. The other distributions are similar to this.

loci with incomplete penetrance could easily be represented by appending phenotype variables (see Figure 13.2). This is no more difficult with the gene representation (see Figure 13.5). Here, as earlier, the phenotype variables are simply placed as offspring to the genotype variables following the assumed inherent causality, and also here conditional probability distributions similar to the ones in Table 13.3 are used, depending on the dominance of the genes.

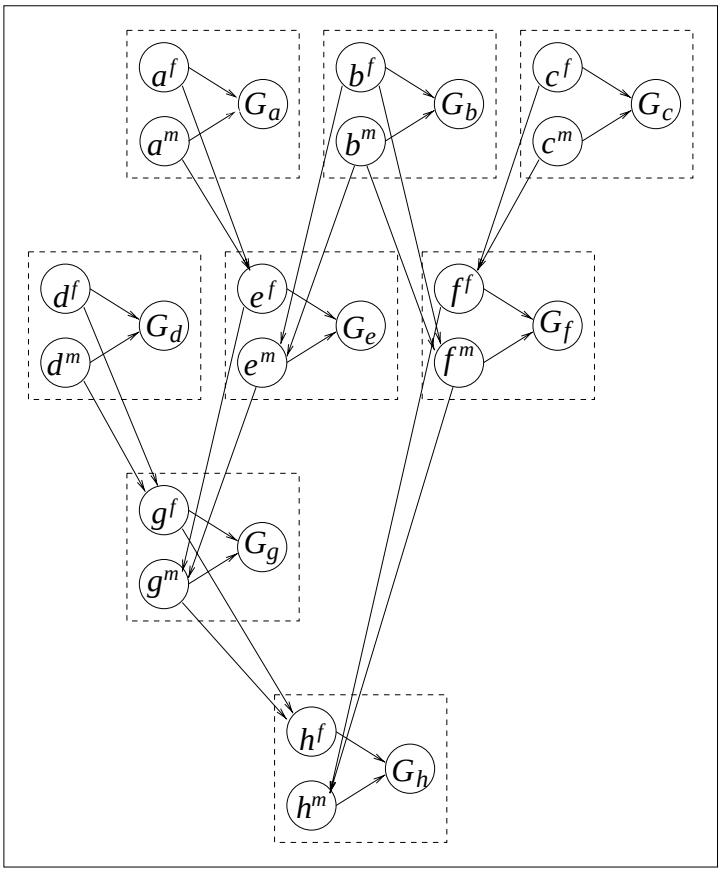


Figure 13.4: The gene representation with explicit variables representing the genotypes. Again, each group of variables belonging to the same individual is surrounded by a dashed box.

		G_a		
a^f	a^m	NN	Nn	nn
N	N	1	0	0
N	n	0	1	0
n	N	0	1	0
n	n	0	0	1

Table 13.5: The conditional probability distribution of the genotype variable, G_a, given the genes, a^f and a^m. The other genotype variables have identical distributions.

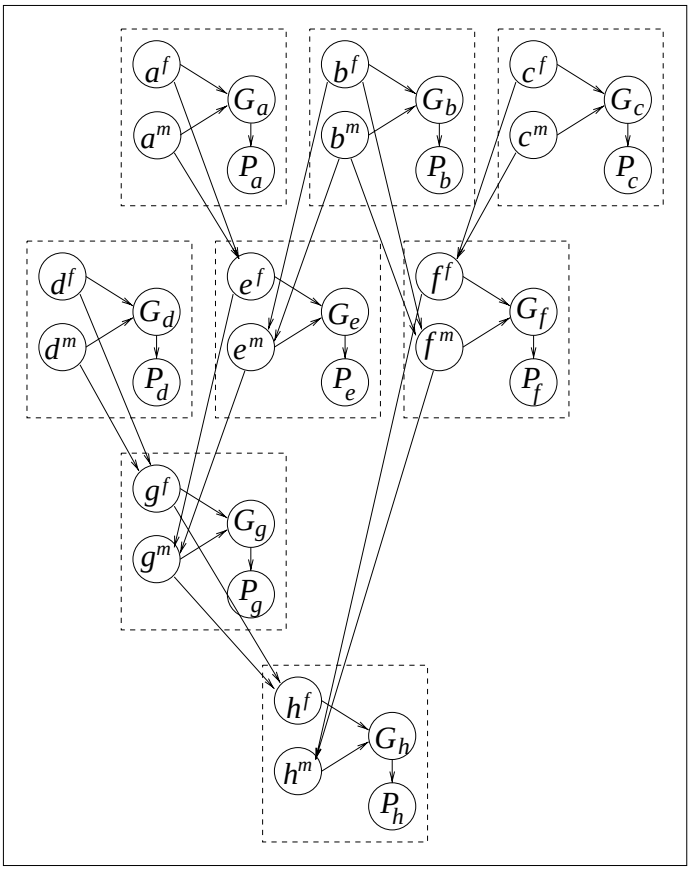


Figure 13.5: A Bayesian network using the gene representation for representing a locus with incomplete penetrance, thus the phenotype variables as offspring to the genotype variables.

13.3 Linkage Representation

The linkage representation is based on the gene representation but takes two (or more) loci into account. In Figure 13.6, a situation with two loci is shown. In this figure we do not use the pedigree of earlier figures, but only a simple three member pedigree with parents A and B, and offspring C. New notation has been introduced to take care of the extra information present with two loci and the variables are now

represented with small circles to avoid cluttering the figure. There are still four gene variables for each individual, A, B, and C, with an extra subscript denoting the locus, a (marker) or d (disease). In addition there are now four indicator variables for each individual : $Z^A_{C,a}$, $Z^A_{C,d}$, $Z^B_{C,a}$ and $Z^B_{C,d}$. The indicator variable $Z^A_{C,a}$ takes on the value 0 if individual C inherits A^f_a from its father A, and takes on the value 1 if individual C inherits A^m_a from its father. Similarly, $Z^B_{C,a}$ takes on the value 0 if individual C inherits B^f_a from its mother B, and takes on the value 1 if it inherits B^m_a instead. The other two indicator variables related to the disease gene are similarly defined. The joint distribution of $Z^A_{C,a}$ and $Z^A_{C,d}$ is :

$$P_\theta(Z^A_{C,a}, Z^A_{C,d}) = \begin{cases} (1-\theta)/2 & \text{if } (Z^A_{C,a}, Z^A_{C,d}) = (0,0) \\ (1-\theta)/2 & \text{if } (Z^A_{C,a}, Z^A_{C,d}) = (1,1) \\ \theta/2 & \text{if } (Z^A_{C,a}, Z^A_{C,d}) = (0,1) \\ \theta/2 & \text{if } (Z^A_{C,a}, Z^A_{C,d}) = (1,0) \end{cases} \tag{13.1}$$

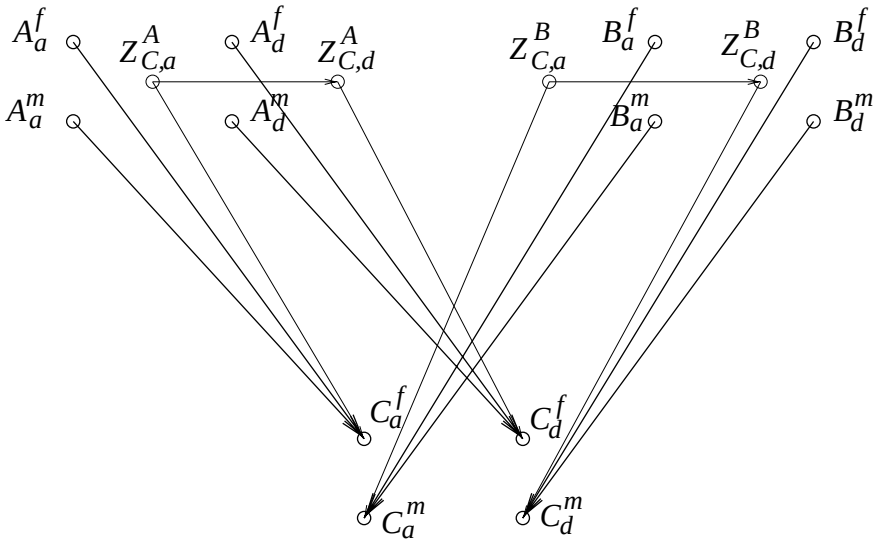


Figure 13.6: The representation of the two-locus linkage problem. As before, A^f represents the gene of individual A inherited from its father. The subscripts denote the locus of the gene, i.e., a for marker and d for disease. There are thus four gene variables for each individual. For each individual there are four indicator variables : $Z^A_{C,a}$, $Z^A_{C,d}$, $Z^B_{C,a}$ and $Z^B_{C,d}$. Only the indicator variables of C are shown here.

The indicator variables $Z^B_{C,a}$ and $Z^B_{C,d}$ have a joint distribution given as in Eq. (13.1). Whenever two associated indicator variables have different values a recombination has occurred. The marker locus indicator variable is a parent of the disease locus indicator variable, e.g., $Z^B_{C,a}$ is a parent of $Z^B_{C,d}$ (see Figure 13.6. For the marker locus indicator variable a simple uniform distribution is used, $\{0.5, 0.5\}$, as there are equal chances for inheritance of each of the genes at the marker locus. The disease locus indicator variable is given a distribution corresponding to Eq. (13.1), see Table 13.6.

$Z^B_{C,a}$	$Z^B_{C,d}$	
	0	1
0	$(1-\theta)/2$	$\theta/2$
1	$\theta/2$	$(1-\theta)/2$

Table 13.6: The conditional probability distribution of the disease locus indicator variable, $Z^B_{C,d}$, given the marker locus indicator variable, $Z^B_{C,a}$. The other disease locus indicator variables have identical distributions.

Chapter 14

Reducibility Problems

In this chapter, simple examples of genotypic configurations causing reducibility problems for the Gibbs sampler are presented.

Problem 1: First, considering pedigree analysis with more than two alleles, problems can occur in situations like the one in Figure 14.1. In this case, the single-site Gibbs sampler will be reducible, due to the fact that it will be unable to switch between the configurations $(A = \texttt{12}, B = \texttt{13})$ and $(A = \texttt{13}, B = \texttt{12})$. However, if blocking Gibbs was applied and A and B were sampled jointly, the sampler would be irreducible. The reducibility problem of Figure 14.1 is very common in pedigrees, and can easily appear in much more complex situations, further discussed in Chapter 18.

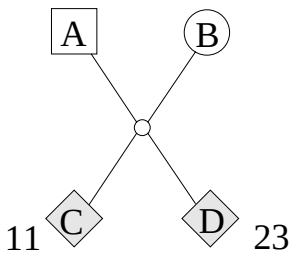


Figure 14.1: A small pedigree causing reducibility problems for the Gibbs sampler due to noncommunicating configurations of A and B.

Problem 2: If the gene representation is used (see Chapter 13) a simple reducibility problem can occur if an individual has been observed to be heterozygous. Considering individual a in Figure 13.3 on page 90, if it has been observed to have genotype Nn, then the single-site Gibbs sampler will be unable to switch between the configurations $(a^f = N, a^m = n)$ and $(a^f = n, a^m = N)$. The blocking Gibbs sampler can easily resolve this problem by sampling a^f and a^m jointly.

Problem 3: If the penetrance probabilities are defined such that one phenotype p_1 can correspond only to one homozygous genotype, e.g., 11, and another phenotype p_2 can correspond only to genotypes different from 11, then the pedigree in Figure 14.2 will be reducible with single-site Gibbs sampling. Again, the solution to the problem in Figure 14.2 is to sample the variables B^f and B^m

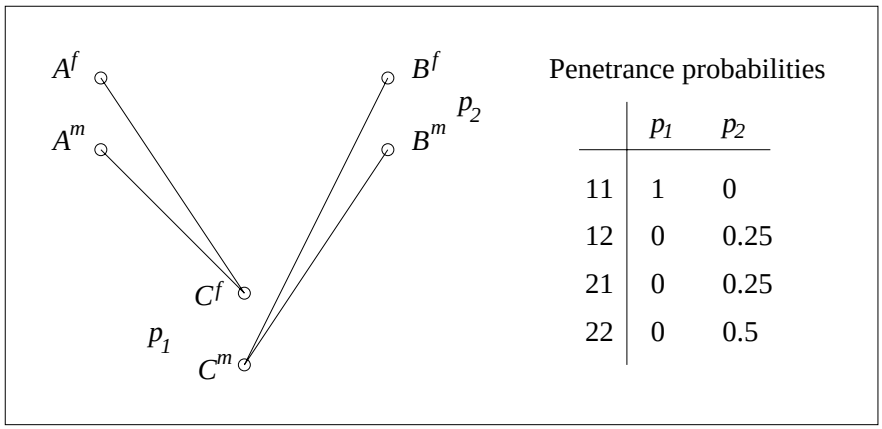


Figure 14.2: Here, the fact that C has genotype 11 forces B with phenotype p_2 to have genotype 12 or 21.

jointly thus allowing switching between the configurations $(B^f = 1, B^m = 2)$ and $(B^f = 2, B^m = 1)$. This problem appears with several individuals in Figure 8.3 on page 52, e.g., individual 56 which has the disease phenotype forces non-diseased individual 54 to have one of the noncommunicating configurations $(54^f = 1, 54^m = 2)$ or $(54^f = 2, 54^m = 1)$.

Problem 4: When the linkage representation is used, further problems can arise. Considering Figure 17.2 on page 118 with the shown configuration, and further observing $Z^{18}_{40,a} = 0$, indicating that the gene 18^f_a (the paternal gene of 18 at the marker locus) is inherited to 40^f_a (the paternal gene of 40 at the marker locus) and is thus 1. Now, the single-site Gibbs sampler will be unable to switch between the two configurations $(Z^{18}_{40,a} = 0, 40^f_a = 1)$ and $(Z^{18}_{40,a} = 1, 40^f_a = 2)$. With the blocking Gibbs sampler it is easy to update the two variables jointly.

For most of the above problems, automatic methods can be constructed to locate them in the pedigree, particularly for the simple problems of Problem 2–4. However, no method currently exists for automatic construction of blocks that guarantee resolving all problems of type 1. The automatic construction of blocks based on reduction of storage requirements described in Chapter 9 can usually create blocks containing more than 90% of the variables, and thus a very large proportion of the variables will be updated jointly. This will in many cases render the blocking Gibbs sampler irreducible, even in the absence of a method for handling Problem 1. Methods for handling problems of type 1 are discussed further in Chapter 18.

Chapter 15

Near Reducibility Problems

The problem of near reducibility (or multimodality) was introduced in Section 5.2.1. It is a very serious problem that when present in a Gibbs sampling application, can severely slow the mixing of the induced Markov chain. The sample space will look like Figure 5.1 on page 36 with two (or more) almost disconnected sets. In this chapter, this problem will be further discussed along with the presentation of some common situations in pedigree analysis where it appears. First, these problems will be presented.

Problem 1: This situation is illustrated in Figure 15.1 where two parents, A and B, have k offspring, $O_1, \ldots, O_k$. Suppose the two alleles, N and n are equally likely and there are no observed data on any member of the family. In that case, there is probability $\frac{1}{16}$ that all members of the family have genotype NN as the probability of each parent being NN is $\frac{1}{4}$. By symmetry, there is probability $\frac{1}{16}$ that all members have genotype nn. Applying the single-site Gibbs sampler to this problem with the initial configuration shown in the figure, we can compute the probabilities of moving into other configurations. Obviously, sampling any of the offspring will not change the joint configuration, as they are forced to have the genotype NN by their parents. Thus, to change the configuration of Figure 15.1, we have to sample either A or B. Arbitrarily choosing to sample A first, we have to compute the probabilities specified in Thm. (5.2.1) on page 36, i.e., $p_s = P(A = s)P(o_1|A = s, b) \cdots P(o_k|A = s, b)$ for each state s of A. A has the states NN, Nn and nn, and for each of these states the probability of sampling it, p_s, is proportional to :

$$p_s \propto \begin{cases} \frac{1}{4} \cdot 1 \cdots 1 = \frac{1}{4} & \text{if } s = NN \\ \frac{1}{2} \cdot \frac{1}{2} \cdots \frac{1}{2} = \left(\frac{1}{2}\right)^{k+1} & \text{if } s = Nn \\ \frac{1}{4} \cdot 0 \cdots 0 = 0 & \text{if } s = nn \end{cases} . \tag{15.1}$$

The probability of moving into state Nn is thus very small for large k. The probabilities for changing the configuration of B are similar, and even if either A or B was changed to Nn, they would have a very large chance of being moved back into state NN next time they were sampled. Thus, if A or B was changed to Nn, this is only a small step towards moving to the configuration of all nn. If k is very large, it is therefore obvious that the single-site Gibbs sampler will be stuck in the same configuration for long periods of time causing extremely slow mixing. This problem can easily be relieved by the blocking Gibbs sampler, as it can update all the variables jointly. A block for the joint updating of the individuals in Figure 15.1 requires very low storage requirements, proportional with the number of offspring as no large cliques

are created. In general, problems like this can be solved by simply making sure that each family is updated jointly at least once in each iteration.

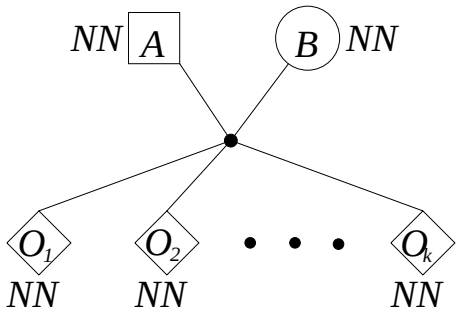


Figure 15.1: Two parents, A and B, with k offspring, $O_1, \ldots, O_k$. Initially, they are all in the state NN, and the probability of moving into other states will be very small if k is large.

Problem 2: Consider the pedigree in Figure 15.2. It represents a large pedigree with two founders, A and B, and a common descendant, O. We have a diallelic locus with two alleles, d and D, d being very rare, e.g., $p_d = 0.01$. O has been observed to dd and the d allele could only have originated from either A or B, or both of them. Now, as d is rare, the most likely configuration is that either A or B is a carrier and the other is not. Assuming we start in a configuration where only A is a carrier, we should be able to move to the configuration where only B is a carrier, as this configuration is equally likely. However, to get to this configuration, we have to move through the configuration where both A and B are carriers. This configuration has a probability proportional to $0.01 \cdot 0.01 = 0.0001$ and is thus very difficult to reach. If the probability of d is even lower, it will be increasingly difficult to reach the configuration of both founders being carriers. This problem can be solved by, for each individual

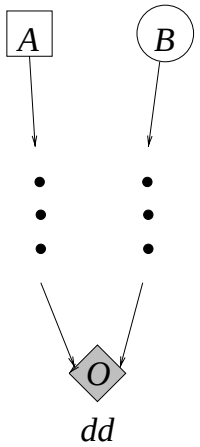


Figure 15.2: A large pedigree with two founders, A and B, and a descendant of them, O. O is observed to dd, and the rare d allele could only have originated from either A or B, or both of them.

carrying a d allele, to update all its ancestors jointly at least once in each iteration. These blocks can obviously become very large. The problem is present in the LQT pedigree in Figure 8.3 on page 52 where the probability of the disease allele is 0.05. In the LQT pedigree, for instance, individual 56 carries two disease alleles. One of these originates from the mother, individual 55, making it necessary to create a block containing all the variables representing individuals $55, 26, 24, 70, 71, 11, 16, 68, 69, 66, 67, 6, 4, 1, 2, 64, 65, 60, 61, 62$, and 63.

Chapter 16

Analysis of a Large Pig Pedigree

In this section, the blocking Gibbs sampler defined in Chapter 6 will be applied to a pedigree analysis study and evaluated wrt. this. The evaluation of the method will be conducted as an empirical comparison study of the convergence properties of single-site and blocking Gibbs for different size networks all of which are sub-networks of a real-world pedigree containing 20,000 breeding pigs. The pedigree is heavily inbred and several animals have an enormous amount of offspring which, in particular, causes serious problems for the convergence properties of single-site Gibbs, see Chapter 15. Based on the outcome of this study, we present rules of thumb and general guidelines to obtain an optimal compromise between complexity and rate of convergence.

In the evaluation, an earlier version than the methods presented in Chapter 9 was used for block selection. This version is described in Section 16.2.1.

First, we present the real-world problem used as the basis for the experiments, and we describe the prerequisites of the experiments. Two investigations are performed: a comparison of blocking and single-site Gibbs, and a sensitivity analysis of blocking Gibbs wrt. choice of parameter values.

16.1 A Real-World Problem

The experiments are based on an extremely complex real-world problem, namely estimation of genotype probabilities for individuals in a heavily inbred pedigree containing approximately 20,000 breeding pigs. The maximum number of generations from top to bottom of the pedigree is 13. Each individual may have a hereditary trait, PSE, which causes the meat to be unfit for human consumption. This trait is assumed to be dominant and diallelic, thus controlled by a gene with two alleles, N and n, yielding three genotypes NN, Nn, and nn. The n allele is dominant, thus the PSE disease is present only if the genotype is nn.

The pedigree data was provided by Søren Andersen of Danish Slaughter-Houses (*Danske Slagterier* in Danish).

16.2 Prerequisites of Comparison

Before describing the actual comparison study we shall describe the assumptions and conditions applied in the comparisons.

The comparisons were carried out on three subsets of the pedigree, referred to as Pedigree A (455 variables), Pedigree B (704 variables), and Pedigree C (1894 variables). We use these relatively small networks, since, for statistical purposes, a large number of long runs are performed on each network. Pedigree B was constructed from Pedigree A by adding a suitable amount of parents, offspring, and parents of offspring of the individuals in Pedigree A (of course, corresponding with the true pedigree). Similarly, Pedigree B is contained in Pedigree C which is contained in the complete pedigree. We chose to avoid evidence to ease the comparison task for reasons explained in the following.

We compute the convergence rate of a sampling scheme by comparing the resulting approximated marginal distributions with the correct ones, i.e., the equilibrium distribution of the associated Markov chain. Since our investigations are based on the absence of evidence, the correct distributions can be approximated very well by means of forward sampling (see Chapter 10) with a very large sample size of, e.g., $n = 1{,}000{,}000$. Given n and an estimate $\widehat{p}_{ij}$ of p_{ij} (the probability of variable i being in state j) obtained by forward sampling, a confidence interval can be computed for p_{ij} by utilizing the asymptotic behaviour of the distribution function for $\widehat{p}$.

The metric used for calculating the accuracy of a result is the average mean squared error,

$$\text{M.S.E.} = \frac{1}{|V|} \sum_{i=1}^{|V|} \frac{1}{|\text{Sp}(v_i)|} \sum_{j=1}^{|\text{Sp}(v_i)|} (\widehat{p}_{ij} - p_{ij})^2 , \tag{16.1}$$

where V is the set of variables.

Empirically, the model in Eq. (16.2) fitted well to the values of the average mean squared errors computed from the simulations. This can be seen in Figures 16.2–16.4 and 16.11. The model is

$$\text{M.S.E.} = \beta \cdot t^{\alpha} + \text{noise}, \tag{16.2}$$

where t is the time (iteration number), and noise is random and has expectation zero. As mentioned by, e.g., DeGroot (1986), the M.S.E. converges towards the variance, thus it is a good approximation for the variance of the estimate. The model specifies a linear relationship between log(M.S.E.) and $\log(t)$. In theory, for large enough t, which depends on the mixing rate of the Gibbs sampler, this model is supposed to hold approximately with $\alpha = -1$. If the fitted value of α is substantially bigger than -1, it is an indication that the particular Gibbs sampling scheme is mixing very slowly.

The comparison between blocking and single-site Gibbs is performed with suboptimal parameter values for blocking Gibbs. The reason for this is, that at the time of the comparison the optimal parameter values of blocking Gibbs were not known, and when they were later actively searched for, they were found to be different from those employed in this comparison. In Section 16.4, we shall conduct a sensitivity analysis to reveal the impact of a suboptimal choice of parameter values. The blocks were constructed according to the following two methods. Unless stated otherwise, Method 1 has been applied. It should be noted that the construction methods presented here are earlier versions than those presented in Chapter 9. They are presented in the three following sections.

16.2.1 Early Block Selection Method

These early block selection methods are based on slightly different and more primitive principles than those described in Chapter 9. Here, as before, we select k blocks, $B_1, \ldots, B_k$, by selecting the respective complementary sets of variables, $A_1, \ldots, A_k$. A_1 is thus created by removing "optimal" variables from the initial junction tree which is an approximation of the more intricate method of Chapter 9, however, here, the "optimality" of the variable is not based on its reduction in storage requirements, but rather on the number of cliques of which it is a member. Clearly, if a variable is a member of many cliques, it is also responsible for much of the storage requirements, however, this measure is not as accurate as the one used in Chapter 9. Still, it is adequate for the empirical investigations performed in Section 16.3.

Also, the blocks are constructed in a slightly different manner. First, all the variables are listed in order of "optimality", i.e., $\{v_1, \ldots, v_n\}$. Then, the first block, B_1, is constructed by removing a sufficient amount of these variables from the initial junction tree starting from the most "optimal", v_1, etc. Thus, A_1 consists of a subset $\{v_1, \ldots, v_r\}$, necessary for lowering the storage requirements of B_1 sufficiently. Thus, using A_1 as a base set, the following sets, $A_2, \ldots, A_k$, are constructed by substituting some of the variables in A_1 with variables from $\{v_{r+1}, \ldots, v_n\}$. When constructing the remaining A-sets, we must make sure that $\bigcap_{i=1}^{k} A_i = \emptyset$ to ensure that all variables get updated at least once in each iteration. Here, the remaining $k-1$ A-sets are constructed from A_1 by defining

$$A_i \cap A_1 = \{v_j \mid (j - i + 1) \bmod (k-1) \neq 0\},\ 1 < i \leq k, \tag{16.3}$$

which is also illustrated in Figure 16.1. Also, let $r_i \geq 1$ be the integer such that

$$\text{red. in storage req. of } \{v_{r+1}, \ldots, v_{r+r_i}\} \geq \text{red. in storage req. of } A_1 \setminus A_i. \tag{16.4}$$

Then

$$A_i \setminus A_1 = \{v_{r+1}, \ldots, v_{r+r_i}\}, \tag{16.5}$$

and

$$A_i = (A_i \cap A_1) \cup \{v_{r+1}, \ldots, v_{r+r_i}\}. \tag{16.6}$$

Thus, the variables $\{v_{r+1}, \ldots, v_{r+r_i}\}$ are included in A_i to make up for the removal of $A_1 \setminus A_i$. Then, it is obvious that the reduction in storage requirements of $\{v_{r+1}, \ldots, v_{r+r_i}\}$ must be greater than or equal to those of $A_1 \setminus A_i$, to ensure that we can update the sets, $A_2, \ldots, A_k$ jointly.

Using this as the basis, we define two variants of the construction method.

16.2.2 Construction Method 1

Using Method 1, $A_1 = \{v_1, \ldots, v_r\}$ is constructed as described in the previous section, and the remaining A-sets are constructed as described by Eqs. (16.3)–(16.5), with the exception that $r_2 = \cdots = r_k$ have been prespecified to $\lfloor \frac{n}{k-1} + 1 \rfloor$ (n variables shared equally between the k blocks) which appeared to be sufficient to allow joint updating of the resulting blocks, $B_2, \ldots, B_k$.

This method has the advantage that A_1 can be rather small, as the most "optimal" variables are contained in A_1. On the other hand, $A_2, \ldots, A_k$ necessarily must be larger than A_1.

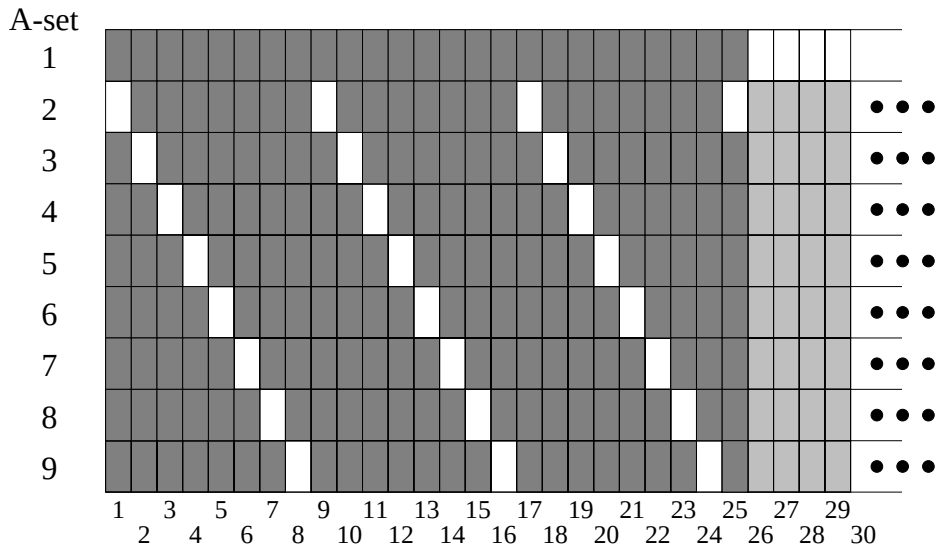


Figure 16.1: The selection of A-sets using Eq. (16.3) and (16.4) with $r = 25$. The dark grey squares denote variables that are included in the A-set according to Eq. (16.3), and the light grey squares denote the variables that are included in Eq. (16.5) to make up for the removal of $A_1 \setminus A_i$. It is seen that each of the "optimal" variables is included in all but one A-set, i.e., each variable is included in exactly one B-set, allowing each variable to be sampled once in each iteration. $r_2, \ldots, r_9$ are all greater than 4 and are thus not represented in the figure.

16.2.3 Construction Method 2

Alternatively, the A-sets could be constructed such that they are all of equal size. To construct k A-sets of size r, we select the set, A, of $r+s$ ($s > 0$) "optimal" variables such that s is large enough to enable the k A-sets to reduce storage requirements sufficiently. The A-sets are now given by

$$A \cap A_i = \{v_j \mid (j - i) \bmod k \neq 0\},\ 1 \leq i \leq k, \tag{16.7}$$

whereby we make sure that $\bigcap_{i=1}^{k} A_i = \emptyset$. This corresponds to only using $A_2, \ldots, A_9$ for constructing the complementary blocks $B_2, \ldots, B_9$ in Figure 16.1. We must, however, make sure that A contains a sufficient amount of the most "optimal" variables (i.e., by making s large enough to allow all the sets $A_1, \ldots, A_k$, to provide a sufficient reduction in storage requirements.

16.3 Comparison of Blocking and Single-Site Gibbs

We will now compare the rates of convergence for blocking and single-site Gibbs by measuring the average mean squared errors for various sample sizes. Below, we show and discuss the results of the comparisons for Pedigrees A–C. In this comparison, construction method 1 is used.

16.3.1 Pedigree A

Using the above terminology, the blocking Gibbs parameters are $k = 5$ (number of blocks) and $r = 50$ (size of A_1) for Pedigree A. The results presented in Figure 16.2 depict the average mean squared error of the estimates obtained for various sample sizes converted to time,[1] and each point denotes an independent run. From this figure we observe that blocking Gibbs converges much faster than single-site Gibbs. However, if we were limited in time to, say, 20 minutes to make our runs, single-site Gibbs would possibly provide the most accurate estimates. After this point, the precision of blocking Gibbs gets increasingly better at a faster rate than that of single-site Gibbs.

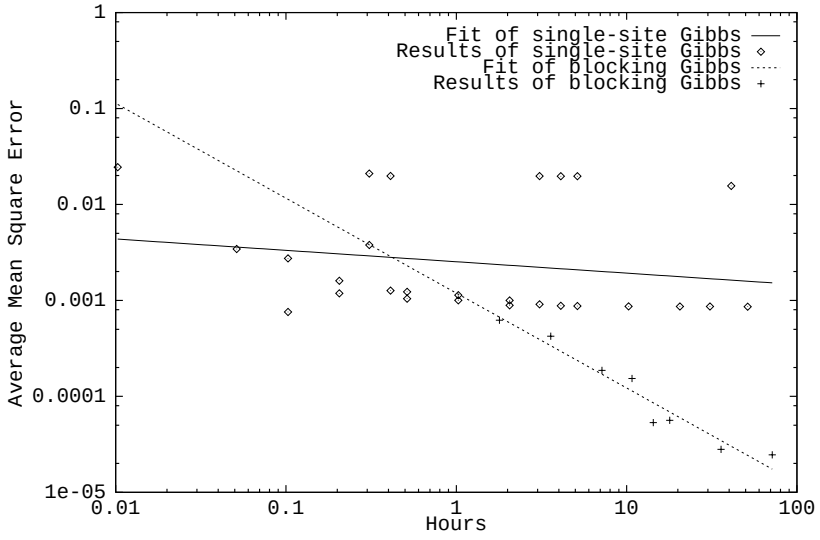


Figure 16.2: Blocking Gibbs vs. single-site Gibbs for Pedigree A.

We also observe that the measurements of blocking Gibbs can be fitted very nicely to a straight line, as opposed to those of single-site Gibbs, which seem to reside in two or more "modes".[2] We believe that this behaviour is caused by the presence of a single individual with many offspring. When such an individual is present, it will be very difficult to jump between modes, thus making the Markov chain get stuck in one of the modes. See Chapter 15 for a further elaboration on this issue.

Following Eq. (16.2), we extract α from the fitted lines. For blocking Gibbs, $\alpha \approx -1$ which is an outstanding result. It indicates that the blocking Gibbs sampler mixes extremely fast for the relatively large pedigree A. For single-site Gibbs, $\alpha \approx -0.13$. This indicates that in the case of Pedigree A, blocking Gibbs mixes at a much higher rate than single-site Gibbs.

16.3.2 Pedigree B

The blocking Gibbs parameters are $k = 5$ and $r = 100$ for Pedigree B. The results are presented in Figure 16.3. Again, blocking Gibbs converges faster than single-

[1] These and all subsequent results in this Chapter were obtained on a Sun 4-40 Workstation.
[2] The lines have been fitted to the M.S.E. measurements (cf. Eq. (16.2)) using linear regression.

site Gibbs, and better precision can be obtained with blocking Gibbs except for very short runs. Here, the measurements of single-site Gibbs do not indicate two or more distinct "modes" of the Markov chain, probably due to the fact that more than one individual of Pedigree B have many offspring, yielding a larger number of intervening modes.

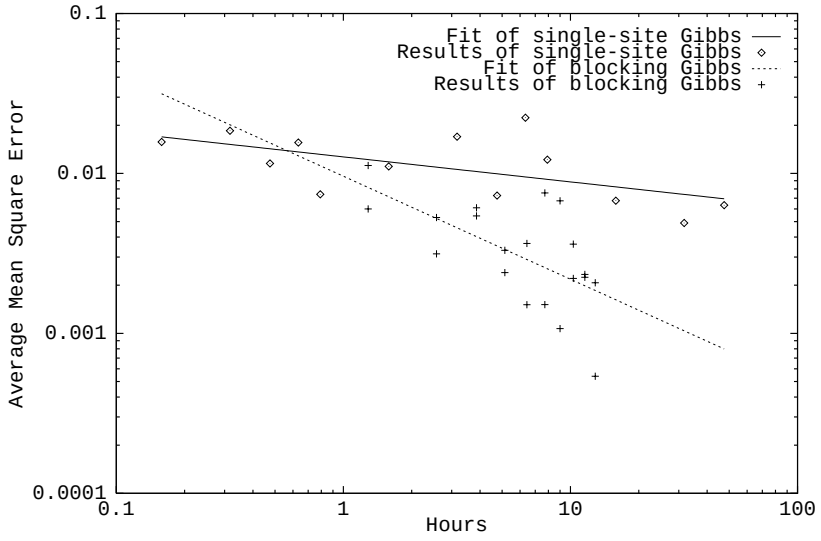


Figure 16.3: Blocking Gibbs vs. single-site Gibbs for Pedigree B.

Here, for blocking Gibbs, $\alpha \approx -0.64$, and for single-site Gibbs, $\alpha \approx -0.18$, again indicating that blocking Gibbs mixes much faster than single-site Gibbs.

16.3.3 Pedigree C

The blocking Gibbs parameters are $k = 5$ and $r = 200$ for Pedigree C. The results are shown in Figure 16.4. As in the previous cases, blocking Gibbs converges faster than single-site Gibbs, but here, it is possible that better results can be obtained with single-site Gibbs for runs shorter than 10 hours.

Here, $\alpha \approx -0.62$ for blocking Gibbs, and $\alpha \approx -0.073$ for single-site Gibbs, again indicating that blocking Gibbs mixes much faster than single-site Gibbs.

16.3.4 Summary

In general, the above results show that blocking Gibbs converges faster than single-site Gibbs for large, complex pedigrees. But better results can sometimes be obtained by single-site Gibbs when an upper bound on the available time is imposed.

All the above comparisons were based on suboptimal parameters for blocking Gibbs. Thus, the superiority of blocking Gibbs over single-site Gibbs can be expected to be even more pronounced (cf. Section 16.5). Optimization of the blocking Gibbs parameters is the issue of the following section.

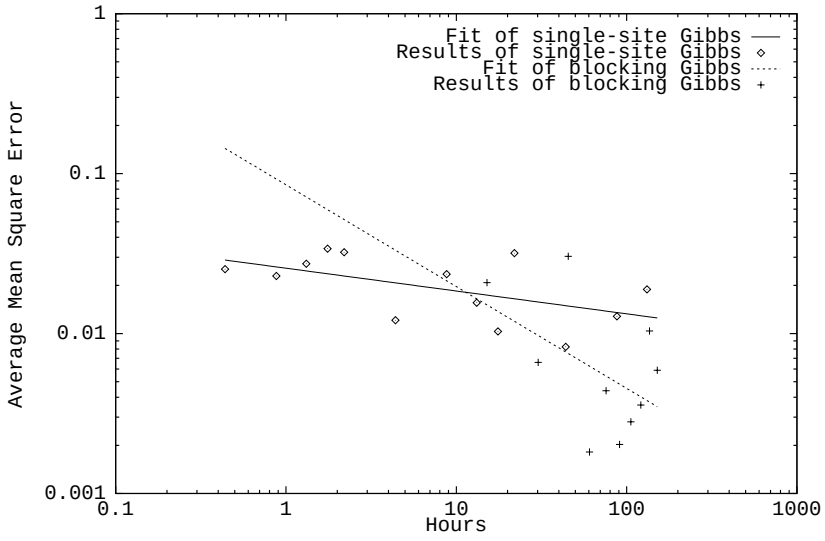


Figure 16.4: Blocking Gibbs vs. single-site Gibbs for Pedigree C.

16.4 Adjusting Parameter Values for Blocking Gibbs

We now conduct an empirical sensitivity analysis of the rate of convergence of blocking Gibbs wrt. choice of parameters values. As mentioned previously, the parameters are r (the size of A_1), k (the number of A-sets) and the method used for constructing the A-sets. The search for optimal parameter values is conducted by varying r, k, and the construction method (Method 1 or Method 2; see Sections 16.2.2–16.2.3). We choose a sample size of 1000 for each configuration of parameter values investigated. The comparison of the different configurations shall be based on the performance measure

$$\text{perf} = \lambda \cdot \log(\text{M.S.E.}) + \log(t).$$

This performance measure expresses the fact that two parameter configurations yielding, respectively,

1. M.S.E. $= 0.001$ in $t = 100$ seconds, and
2. M.S.E. $= 0.01$ in $t = 10$ seconds,

are usually not equally good. If the sampler with configuration 2 had been run for 100 seconds, we would not necessarily have obtained an M.S.E. of 0.001, but probably a higher value (lower precision). The coefficient λ is chosen such that two points on a line following Eq. (16.2) have identical performance measures.

We have performed the sensitivity analysis for Pedigree B only.

16.4.1 Size of A-sets

We present three figures showing respectively M.S.E. time and performance as a function of the size of A_1 (or A with method 2), r. In all three figures, results are presented for both Method 1 and Method 2. The results are as follows.

Average mean squared error. See Figure 16.5. The results for Method 1 are denoted "M.S.E. (1)", and likewise for Method 2. For both methods the M.S.E. seems to decrease as r decreases. This behaviour is anticipated, since in the limit, where the A-sets are empty, the samples become independent, and the larger the A-sets, the more blocking Gibbs resembles single-site Gibbs (i.e., the samples become "maximally dependent").

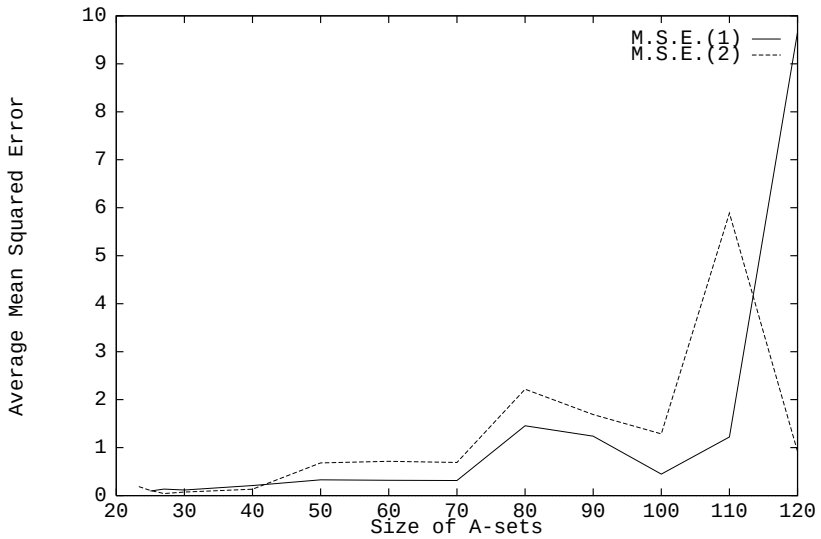


Figure 16.5: Precision of blocking Gibbs as a function of the size of A-sets using Method 1 and Method 2.

Iteration time. See Figure 16.6. As expected, the iteration time increases enormously as r decreases, rendering the A-sets small. That is, the size of the blocks increases, resulting in large cliques in the junction tree which slows down the computations.

Performance. See Figure 16.7. For both methods, optimal performance is obtained for r ranging from 27 to 40.

Notice the two conflicting tendencies. As r increases, the iteration time decreases, but the M.S.E. increases. To find the optimal configuration of parameters, some compromise must be established. A rule of thumb may be to choose the size of the A-sets as small as possible while not increasing the iteration time significantly.

16.4.2 Number of Blocks

Again, three figures show M.S.E., iteration time and performance as functions of the number of blocks, k. The results are as follows.

Average mean squared error. See Figure 16.8. The results reveal no obvious pattern, though it is clear that Method 2 is superior in all cases.

Iteration time. See Figure 16.9. It is clear that the iteration time increases when k is less than 4 and greater than 6. The optimal k-value seems to be 4, 5 or 6.

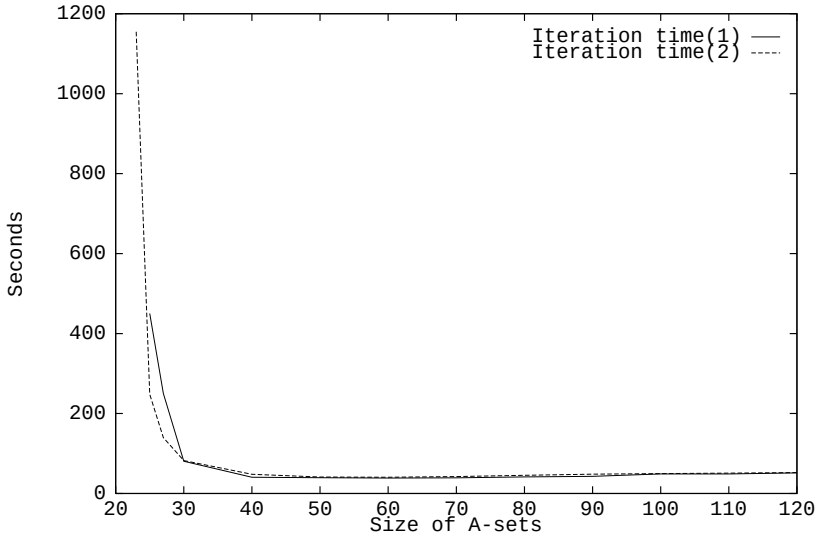


Figure 16.6: Iteration time of blocking Gibbs as a function of the size of A-sets using Method 1 and Method 2.

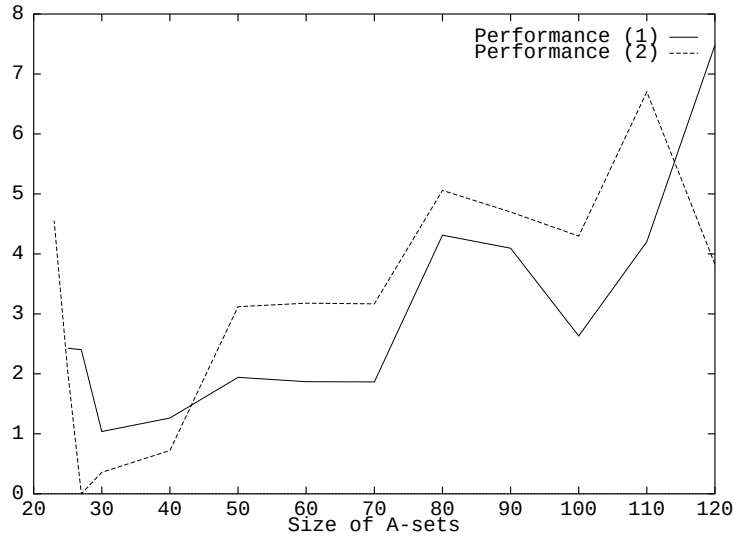


Figure 16.7: Performance of blocking Gibbs as a function of the size of A-sets using Method 1 and Method 2.

Performance. See Figure 16.10. Again, no obvious pattern can be observed, except for the fact that Method 2 is superior.

The results are not as clear as for the size of the A-sets. It seems that the M.S.E. does not depend on the number of blocks. In this case the best choice may be the number of blocks that yields the smallest iteration time. However, further

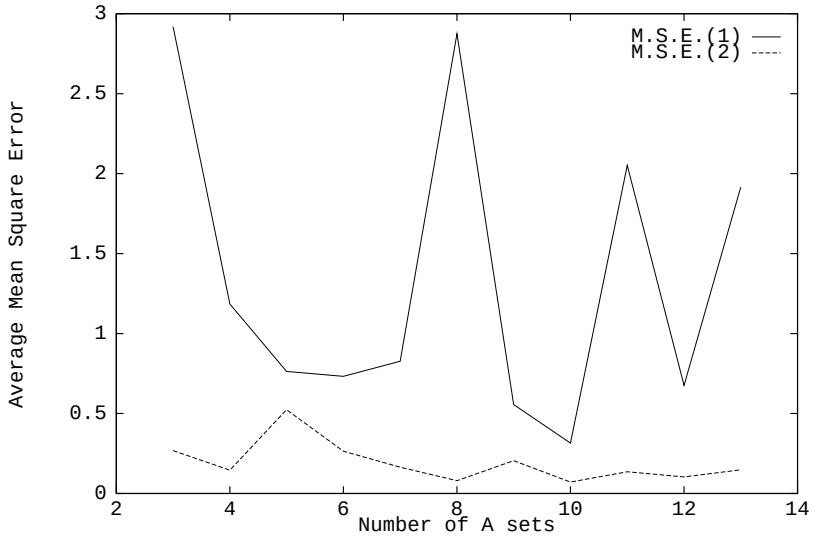


Figure 16.8: Precision of blocking Gibbs as a function of the number of blocks using Method 1 and Method 2.

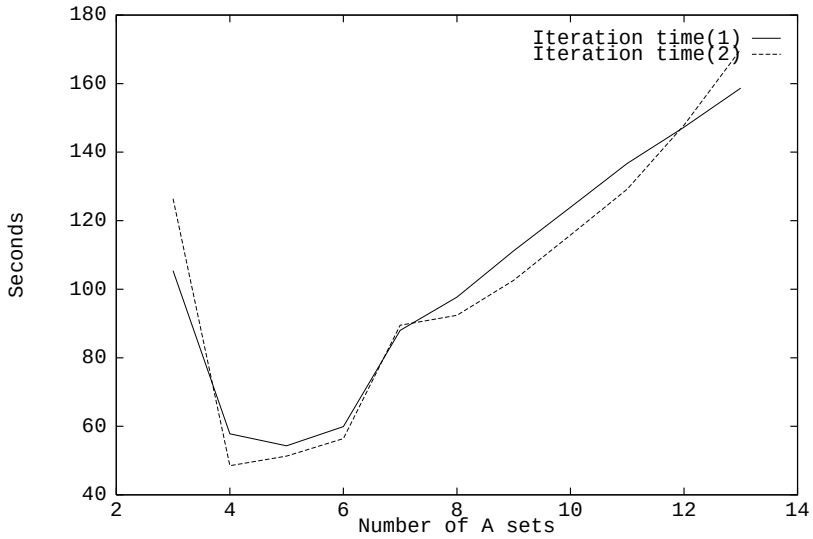


Figure 16.9: Iteration time of blocking Gibbs as a function of the number of A-sets using Method 1 and Method 2.

investigations should be conducted.

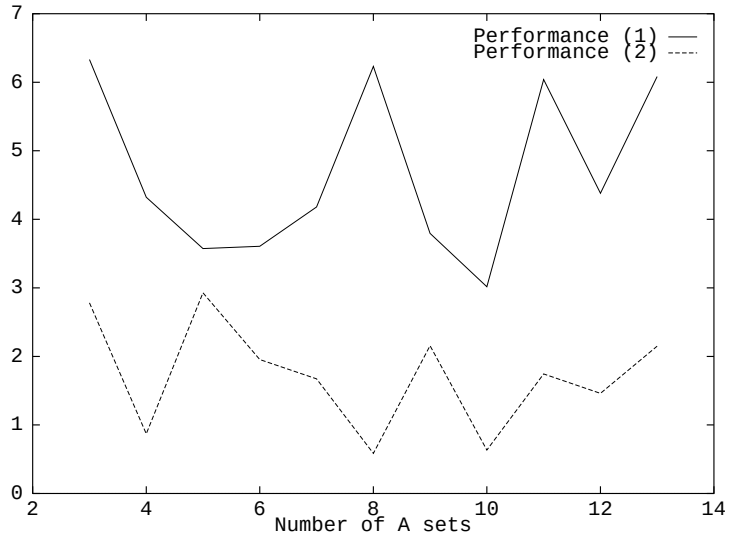


Figure 16.10: Performance of blocking Gibbs as a function of the number of A-sets using Method 1 and Method 2.

16.4.3 Construction of A-sets

The performance of the two construction methods may be evaluated through further analysis of previous results (relative to the size and number of A-sets). The results are as follows.

Size of A-sets. See Figures 16.5, 16.6 and 16.7. Method 2 has the best overall performance for sizes in the range 27–40.

Number of A-sets. See Figures 16.8, 16.9 and 16.10. Method 2 is best in all cases.

The result is obvious: Method 2 should always be used.

16.4.4 Summary

From the results in the previous sections we can list a few rules of thumb for selecting optimal parameter values for blocking Gibbs.

1. The size of A-sets should be as small as possible without increasing the iteration time significantly.
2. The number of blocks should be selected such that the iteration time becomes minimal.
3. Method 2 should be used for construction of A-sets if we have only Methods 1 and 2 to choose from. It is almost certain that the more recent block selection methods described in Chapter 9 are better, however.

Obviously, these guidelines are not as clear as we might wish, especially Rule 2 could be clarified by further investigations. The investigations were performed without inclusion of any evidence. However, the presence of evidence does not affect

the results obtained here. Although, the above results were derived from a heavily inbred pedigree of breeding pigs, it seems likely that they will also apply to other areas of interest, for example, human pedigrees which differ from pig pedigree in that they are not as inbred and usually contains far more observed variables.

16.5 Impact of Parameter Adjustment

The impact of employing optimal parameters for blocking Gibbs will now be investigated for Pedigree B. The results of applying the following suboptimal and optimal parameter values are compared. The optimal parameters are chosen according to the previously listed rules of thumb.

Suboptimal. Five blocks, 100 variables in the initial A-set, construction method 1 (used in Section 16.3).

Optimal. Six blocks, 30 variables in the initial A-set, Method 2.

The result of this comparison can be seen in Figure 16.11 which displays the average mean squared error (M.S.E.) as a function of length of run (measured in time) of blocking Gibbs with suboptimal and optimal parameters.

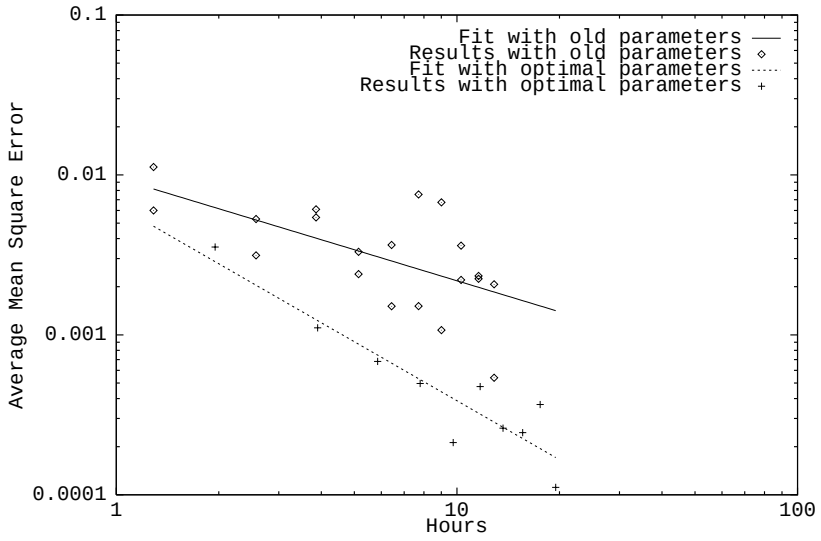


Figure 16.11: Blocking Gibbs with suboptimal parameters vs. blocking Gibbs with optimal parameters, Pedigree B.

It appears from Figure 16.11 that much faster convergence is obtained with optimal parameter values. Thus, it seems likely that the choice of parameter values has a great impact on the rate of convergence of blocking Gibbs.

As earlier, α can be read from the fitted lines according to Eq. (16.2). For the optimal blocking Gibbs sampler, $\alpha \approx -1.1$, and for the suboptimal sampler, $\alpha \approx -0.53$, i.e., with optimal parameters the blocking Gibbs sampler can mix much faster than with suboptimal parameters. Thus, it is of great importance to use optimal parameters when the performance of the sampler can be affected this much.

16.6 Discussion

Using an application in genetic pedigree analysis, we have in this chapter demonstrated how exact joint updating of many variables can be combined with Gibbs sampling to effectively handle large and complex networks. The strengths of the two methods complement each other. While a rather detailed case study has been performed, some general questions about blocking Gibbs have to be addressed. For example, potential users need to understand under what type of situations will blocking Gibbs perform better than single-site Gibbs after adjusting for the extra time needed to perform one iteration. Also, how to choose the blocks and how to effectively utilize the generated samples are important practical issues.

In general, single-site Gibbs tends to perform very well if the network is not too large and the unobserved variables are not too highly dependent on each other. The empirical results in previous sections show that the performance of single-site Gibbs gets worse as the size of the pedigree increases. The reason for this should probably be found in the presence of families with large numbers of offspring. This reduces the mixing of single-site Gibbs extremely much. This problem is further discussed in Section 15. Even though this problem demonstrates how quickly the performance of single-site Gibbs can deteriorate with only two alleles, it is at least true that the correct answer can be obtained if enough iterations are performed, see (Sheehan & Thomas 1993). This is not necessarily the case with three or more alleles, see Chapter 18, where examples of this are given.

In the previous sections, it was shown how much faster blocking Gibbs mixes than single-site Gibbs, enabling it to converge at a much higher rate. Also, it was shown that using optimal parameters for the blocking Gibbs sampler is very significant for the performance of the sampler. In the example examined here, the mixing rate was doubled.

In general, single-site Gibbs will fail entirely if the induced Markov chain is reducible. As demonstrated in Chapter 18, this can happen for very simple family structures in pedigree analysis, and certainly also in other applications. The problem can be easily solved with blocking Gibbs by updating the correlated variables jointly. However, finding an algorithm for choosing blocks that guarantee irreducibility is a challenging problem, further described in Section 9.4 and Chapter 18.

Literature on the theoretical properties of Gibbs sampling has grown quickly in the last few years, but most of the results either do not apply to blocking Gibbs, or do not address the practical problems. For example, some of the only theoretical work, as far as we know, that studies the effect of blocking is Liu et al. (1994) and it only considers the case where the blocks do not overlap. In situations where the blocks overlap, some of the variables are present in more than one block and thus get sampled multiple times in each iteration.

16.7 Testing for Irreducibility

This section outlines a simple method for testing whether irreducibility holds in a general Gibbs sampling scheme applied to pedigree analysis. The idea is simple and based on the proof of Sheehan & Thomas (1993) that if the single-site Gibbs sampler is applied to a pedigree analysis with a diallelic trait with penetrance probabilities P, then all consistent genotype configurations communicate, if

$$P(p|AA) > 0 \text{ and } P(p|BB) > 0 \Rightarrow P(p|AB) > 0 \text{ for all phenotypes } p \tag{16.8}$$

The condition in (16.8) is almost always fulfilled, so single-site Gibbs samplers applied to diallelic pedigree analysis are almost always irreducible.

If a pedigree analysis with $k > 2$ alleles should be carried out and it is unknown whether the configurations communicate, the trait can be collapsed to a diallelic trait, by representing the $k - 1$ least probable alleles by a single allele. The most probable allele is kept. Thus, when applying the Gibbs sampler to the collapsed trait, results can be obtained under guaranteed irreducibility. These results can be compared with the results obtained when running with all k alleles and it is indicated whether the k-allele Gibbs sampler is irreducible.

Chapter 17

Linkage Analysis on a Human Pedigree

For linkage analysis - the problem of estimating the relative positions of the genes on the chromosomes - many methods have been developed over recent years. Fast and exact methods for computation in Bayesian networks (e.g., pedigrees) (Cannings et al. 1976, Pearl 1986b, Lauritzen & Spiegelhalter 1988, Shenoy & Shafer 1990, Lauritzen 1992) handle only small problems, as the computation is NP-hard. Markov chain Monte Carlo (MCMC) methods (Gelfand & Smith 1990, Thomas et al. 1992, Gelman & Rubin 1992, Geyer 1992, Smith & Roberts 1993) have provided a good alternative as they are able to handle problems of very large size. Using these methods, computation time often exceeds any acceptable level when considering very large networks (e.g., pedigrees of thousands of individuals), and it is often difficult to decide whether the desired precision has been reached.

Linkage analysis represents a problem of high complexity that has been particularly hard to handle. Existing methods such as those implemented in the `LINKAGE` (Lathrop & Lalouel 1984, Lathrop et al. 1985) and `FASTLINK` software packages (Cottingham Jr. et al. 1993, Schäffer et al. 1994) are unable to handle even pedigrees with a moderately low number of loops (≈ 10), the computation time being exponential in the number of loops. Sequential imputation (Kong et al. 1993, Irwin et al. 1994) which is essentially also a blocking scheme handles multiple loci very well, but only with zero or very few loops. Simulated tempering/annealing MCMC (Geyer & Thompson 1995) is a promising approach that handles these cases, although it seems to require a difficult choice of initial parameters and may suffer from problems with low acceptance rates.

Blocking Gibbs that have been described in Part I of this thesis allows general inference in very large complex Bayesian networks and is a particularly promising method for linkage analysis as well as many other problems requiring inference in large Bayesian networks. The method combines exact local computations and Gibbs sampling (Geman & Geman 1984), such that instead of sampling a single variable at a time (*single-site Gibbs*), a very large part of the variables (usually more than 90%) is sampled jointly using exact local computations. Joint sampling of many variables (i.e., block updating) hinges on the fact that conditioning on certain variables breaks loops in the network, creating a network with fewer loops. This network, given that enough loops are broken, then becomes feasible for exact computation, allowing us to sample the variables of the network jointly. As blocking Gibbs operates on general Bayesian networks where the general unit of information is a variable, this notation will be kept throughout the chapter. A variable can

thus represent any chosen unit of information such as a genotype, a single gene, a phenotype, etc.

Jensen et al. (1995) compared blocking Gibbs with single-site Gibbs sampling, see Chapter 16. They applied both methods to three pedigrees consisting of 455, 704 and 1894 individuals. It was shown that in all cases considered blocking Gibbs performs better than single-site Gibbs. In general, blocking Gibbs mixes fast while single-site Gibbs often mixes very slowly or even gets completely stuck. The example pedigrees used in (Jensen et al. 1995) were small (blocking Gibbs can handle much larger pedigrees) but highly inbred with hundreds if not thousands of loops. No other known method can handle pedigrees this large - except of simulated tempering, given that suitable starting parameters can be found.

In this chapter we will apply blocking Gibbs to a 73 individual linkage problem (see Figure 8.3 on page 52) (Kong 1991) concerning a rare heart disease called the long QT syndrome (LQT). The LQT pedigree originating from Professor Brian Suarez is affected by the long QT syndrome. Blood samples have been collected from individuals 9, 13 and 17 through 59. Thus marker data are available for only these individuals. The marker data in Figure 8.1 are simulated by Professor Suarez to mimic close linkage (recombination fraction, θ close to 0). The marker is assumed to have 4 alleles with equal population frequencies. The LQT syndrome is assumed to be determined by 2 alleles with population frequencies 0.05 and 0.95, the disease allele being the rarest.

We will perform a two-point linkage analysis on this pedigree, and analyze the behaviour and performance of blocking Gibbs. This example contains many loops, however, it is still possible to use exact computation on it, providing us with the correct result. Though exact computation is possible, this example is just within the limits of what is currently possible to do with exact methods. Still, the blocking Gibbs method is able to handle much larger examples.

In the next section, it will be explained how the linkage analysis problem is represented with a Bayesian network, then the blocking Gibbs method will be applied to the LQT pedigree, and a discussion follows.

17.1 Linkage Analysis Representation

In the present implementation of blocking Gibbs, the representation described by Kong (1991) was used. The representation was outlined in Section 13.3.

In this representation (*the gene representation*) each variable represents a single gene instead of a genotype, such as described in Section 13.2 and shown in Figure 17.1.

When this representation is used, reducibility can occur in a number of special cases, as described in Chapter 14. When more than one locus is considered, an extended version of the gene representation is used, denoted the *linkage representation*. This representation is described fully in Section 13.3 with definitions of the variables representing the paternal and maternal genes at each of the loci, a, marker, and d, disease. The network representing the two locus situation is shown in Figure 13.6 on page 93. The joint distribution of the indicator variables indicating presence or absence of recombination, $Z^A_{C,a}$ and $Z^A_{C,d}$, is shown at the same place in Eq. (13.1).

The joint distribution of the variables, A^f_a, A^m_a, A^f_d, A^m_d, $Z^{f(A)}_{A,a}$, $Z^{f(A)}_{A,d}$, $Z^{m(A)}_{A,a}$, $Z^{m(A)}_{A,d} \in W$ (set of variables in linkage representation of the pedigree) for all individuals $A \in V$ (set of individuals) can be written as (where $V' \subset V$ is the set of

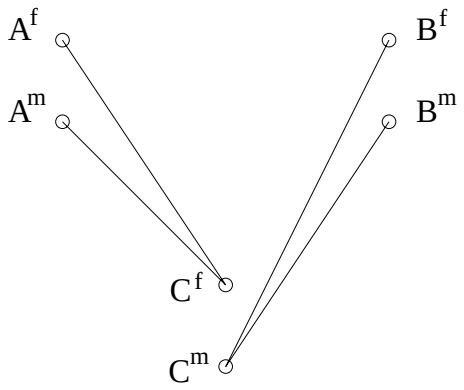


Figure 17.1: A and B have offspring C. The notation is understood as follows. A has genes A^f and A^m. A^f is the gene originating from the father of A (i.e., the paternal gene), and A^m is the one originating from the mother of A (i.e., the maternal gene).

individuals with no parents, and $f(A)$ and $m(A)$ denotes the parents of A) :

$$\begin{aligned} P_\theta(W) \quad = \quad & \prod_{A\in V'} P(A_a^f)P(A_a^m)P(A_d^f)P(A_d^m) \prod_{A\in V-V'} P_\theta(Z_{A,a}^{f(A)}, Z_{A,d}^{f(A)})P_\theta(Z_{A,a}^{m(A)}, Z_{A,d}^{m(A)}) \\ & \prod_{A\in V-V'} P(A_a^f|f(A)_a^f, f(A)_a^m, Z_{A,a}^{f(A)})P(A_a^m|m(A)_a^f, m(A)_a^m, Z_{A,a}^{m(A)}) \\ & \prod_{A\in V-V'} P(A_d^f|f(A)_d^f, f(A)_d^m, Z_{A,d}^{f(A)})P(A_d^m|m(A)_d^f, m(A)_d^m, Z_{A,d}^{m(A)}), \end{aligned} \tag{17.1}$$

i.e., P admits recursive factorization according to the DAG of the Bayesian network representation of the variables in W.

Kong (1991) discussed the merits of representation (17.1) exemplified in Figure 13.6. This representation results in more variables but the conditional probability tables of Eq. (17.1) are simpler and require less storage space. The representation is easily represented with a Bayesian network and can be handled immediately by blocking Gibbs. Each of the components in (17.1) specifies a variable in a Bayesian network. The conditional distributions furthermore specify edges from parents (the conditioning variables) to a child. The pedigree in Figure 13.6 thus shows an example of the Bayesian network representation of Eq. (17.1). The entire LQT pedigree is thus represented with a Bayesian network using the representation shown in Figure 13.6.

17.2 Linkage Analysis with Blocking Gibbs

To perform linkage analysis with blocking Gibbs, we first pick a suitable value for θ (θ_0) such that all samples are produced conditional on this recombination fraction. Then a starting configuration is found, and blocking Gibbs can start. At each iteration, the number of recombinations (n_r) and non-recombinations (n_{nr}) are counted. A recombination has occurred if a pair of associated indicator-variables (e.g., $Z_{C,a}^A$ and $Z_{C,d}^A$) have different values, and a non-recombination if they have the same value. This simple counting scheme will be referred to as Method 1 in the following.

However, this scheme can be refined. If, in Figure 13.6, we examine the pair of indicator variables $Z^A_{C,a}$ and $Z^A_{C,d}$ and A^f_a is identical to A^m_a, we do not know which of these genes has been inherited by A. This means that we do not know whether a recombination has occurred or not. Counting this case as either a recombination or a non-recombination corresponds to adding noise to the estimate. Therefore, leaving it out leads to a better estimate. In general, cases where the parent is homozygous at one of the loci should be left out. This refined counting scheme will be referred to as Method 2 in the following. Currently, linkage analysis implementations using a MCMC method always use either Method 1 or 2.

However, Method 2 can be refined even further. Consider the LQT pedigree of Figure 8.3 on page 52. For individuals with no offspring it is relatively easy to directly estimate the probabilities of recombination in each iteration. Considering individual 40, there are two recombination fractions to consider for this individual: one relating to the inheritance from the father, and one to the inheritance from the mother. For example, given that we know the values of the father's genes at the disease locus $(18^f_d, 18^m_d)$ and the marker locus $(18^f_a, 18^m_a)$ we know the possible outcomes for the genes of 40, 40^f_a and 40^f_d. We can easily calculate the probability of each of the outcomes, and the probability of recombination and non-recombination for each outcome. The probabilities of recombination for the outcomes are summed, and the probabilities of non-recombination are summed. After a normalization we have calculated the probability of a recombination occurring at individual 40's inheritance from individual 18. This computation can be performed for all individuals without offspring. For individuals with offspring, the recombination fractions are not independent and the calculation cannot be performed easily. However, in Figure 8.3, 42% of the individuals have no offspring making the benefit of using this refined scheme large. This counting scheme will be referred to as Method 3 in the following.

Example of Method 3 : In Figure 17.2 an example configuration of individuals 18 and 20 is shown. In Table 17.1 the computation leading us to the probabilities of recombination is shown.

First, given the configuration of the parents, the possible configurations for the offspring (40) are found. We know that 40 has the genotype 24 at the marker locus, and as we know that allele 2 (at the marker locus a) must originate from the father 18 and allele 4 (also at the marker locus) must originate from the mother 20, and that individual 40 does not have the LQT disease, we have the three possible configurations shown in the second row of Table 17.1. In the second column of the second row, the format of the configurations is shown.

For each of these configurations we check whether we have a recombination at the inheritance from the father and/or at the inheritance from the mother. With configuration 1, we have a recombination when inheriting from the father, as the allele 1 (locus d) originates from the grandfather, and allele 2 (locus a) originates from the grandmother. With configuration 1, we have a non-recombination (i.e., no recombination) when inheriting from the mother, as both allele 2 (locus d) and allele 4 (locus a) originate from the grandfather.

Then, we compute the probability of each of these configurations occurring. For the first configuration this is found simply by multiplying the probability of one recombination and one non-recombination. The others are computed similarly.

Finally, the probability of recombination from the father (18) is calculated by summing together the probability contributions from the configurations

where we saw a recombination when inheriting from the father, and dividing this number by the total sum of the probability contributions.

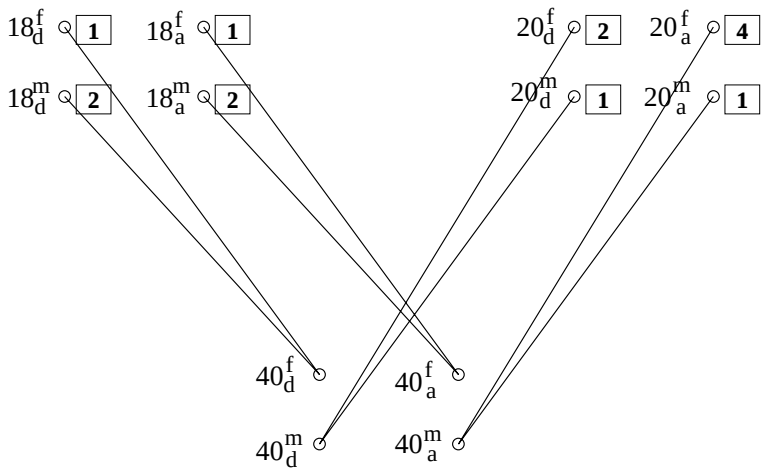


Figure 17.2: An example configuration of the individuals 18 and 20. 18 has the disease genotype 12, meaning that it is unaffected but carries the disease allele (1). Also, 18 has the marker genotype 12. 20 is unaffected as well, having the genotype 21, and it has the marker genotype 41.

		1	2	3
Configurations of 40	40^f_d 40^f_a 40^m_d 40^m_a	1 2 2 4	2 2 1 4	2 2 2 4
Recombinations	From 18 From 20	Rec. Non-rec.	Non-rec. Rec.	Non-rec. Non-rec.
Probability		$\frac{\theta_0}{2} \cdot \frac{1-\theta_0}{2}$	$\frac{\theta_0}{2} \cdot \frac{1-\theta_0}{2}$	$\left(\frac{1-\theta_0}{2}\right)^2$

Table 17.1: A table illustrating the calculation of probabilities of recombination in Figure 17.2. The second row shows the three possible configurations of 40 given the configurations of 18 and 20. The third row shows where there recombinations and where there are not. Fourth row shows the probability of each configuration.

In Table 17.2, the three methods have been compared on the LQT pedigree. For each method, 10 runs have been performed (at both 100 and 1,000 iterations). The same seed for random numbers was used for the ith run of each method to make sure that the results are not coincidental. The complexities of the three methods are almost identical as the more complex computations performed in Method 3 takes negligible extra time compared with the remaining computations used by the blocking Gibbs sampler in each iteration. The average and standard deviation of $\log_{10} \frac{L(\theta_1)}{L(\theta_0)}$ where $L(\theta)$ is the likelihood of θ are shown in Table 17.2. We put $\theta_0 = 0.2$ and $\theta_1 = 0.3$ in this case. This value will in the following be referred to as the *log-likelihood difference* as it is in effect the difference between the log-likelihoods at two values for θ. The lod score corresponds to these log-likelihood differences in the

100 iter.	S.D.(1)	Diff. : $\log_{10}\frac{L(0.2)}{L(0.3)}$ Average	S.D.(2)
Method 1	0.22	1.9	0.3
Method 2	0.032	1.93	0.07
Method 3	0.028	1.90	0.04
1000 iter.			
Method 1	0.023	1.88	0.05
Method 2	0.013	1.87	0.03
Method 3	0.0037	1.91	0.02
Exact	-	1.85	-

Table 17.2: The three counting methods are compared on the LQT pedigree. 20 runs are performed for each method, 10 with 100 iterations, and 10 with 1,000 iterations. The average over the 10 log-likelihood differences is shown in the *Average* column. *S.D.(1)* is the standard deviation of the Markov chain found using the autocorrelations method by Geyer (1991) and *S.D.(2)* is the standard deviation over the 10 results. Finally, the exact result is shown for comparison.

following way :

$$\mathrm{LOD}(\theta) = \log_{10}\frac{L(\widehat{\theta})}{L(0.5)},$$

where $\widehat{\theta}$ is the recombination fraction with the maximum likelihood. The relationship between the log-likelihood difference in Table 17.2 and the lod score will be elaborated further later in the next section. For now, it suffices to state that the log-likelihood differences shown in Table 17.2 are used in the computation of the lod score, and as they become more precise, the lod score becomes more precise.

The standard deviation shown in the second column of Table 17.2 (S.D.(1)) is computed by the autocorrelations method described by Geyer (1991). This standard deviation expresses the variation between the dependent Markov chain samples. As the runs on the three methods are based on the same seed for random numbers, it is clearly seen that the standard deviation of the estimate can be lowered significantly by using the more advanced methods. Using Method 2 instead of Method 1 basically corresponds to removing noise from the Method 1 estimate, and thus obtaining a smaller standard deviation. Further noise is removed when Method 3 is applied. As Method 3 is clearly the optimal of the three methods, this method has been used in all subsequent runs.

Even though the results of Table 17.2 shows that Method 3 is farther from the exact results than the other two methods, it is believed that after sufficient iterations, all three methods will converge to a result close to 1.91, as with the estimation method used it is impossible to get the exact result. This will be further elaborated in the next section.

17.3 Estimation of Recombination Fraction

The recombination fraction is estimated by using the number of recombinations n_r, the number of non-recombinations n_{nr}, and the calculated recombination fraction of bottom level individuals $r_1, \ldots, r_k$ by the means of Eq. (17.2). If M iterations have been performed with the blocking Gibbs sampler at a fixed recombination fraction

θ_0, the likelihood ratio can be estimated with the following expression ($n_r^{(k)}$ is the number of calculated recombination probabilities of bottom level individuals at iteration k):

$$\frac{L(\theta_1)}{L(\theta_0)} = \frac{1}{M}\sum_{k=1}^{M}\left(\frac{\theta_1}{\theta_0}\right)^{n_r^{(k)}}\left(\frac{1-\theta_1}{1-\theta_0}\right)^{n_{nr}^{(k)}}\left[\prod_{i=1}^{n_r^{(k)}}\left(\frac{\theta_1}{\theta_0}r_i + \frac{1-\theta_1}{1-\theta_0}(1-r_i)\right)\right] \tag{17.2}$$

The lod score can be found by setting $\theta_0 = 0.5$, maximizing the ratio (17.2) over θ_1 and applying $\log_{10}$.

Using methods presented by Meng & Wong (1996), it is now possible to combine the results from two runs with different recombination fractions of some sampling scheme. We hereby present the first results of using these methods in a practical application. The theory underlying the methods is described very well by Meng & Wong (1996) and will not be covered in detail here. We will only present the formulas that have been used in this paper, and the notation has been changed to better suit this application. Imagine, we have run 100 iterations at $\theta_0 = 0.1$, and 100 iterations at $\theta_1 = 0.3$. Now, if we want to compute $\frac{L(0.3)}{L(0.1)}$, instead of using Eq. (17.2) and use only the run at 0.1, we can use both runs. As more information is included in the computation, it is expected that by using this method we will get better results. Meng & Wong (1996) discuss various methods for performing this estimation. We will compare two of these methods with Eq. (17.2).

Eq. (17.2) can be expressed as (where w_i is observations found given θ_i, and E_0 implies that we average over observations found given $\theta = \theta_0$):

$$\frac{L(\theta_1)}{L(\theta_0)} = E_0\left[\frac{q_1(w_0|\theta_1)}{q_0(w_0|\theta_0)}\right] \tag{17.3}$$

q_0 and q_1 are the two expressions for computing the likelihood of the data given respectively θ_0 and θ_1. Considering Eq. (17.2), we see :

$$\frac{q_1(w_0|\theta_1)}{q_0(w_0|\theta_0)} = \left(\frac{\theta_1}{\theta_0}\right)^{n_r^{(k)}}\left(\frac{1-\theta_1}{1-\theta_0}\right)^{n_{nr}^{(k)}}\left[\prod_{i=1}^{n_r^{(k)}}\left(\frac{\theta_1}{\theta_0}r_i + \frac{1-\theta_1}{1-\theta_0}(1-r_i)\right)\right], \tag{17.4}$$

where w_0 simply implies that the data used in the right hand side of the expression, i.e., $n_r^{(k)}$, $n_{nr}^{(k)}$ and $r_1, \ldots, r_{n_r^{(k)}}$, are produced at the recombination fraction θ_0. If w_1 had been used, data produced at θ_1 would have been used.

Eq. (17.2) is actually a generalization of an equation, that states :

$$\frac{L(\theta_1)}{L(\theta_0)} = \frac{E_0\left[q_1(w_0|\theta_1)\alpha(w_0)\right]}{E_1\left[q_0(w_1|\theta_0)\alpha(w_1)\right]} \tag{17.5}$$

where $\alpha(w)$ is an arbitrary function. Different choices of $\alpha(w)$ is discussed by Meng & Wong (1996). If $\alpha(w) = \frac{1}{q_0(w|\theta_0)}$, Eq. (17.5) reduces to Eq. (17.3).

As mentioned, we will look at two choices of $\alpha(w)$ that combines the results obtained at two recombination fractions :

1. $\alpha = \frac{1}{\sqrt{q_0 q_1}}$. Using this α, Eq. (17.5) looks as Eq. (17.6). This method will be referred to as the square-root method in the following (Meng & Wong

(1996) denotes it the *geometric-mean method*). It can be easily computed by substituting Eq. (17.4) with the correct parameters in each of the square-roots.

$$\frac{L(\theta_1)}{L(\theta_0)} = \frac{E_0\left(\sqrt{\frac{q_1(w_0|\theta_1)}{q_0(w_0|\theta_0)}}\right)}{E_1\left(\sqrt{\frac{q_0(w_1|\theta_0)}{q_1(w_1|\theta_1)}}\right)} \tag{17.6}$$

2. $\alpha = \frac{c}{s_1 q_1 + s_0 r q_0}$, where c is a constant and if n_0 is the number of observations with θ_0, and n_1 is the number of observations with θ_1, and $n = n_0 + n_1$, then $s_0 = \frac{n_0}{n}$ and $s_1 = \frac{n_1}{n}$. α furthermore depends on the ratio r, which is computed in an iterative fashion. In Eq. (17.7), the iterative estimator is shown.

$$\frac{L(\theta_1)}{L(\theta_0)} = \frac{E_0\left[\frac{q_1(w_0|\theta_1)}{s_1 q_1(w_0|\theta_1)+s_0 r q_0(w_0|\theta_0)}\right]}{E_1\left[\frac{q_0(w_1|\theta_0)}{s_1 q_1(w_1|\theta_1)+s_0 r q_0(w_1|\theta_0)}\right]} \tag{17.7}$$

Starting with an initial guess of r, $\widehat{r}^{(0)}$ found, e.g., by using the square-root method, we calculate the estimate of r iteratively by using the previous estimate of r. Specifically, at the $(t+1)$st iteration, we compute :

$$\begin{aligned}
\widehat{r}^{(t+1)} &= \frac{\frac{1}{n_0}\sum_{i=1}^{n_0}\left[\frac{q_1(w_{0i}|\theta_1)}{s_1 q_1(w_{0i}|\theta_1)+s_0\widehat{r}^{(t)} q_0(w_{0i}|\theta_0)}\right]}{\frac{1}{n_1}\sum_{i=1}^{n_1}\left[\frac{q_0(w_{1i}|\theta_0)}{s_1 q_1(w_{1i}|\theta_1)+s_0\widehat{r}^{(t)} q_0(w_{1i}|\theta_0)}\right]} \\
&= \frac{\frac{1}{n_0}\sum_{i=1}^{n_0}\left[\frac{l_{0i}}{s_1 l_{0i}+s_0\widehat{r}^{(t)}}\right]}{\frac{1}{n_1}\sum_{i=1}^{n_1}\left[\frac{1}{s_1 l_{1i}+s_0\widehat{r}^{(t)}}\right]},
\end{aligned} \tag{17.8}$$

where w_{ij} is data produced at θ_i in the jth iteration, $l_{0i} = \frac{q_1(w_{0i}|\theta_1)}{q_0(w_{0i}|\theta_0)}$ and $l_{1i} = \frac{q_1(w_{1i}|\theta_1)}{q_0(w_{1i}|\theta_0)}$. These values need only be calculated once at the beginning of the algorithm using Eq. (17.4).

There is one problem with this method. When the samples are independent, we know the exact sample sizes, n_0 and n_1. However, with dependent samples, such as with the blocking Gibbs sampler, n_0 and n_1 are no longer the true sample sizes, since the dependence between successive samples typically reduces the "effective sample sizes" and thus using n_0 and n_1 may lead to simulation errors.

The square-root method is a new method by Meng & Wong (1996) but the iterative method is not new. It has been discussed by Bennett (1976) in the area of physics. The square-root method is an interesting addition, as it is sometimes desirable to have simple, non-iterative procedures that have good, not necessarily optimal, properties. Such a non-iterative estimator, for example, can be used as a starting value of the iterative method. As we will see in Table 17.3, a non-iterative estimator can be better than the iterative when the samples are not independent.

The potential of the simple identity of Eq. (17.5) has been further investigated by Gelman & Meng (1994), Gelman & Meng (1996), and Meng & Schilling (1996).

In Table 17.3, results from runs with the three previously described methods are shown. 10 new runs have been performed for each of the methods for 100 and

Iterations	Method	Diff. : $\log_{10}\frac{L(0.3)}{L(0.4)}$	
		Average	S.D.
100	simple_1	1.66	0.07
	simple_2	1.50	0.05
	sqrt	1.58	0.04
	iterative	1.59	0.04
1000	simple_1	1.63	0.07
	simple_2	1.53	0.02
	sqrt	1.59	0.01
	iterative	1.60	0.01
10,000	simple_1	1.64	0.01
	simple_2	1.534	0.005
	sqrt	1.591	0.005
	iterative	1.599	0.005
-	exact	1.592	-

Table 17.3: Comparison of the three methods for estimating θ. The four rows are - $simple_1$: log-likelihood difference for 0.3 over 0.4 using results at $\theta_0 = 0.3$, $simple_2$: difference for 0.3 over 0.4 using results at $\theta_0 = 0.4$, *sqrt* : difference for 0.3 over 0.4 combining results at $\theta_1 = 0.3$ and $\theta_0 = 0.4$ using the square-root method, *iterative* : difference for 0.3 over 0.4 combining results at $\theta_1 = 0.3$ and $\theta_0 = 0.4$ using the iterative method.

1,000 iterations. It is seen that the log-likelihood differences found by combining the results from two runs with different θ_0 are consistently better than those obtained by using only the results of a single run.

It is also interesting to note that the non-iterative square-root method seems to be significantly closer to the exact value than the iterative method. This is probably due to the fact that the optimality of the iterative method was derived under the independence assumption, but the samples of the blocking Gibbs sampler are dependent. In such cases, Meng & Wong (1996) speculated that the square-root method can be better. This is a useful result, showing that the simple non-iterative estimator can be better than the more complex iterative method in cases where samples are dependent. Furthermore, with the blocking Gibbs sampler, the samples are much less dependent than with the single-site Gibbs sampler, indicating that the square-root method may be significantly better than the iterative method in this case.

17.4 Results

In this section, blocking Gibbs will be applied to the LQT pedigree, and the results are presented in Table 17.4. In this table, the log-likelihood differences for a number of pairs of recombination fractions have been found. Each element in the table represents the log-likelihood difference for a pair of recombination fractions, e.g., $\log_{10}\frac{L(0.2)}{L(0.1)}$. A good estimate for the lod score, i.e., $\log_{10}\frac{L(\widehat{\theta})}{L(0.5)}$ (with $\widehat{\theta} = 0.0$ as the LQT pedigree data is simulated to close linkage) can be found by adding together the differences. Each of the differences of Table 17.4, $\log_{10}\frac{L(0.0)}{L(0.01)}$, $\log_{10}\frac{L(0.01)}{L(0.1)}$, $\log_{10}\frac{L(0.1)}{L(0.2)}$, $\log_{10}\frac{L(0.2)}{L(0.3)}$, $\log_{10}\frac{L(0.3)}{L(0.4)}$ and $\log_{10}\frac{L(0.4)}{L(0.5)}$ corresponds to a piece of the

graph in Figure 17.3. The maximum likelihood estimate of θ can be found in Figure 17.3 by finding the highest point on the graph, and the lod score can be read as the log-likelihood difference of this point. Furthermore, the exact results have been plotted in Figure 17.3, and can be compared with the estimates. Clearly, the estimates converge towards the exact results as the graph of the exact results and the graph at 10,000 iterations cannot be visibly distinguished. At each log-likelihood difference 10 runs[1] have been made and the values occurring in the table are the mean and the standard deviation of these runs.

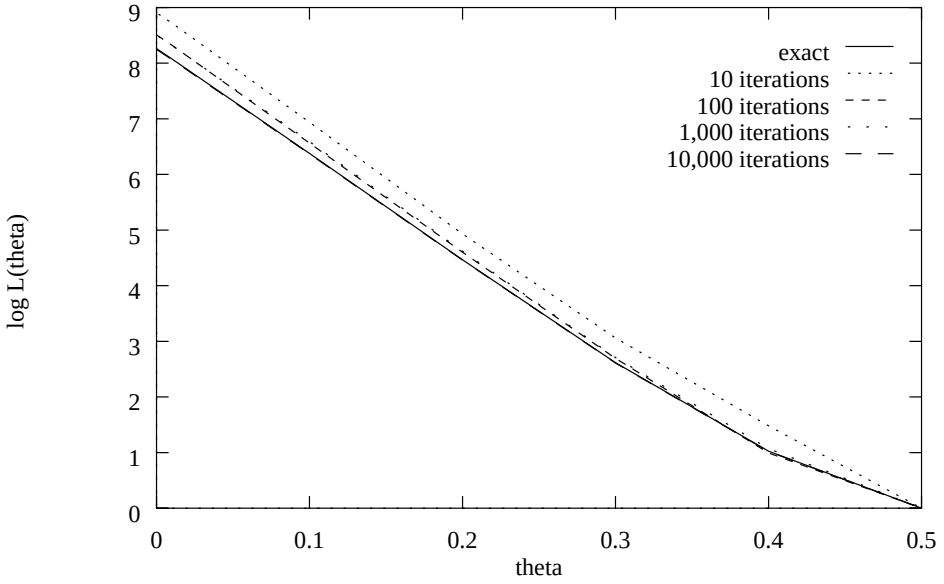


Figure 17.3: The log likelihood of θ ($\log_{10} L(\theta)$) has been plotted against θ assuming that $\log_{10} L(0.5) = 0$. For 10, 100, and 1,000 iterations, the maximum likelihood estimate for θ is clearly 0.0, and it can be seen that the graphs converge towards the exact curve.

From Table 17.4 it can be seen that the most likely recombination fraction is indeed 0. Adding the log-likelihood differences together we find the lod score to be approximately 8 thus providing strong evidence of tight linkage.

Furthermore, it can be seen that the accuracy of the estimates improves significantly when more iterations are performed. After 10,000 iterations the standard deviation is approximately 1% of the estimate for almost all of the differences, showing that very high precision can be obtained.

However, it can also be seen from Table 17.4 that the estimates do not seem to converge towards the exact results. From 1,000 to 10,000 iterations the estimates have not moved further towards the exact results, indicating that they do not converge towards them. The reason for this is that as explained previously, the estimator used in Table 17.4, i.e., called $simple_1$ in Table 17.3, is not optimal. Thus, even if an infinite number of iterations are performed, we will not be able to obtain better results with this method. However, better results can be obtained in several ways, see Table 17.5. Here, we compare the results for the $simple_1$ and $simple_2$ methods with the *square-root* and *iterative* estimation methods, and a new method denoted $simple_3$. The simple$_3$ method combines the results of the runs of

[1] On a SPARCstation-20, a 1000 iteration run took approximately 9 hours.

θ_1 θ_0	0.0 0.01	0.1 0.01	0.2 0.1	0.3 0.2	0.4 0.3	0.5 0.4	lod score
Diff.	$\log \frac{L(0.0)}{L(0.01)}$	$\log \frac{L(0.01)}{L(0.1)}$	$\log \frac{L(0.1)}{L(0.2)}$	$\log \frac{L(0.2)}{L(0.3)}$	$\log \frac{L(0.3)}{L(0.4)}$	$\log \frac{L(0.4)}{L(0.5)}$	$\log \frac{L(0.0)}{L(0.5)}$
Iter.							
10	0.18 (0.01)	1.6 (0.2)	1.92 (0.07)	1.8 (0.2)	1.5 (0.3)	0.9 (0.5)	8 (1)
100	0.184 (0.004)	1.72 (0.04)	1.97 (0.03)	1.90 (0.04)	1.66 (0.07)	1.0 (0.2)	8.4 (0.4)
1,000	0.186 (0.001)	1.73 (0.02)	1.966 (0.008)	1.91 (0.02)	1.63 (0.07)	1.05 (0.04)	8.5 (0.2)
10,000	0.1856 (0.0003)	1.73 (0.01)	1.966 (0.006)	1.912 (0.008)	1.64 (0.01)	1.02 0.02)	8.45 (0.05)
Exact	0.1859	1.70	1.912	1.851	1.592	1.02	8.26

Table 17.4: The results of the blocking Gibbs sampler applied to the LQT linkage problem. Each of the numbers specify the average of ten log-likelihood differences, with either 10, 100, or 1,000 iterations obtained using the simple_1 estimator. θ_0 and θ_1 are specified, θ_0 being the recombination fraction used during the run. The *Diff.* row specifies which log-likelihood difference is shown in the column. The numbers in parentheses are the standard deviations over the 10 runs. The *Exact* row specifies the exact results. The lod score column is the sum of the previous six columns.

Table 17.4 in a more clever way. For example, instead of estimating $\log_{10} \frac{L(0.2)}{L(0.1)}$ only by using the results at $\theta_0 = 0.1$, we can use the results at $\theta_0 = 0.1$ to compute $\log_{10} \frac{L(0.1)}{L(0.15)}$ and the results at $\theta_0 = 0.2$ to compute $\log_{10} \frac{L(0.15)}{L(0.2)}$, and then,

$$\log_{10} \frac{L(0.2)}{L(0.1)} = \log_{10} \frac{L(0.1)}{L(0.15)} + \log_{10} \frac{L(0.15)}{L(0.2)}.$$

Using this method allows us to get better estimates than with simple_1 and simple_2, as seen clearly in Table 17.5. In fact, the results obtained with simple_3 are almost as good as those obtained with the square-root method. As expected, the square-root method is still the optimal, significantly better than the iterative method.

17.5 Discussion

We have applied the blocking Gibbs method successfully to a particularly hard problem in genetics, linkage analysis. The results in Table 17.4 documented that the algorithm converges towards the correct distribution and mixes fast. Single-site Gibbs sampling would have been totally useless applied to a problem of this size, and would also have been reducible. The LQT pedigree is just within the limits of exact methods so we have been able to check the accuracy of the results.

The blocking Gibbs method has shown to be successful in this very hard case and it can easily be applied to larger problems. Due to the way the blocks are selected, the method scales well, and we expect multipoint linkage analysis to pose no further theoretical problems. However, it will make the storage requirements exponentially higher with the number of loci. In Figure 17.4, the network representation of the three-loci linkage problem has been shown. It is seen that compared with Figure 13.6, it is a trivial extension to represent three or more loci.

θ_1 θ_0	0.01 0.1	0.1 0.2	0.2 0.3	0.3 0.4	0.4 0.5	sum
Diff.	$\log \frac{L(0.01)}{L(0.1)}$	$\log \frac{L(0.1)}{L(0.2)}$	$\log \frac{L(0.2)}{L(0.3)}$	$\log \frac{L(0.3)}{L(0.4)}$	$\log \frac{L(0.4)}{L(0.5)}$	$\log \frac{L(0.01)}{L(0.5)}$
Method						
simple_1	1.73 (0.01)	1.966 (0.006)	1.912 (0.008)	1.64 (0.01)	1.02 (0.02)	8.27 (0.05)
simple_2	1.664 (0.001)	1.856 (0.002)	1.783 (0.002)	1.524 (0.005)	0.968 (0.007)	7.80 (0.02)
simple_3	1.694 (0.002)	1.910 (0.003)	1.849 (0.004)	1.591 (0.006)	1.02 (0.01)	8.06 (0.03)
sqrt	1.700 (0.001)	1.912 (0.002)	1.850 (0.003)	1.591 (0.005)	1.019 (0.005)	8.07 (0.02)
iterative	1.7035 (0.0009)	1.914 (0.002)	1.854 (0.003)	1.599 (0.005)	1.031 (0.006)	8.10 (0.02)
Exact	1.6990	1.912	1.851	1.592	1.024	8.078

Table 17.5: A comparison of results with different estimation methods at 10,000 iterations. The last column is a sum of the five previous.

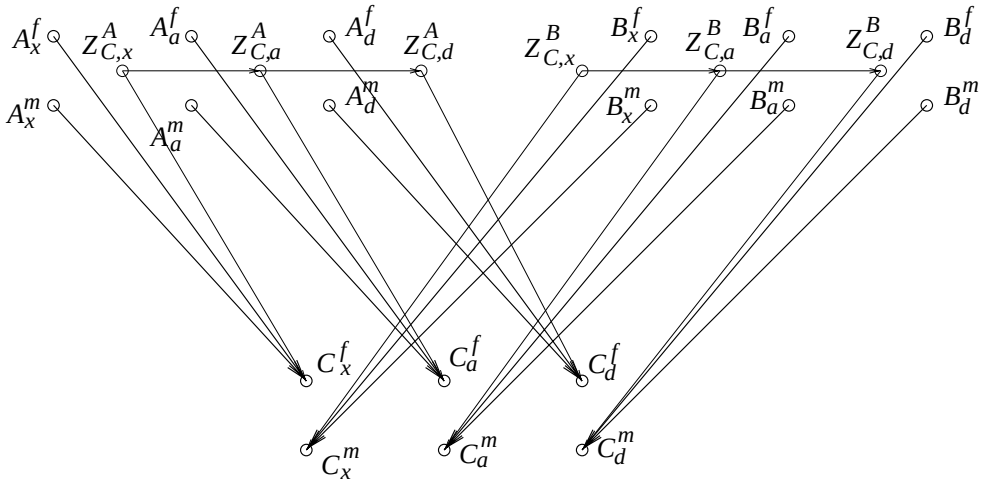


Figure 17.4: The Bayesian network representation of the three-loci linkage problem.

The major problem still remaining with the method is that we cannot yet prove that it is irreducible in the general case. This requires the construction of a general method for finding the noncommunicating classes of the Gibbs sampler such as discussed by Lin et al. (1994) and later by Jensen & Sheehan (1997). This is further discussed in Chapter 18.

It is at this point uncertain, however, whether such a general method can be found. Further, the problem of detecting these classes may be NP-hard. If these classes were identified it would be possible to design blocks tailored to allow the blocking Gibbs sampler to jump between the classes, thus guaranteeing irreducibility of the sampler.

In practice, it is often possible to design these blocks by hand such as has been

done in the case of the LQT pedigree, and the built-in robustness of the blocking Gibbs sampler, usually allowing it to sample more than 90% of the variables jointly, will render it irreducible in most cases. Still, of course, this cannot be guaranteed in the general case.

Chapter 18

Determination of the Noncommunicating Classes

This chapter corresponds with (Jensen & Sheehan 1997).

18.1 Introduction

As mentioned in Section 9.4 and Chapter 14, it is an acute problem for Markov chain Monte Carlo methods to ensure irreducibility of the Markov chain, in particular in pedigree analysis.

A full specification of the noncommunicating classes of the relevant Markov chain would be very pertinent to these methods. With such a specification, it would be relatively straightforward to design a relaxed sampler (e.g., the companion chain method of Lin et al. (1993)) that would move far more efficiently between the classes than contemporary MCMC methods. Furthermore, identifiability of these classes would allow for an intelligent blocking scheme that would guarantee irreducibility of the blocking Gibbs sampler described in Chapter 6, and would thus substantially increase the general applicability of these methods to very hard problems. Lin et al. (1994) propose a deterministic algorithm to fully determine the noncommunicating classes, or "islands", of genotypic configurations on any pedigree for any number of alleles. Lin (1995) then proposes a Metropolis jumping kernel algorithm to sample from the identified classes and extends this method to a multipoint linkage application in (Lin 1996). The purpose of this chapter is to point out that, unfortunately, the first of these two algorithms is *not* a general procedure for identifying the noncommunicating classes. It breaks down completely in a number of cases. To begin with, we describe the algorithm, then we outline the difficulties associated with it (we construct several counterexamples) and finally we discuss the problem of finding a useful algorithm.

18.2 The Island-Finding Algorithm

Islands are created by constraints imposed on genotypes of pedigree members by observed genetic data. Following Lin et al. (1994), they arise specifically "when data on children create noncommunicating alternatives for unobserved parents" and can thus "be identified by looking at each nuclear family successively from the bottom of a pedigree, tracing up." The following marriage chain example, taken directly

from Lin et al. (1994) illustrates how islands can be characterized and identified and how, in some situations, this process can be quite involved.

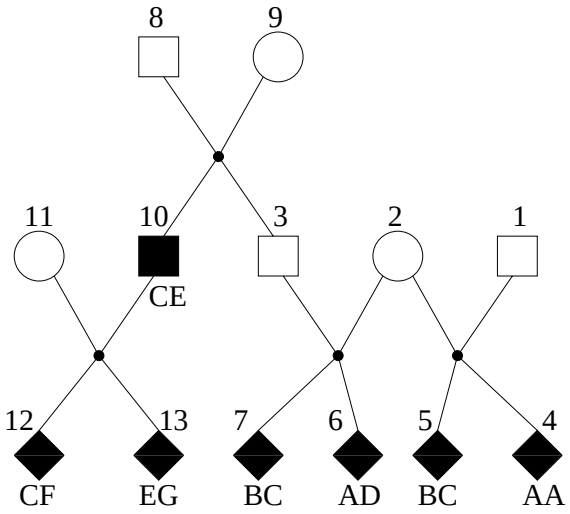


Figure 18.1: Example of a marriage chain with islands taken from Lin et al. (1994).

Consider a locus with seven alleles, denoted "A", "B", "C", "D", "E", "F" and "G" for the pedigree of Figure 18.1. The observed data on individuals 4 and 5 (AA and BC, respectively) constrain their parents, 1 and 2, to be either of the pairs ($g_1 = AB,\ g_2 = AC$), or ($g_1 = AC,\ g_2 = AB$), where g_i gives the genotype of individual i. In the notation of Lin et al. (1994), if we let $\mathbf{g_1}$ denote a configuration of genotypes on the entire pedigree in which the first case holds and $\mathbf{g_2}$, a configuration in which the second holds, we note that the legal configuration $\mathbf{g_2}$ can never be reached from the legal configuration $\mathbf{g_1}$ with a sampling scheme that updates individuals one at a time, such as the single-site Gibbs sampler. Thus, all the genotypic configurations that belong to the $\mathbf{g_1}$ class cannot be reached from any of the configurations that belong to the $\mathbf{g_2}$ class and so we have created two potential noncommunicating classes, or islands. Similarly, the data on their children 6 and 7 lead to four more noncommunicating possibilities on the genotype pairing of individuals 2 and 3, namely, ($g_2 = AB,\ g_3 = CD$), ($g_2 = AC,\ g_3 = BD$), ($g_2 = CD,\ g_3 = AB$) and ($g_2 = BD,\ g_3 = AC$). However, the last two of these possibilities are inconsistent with the islands created from the restrictions on 1 and 2. Hence the marriage chain linking individuals 1, 2 and 3 yields two islands formed by the genotype triplets ($g_1 = AB,\ g_2 = AC,\ g_3 = BD$) and ($g_1 = AC,\ g_2 = AB,\ g_3 = CD$). Finally, for the first of these possibilities, there are four more islands characterized by the genotype pairing for individuals 8 and 9 leading to five islands in total for this small example. These are enumerated in Table 18.1 taken from Lin et al. (1994).

Assuming that no mutation has occurred and that the pedigree is correct, Lin et al. (1994) claim that the following algorithm will find all the noncommunicating classes for any pedigree at a locus of arbitrary polymorphism. Each nuclear family is processed in turn and after a family has been processed, all offspring are discarded from the pedigree. Thus, "parents no longer have offspring present unless they are involved in other marriages which have not yet been processed." Furthermore, a nuclear family cannot be processed if any of the offspring are, themselves, parents in a family that has not yet been processed. An individual is defined as "typed" at

	Genotype of Individual				
Island	1	2	3	8	9
1	AC	AB	CD		
2	AB	AC	BD	BC	DE
3	AB	AC	BD	BE	CD
4	AB	AC	BD	DE	BC
5	AB	AC	BD	CD	BE

Table 18.1: Island characterization for the pedigree in Figure 18.1

the locus of interest if his genotype is either fully observed or else forced from the data on the pedigree. Otherwise, the individual is "untyped". The island-finding algorithm proceeds as follows :

1. Form a sequence of nuclear families from the bottom of the pedigree to the top of the pedigree.
2. For each nuclear family in the sequence do the following :
   (a) If both parents are untyped, find all the islands and check their consistency with other already-processed marriages of either member of this couple, if any.
   (b) If one parent is untyped, force the observed information from the children and the spouse to this untyped parent. Then check consistency with previously processed marriages of this parent, if any.
   (c) If both parents are typed, continue to the next family in the sequence immediately.
3. Print out the following information.
   (a) List of all the families in the order of processing.
   (b) Total number of islands and the characterization of each island.
   (c) The individuals who are forced to have certain genotypes from the pedigree data.

18.3 Counterexamples

The above algorithm is *not* able to find the islands for any pedigree at a locus of arbitrary polymorphism, as originally claimed. In this section we will give some instances in which it is guaranteed to fail. From the simplicity of these examples, it is clear that much more complex counterexamples can easily be constructed, so, in effect, we are showing that the algorithm may fail in virtually any type of pedigree.

The first and most simple example arises in the case of the full-sib mating depicted in Figure 18.2.

Here, there are two noncommunicating classes, or islands, characterized by ($g_3 = AB, g_4 = AC$) and ($g_3 = AC, g_4 = AB$), which are not detected by the algorithm, as described in (Lin et al. 1994). The AA genotype on the individual labelled as 1 constrains both his offspring, 3 and 4, to each carry an A-allele. The fact that one of them must also have a B allele and the other a C allele from the data on 5 is not picked up by the algorithm when processing the marriage of 3 and 4 since this information, in itself, does not create a reducibility problem.

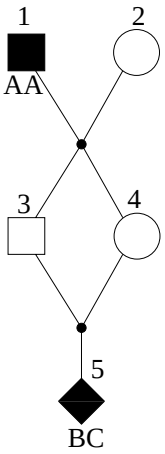


Figure 18.2: A simple inbreeding loop caused by a full sib mating.

The above example can be easily generalized to the slightly less inbred example of Figure 18.3.

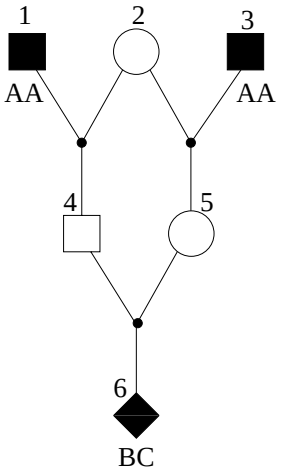


Figure 18.3: A marriage chain with a half-sib mating.

Again, two islands exist : $(g_4 = AB, g_5 = AC)$ and $(g_4 = AC, g_5 = AB)$, and again they are not found by the algorithm as it does not consider the effects of the data for 1 and 3 on their offspring 4 and 5 who, as a result, are each forced to carry an A allele.

In the context of human pedigrees, it might be argued that the pedigree structures of the last two examples are somewhat contrived. However, these structures would not be uncommon at all in animal pedigrees. The next example depicts a first-cousin mating (Figure 18.4) which would *not* be unusual in many human communities.

Again, as will be discussed in more detail in Section 18.4, the islands are not detected by the algorithm given in Section 18.2. In this case there are 4 islands characterized in Table 18.2. Individuals 11 and 12 are forced to carry an A allele

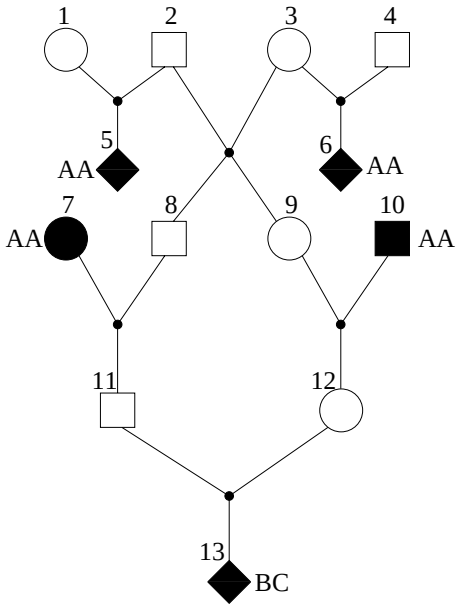


Figure 18.4: A more complex inbred pedigree depicting a first-cousin mating.

from their respective homozygous mother and father. The observed data on their common child, 13, causes a reducibility problem resulting in two islands specified as $(g_{11} = AB, g_{12} = AC)$ and $(g_{11} = AC, g_{12} = AB)$. However, for each of these two islands there are two other islands caused by 2 and 3: the B and C alleles carried by 13 must have originated from 2 and 3. As 2 and 3 each are forced to carry an A allele by their respective homozygous offspring 5 and 6, we have another reducibility problem resulting in two islands : $(g_2 = AB, g_3 = AC)$ and $(g_2 = AC, g_3 = AB)$. When all the islands are combined, we get the four islands of Table 18.2.

	Genotype of Individual			
Island	2	3	11	12
1	AB	AC	AB	AC
2	AB	AC	AC	AB
3	AC	AB	AB	AC
4	AC	AB	AC	AB

Table 18.2: Island characterization for the pedigree in Figure 18.4

For the example depicted in Figure 18.5, we see that the pedigree does not have to be inbred to create problems for the island-finding algorithm.

Here, the algorithm fails to account for the constraint that the data on 1 places on its offspring 4. The fact that 4 *must* carry an A allele, and either a B or C allele form the data on 6, combined with the fact that this B or C allele must originate from 2, creates two islands : $(g_2 = BD, g_4 = AB)$ and $(g_2 = CD, g_4 = AC)$.

Figure 18.6 shows a larger non-inbred pedigree with data creating the islands characterized in Table 18.3.

Information on their homogeneous parents at the top of the pedigree forces

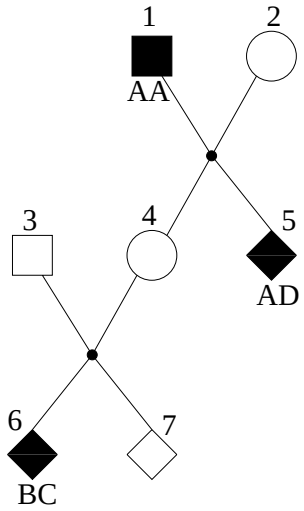


Figure 18.5: A simple non-inbred pedigree.

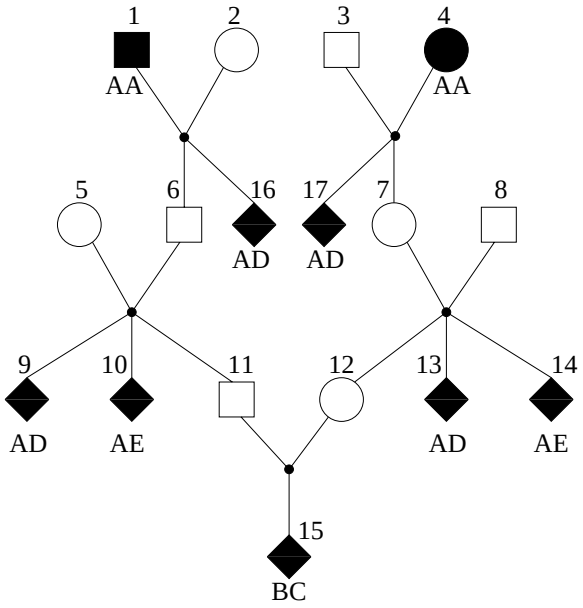


Figure 18.6: A more complex non-inbred pedigree.

individuals 6 and 7 to each carry an A allele. The observation that 15 has a BC genotype forces 11 and 12 to carry the alleles B and C. With the knowledge that B or C must be carried by 5 or 6, we can deduce that 5 must be DE as this is the only legal configuration given the three offspring, 9, 10 and 11. Likewise, 8 must be DE. This forces 6 and 7 to carry the B and C alleles which finally forces 2 and 3 to carry them as well. As 2 and 3 each carries a D allele, enforced by their respective offspring, we have a reducibility problem here, resulting in two islands ($g_2 = BD, g_3 = CD$) and ($g_2 = CD, g_3 = BD$). Considering the first of these two, 6 must have the genotype AB, and 7 must be AC. Also, 11 must carry the B allele

	Genotype of Individual					
Island	2	3	6	7	11	12
1	BD	CD	AB	AC	BD	CD
2	CD	BD	AC	AB	CD	BD

Table 18.3: Island characterization for the pedigree in Figure 18.6

and it can have either of the genotypes BD or BE. As a single-site Gibbs sampler can jump between these two configurations, no new islands are generated at this point. Likewise, 12 must carry the C allele. As a result, we get the first island in Table 18.3. The remaining island can be constructed in the same manner.

We conclude with the example in Figure 18.7 depicting a simple marriage chain where all the information is on the bottom generation. There are two islands created by the data in this example, characterized by ($g_2 = AB$, $g_3 = AC$) and ($g_2 = AC$, $g_3 = AB$). From the description given in (Lin et al. 1994), and quoted in Section 18.2, it would appear that each nuclear family be considered *separately*, in some order. As individuals 5, 6 or 7 are not involved in any marriages themselves, they can each be discarded as soon as their parents' marriage has been processed. Their parents may still have offspring present from their other unprocessed marriages, but at no stage is there enough information on any one mating to indicate a problem, and the algorithm fails again.

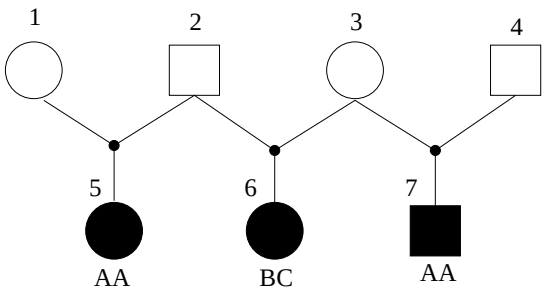


Figure 18.7: A marriage chain.

From the above illustration, it is obviously possible to make the counterexamples arbitrarily complex, thus requiring an algorithm far more sophisticated than the algorithm of Lin et al. (1994). When looking at small pedigrees like these examples, it is easy to locate the islands by eye and thus check whether or not the algorithm has done the right thing. However, in the large pedigrees that we really want to consider, such as the Greenland Eskimos, this cannot be done and there is no way of evaluating the outcome of such a procedure.

18.4 Why Does the Algorithm Fail ?

In order to quantify the limitations of the island-finding algorithm, we will illustrate how it would proceed, as we understand it from the description given in (Lin et al. 1994), on the example in Figure 18.4 of the previous section.

1. A sequence of nuclear families from the bottom to the top of the pedigree is

formed. This could for instance be: $(11, 12, 13)$, $(7, 8, 11)$, $(9, 10, 12)$, $(1, 2, 5)$, $(2, 3, 8, 9)$, $(3, 4, 6)$.

2. Each nuclear family is processed in turn :

   $(11, 12, 13)$**:** both parents are untyped but no islands are found. 13 is discarded and there is no information on 11 and 12 to cause a problem.

   $(7, 8, 11)$**:** 8 is untyped but no information can be forced upon it from 7 and 11.

   $(9, 10, 12)$**:** again, 9 is untyped but no information is forced upon it.

   $(1, 2, 5)$**:** both parents are untyped but no information can be forced upon them, and no islands are formed.

   $(2, 3, 8, 9)$**:** both parents are untyped but no islands are formed.

   $(3, 4, 6)$**:** again, both parents are untyped but no islands are formed.

3. No islands were found, so no information is printed out.

The algorithm fails in this case for two reasons. Firstly, it discards the information forced up onto 11 and 12 by their offspring, 13, which is vital when considering individuals 7, 8, 9, and 10. Had this information been propagated upwards, some of the islands would have been found. Secondly, it fails to take account of the information forced down onto 11 and 12 by their respective homozygous mother and father when looking for islands on 11 and 12. Had this been done the islands on 11 and 12 would have been found.

The fundamental problem with the island-finding algorithm is that it is based on the erroneous assumption that all the noncommunicating classes are due to "data on children creating noncommunicating alternatives for unobserved parents". In the counterexample illustrated in Figure 18.5, the data create noncommunicating alternatives for a father and son pairing. Clearly, once it is recognised that genetic information can also travel *down* a pedigree, it becomes a routine matter to construct counterexamples. We disagree with the comment by Lin (1996) that this general situation would be "unusual" in practice. Although genetic data are often observed only on the bottom two generations, it is by no means uncommon to have data elsewhere. Besides, generations are not always distinct as the terms "up" and "down" do not necessarily define disjoint sets in a complex pedigree. For a specific individual, Isaac, the pedigree members defined to be "above" Isaac are those related to him via his parents and hence comprise his ancestors, his siblings and all *their* descendants and their descendants' ancestors. Individuals "below" Isaac comprise those related to him via his children and hence include his spouses and *their* ancestors, etc., see Thompson (1986). The two terms are only really clearcut when the pedigree has no loops. This would usually be the case for the "practical" datasets arising from studies on Alzheimers' disease etc. referred to by Lin (1996). Pedigree information often has to be discarded for such analyses due to the lack of a satisfactory methodology for coping with it. In a large complex pedigree it would be very common to find individuals who are both related "above" and "below".

Even if the original assumption were true, however, *and* the terms "ancestor" and "descendant" were distinct, the island-finding algorithm would still not work, despite the claims in (Lin 1996) to the contrary. This is because it only seems to be able to cater for the cases in which genetic information on two or more children *of the same marriage* create a problem. The simple marriage chain example in Figure 18.7 clearly demonstrates that there are other ways of creating noncommunicating classes, even within the restrictions of the original inappropriate assumption. Of course, any sampling scheme which relies specifically on this algorithm will also be of limited use.

18.5 Discussion

It is obvious from the above, that a completely general algorithm would have to be able to take information from above and below into account. However, one simple upwards propagation of genetic information (such as that in (Lin et al. 1994) and one simple downwards propagation is not necessarily sufficient. The problem is illustrated by the diagram in Figure 18.8.

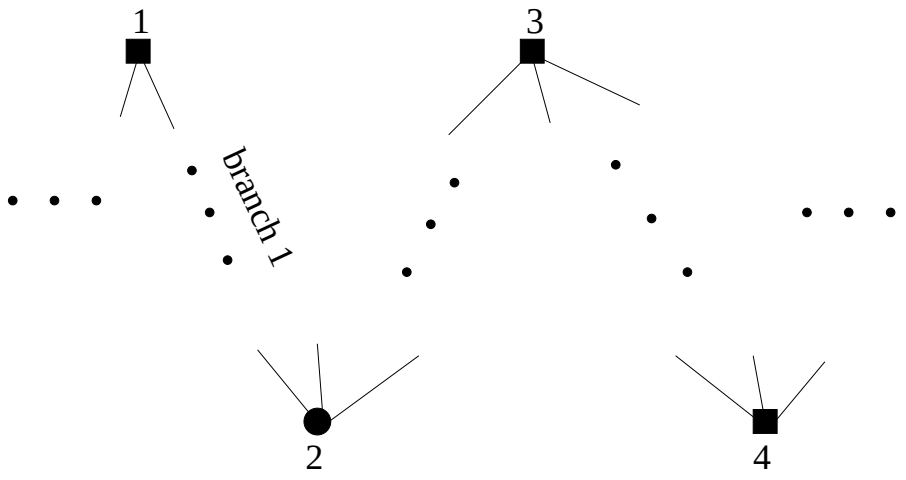


Figure 18.8: A larger pedigree illustrating that more than two propagations may be needed.

The downwards propagation of information along branch 1 in Figure 18.8 may cause new information to be created at individual 2, which after an upwards propagation may cause new information at individual 3, which may then percolate back down to individual 4. This again has to do with the notion of individuals being possibly related both through the parents *and* through the children. Individuals related to Isaac through his parents may very well be located below Isaac in the pedigree (e.g., descendants of siblings) and at least one upwards and one downwards propagation will be required before all relevant genetic information can be passed on to Isaac. It is uncertain whether any fixed number of such upward and downward propagations can be assigned to guarantee a complete dissemination of information. This would depend very much on the pedigree structure and the data at hand. One possibility would be to continue alternating between upward and downward propagations until no more information can be gained.

Furthermore, the representation of information used in the island-finding algorithm of Lin et al. (1994) is insufficient to support an algorithm that works in all generality. The island characterizations e.g., $(g_1 = CD, g_2 = AB, g_3 = AC)$ and the genotypes of offspring and spouse that are used to infer new islands during the processing of a nuclear family, are the only forms of information used in the original algorithm. However, to find more islands, the algorithm must be able to represent partial information that can be useful at a later stage. For instance, the information that an individual carries a specific allele is clearly not representable within the definition of the algorithm as outlined Section 18.2. Moreover, as it is difficult to determine which particular items of information will come into play at a later point, *all* of this partial information really should be carried along. There is no way of doing this within the framework of the above algorithm. We note, however, that if the possible configurations for all individuals were simply propagated through the

pedigree, the storage requirements would grow at the same rate as they do for the exact methods, such as peeling. Using *constraints* of the form "individual X must carry allele A", for example, to represent partial information could be the solution. Obviously, the constraints, themselves, can become increasingly complex.

Finally, although this particular algorithm fails, the problem that is addressed seems to us to identify a central issue in the general applicability of Markov chain Monte Carlo methods to the analyses of genetic data on large complex pedigrees. In many practical situations, important pedigree information is often ignored because exact calculations are infeasible and reducibility of the underlying Markov chain makes it impossible to obtain reliable estimates from simulation methods. Some more sophisticated analysis of the relevant graph theory will surely be required to make progress in this area.

Chapter 19

Discussion and Conclusions

In Chapter 8, a realistic method for finding a legal starting configuration for MCMC methods was presented. It was shown with empirical results that the method behaves well and has almost polynomial complexity in most real world problems. Even in specially designed "hard" problems, the algorithm behaved well and backtracked only rarely. However, it is clear that more investigations of the situations where backtracking occurs could be interesting. The method can easily be improved quite substantially. First, it would be beneficial to visit the variables in an order corresponding with their spatial arrangement in the network. This would prevent most backtracking as highly correlated variables (that are usually located close to each other) would be visited in close sequence. Another way of preventing backtracking would be to only "explode" a minimal proportion of the network, just sufficient to allow exact inference. This minimal proportion of the network could easily be selected using the methods for computing variables' reductions in storage requirements in Section 9.2.2. If only a minimal part of the network was "exploded", the majority of the network could be treated using the random propagation algorithm of Dawid (1992), allowing most of the variables to be sampled jointly.

Although, it has been shown that finding a legal configuration in a general Bayesian network is NP-hard, this algorithm is most often able to do it in polynomial time. Probably, it is possible to construct hard cases where the algorithm works in exponential time, however, it does seem like in most cases the algorithm is able to perform much better. It would be interesting to investigate further how the algorithm is able to do this and in which cases it fails.

In Chapter 9, a suboptimal method for selecting blocks was presented. The method is not completely optimal as it is local, i.e., it only looks one step ahead when selecting the next variable. Although looking more than one step ahead is possible, the complexity increases exponentially for each step looked ahead. The method for computing the reductions in storage requirements obtained when conditioning upon variables presented in Section 9.2.2 has not yet been implemented. A simpler approximate version of it was used to produce the empirical results of the thesis. These results indicate that the block selection algorithm performs adequately in many genetics problems, see Chapters 16 and 17. In Chapter 9 it was also discussed whether forcing variables with high reductions in storage requirements to be present in only one block lowers the performance of the blocking Gibbs sampler. Empirical results indicated that this is the case. However, as stated, it is often necessary to condition upon these "optimal" variables in as many blocks as possible to be able to construct blocks with sufficiently low storage requirements.

It is still not completely known which variables yields the highest reduction in

storage requirements, i.e., what makes these variables special when considering the Bayesian network. It would be interesting to investigate this further, and also look into the limits of the block selection algorithm, i.e., how large and complex networks it is possible to break into reasonably sized blocks within given storage constraints.

At the end of Chapter 12, a lot of underlying assumptions regarding the genetic models used by most geneticists and also in this thesis were outlined, e.g., no mutation, or different recombination fractions for males and females. Some of these assumptions are easily incorporated into the genetic model by adding extra variables to the Bayesian network, however, it is doubtful whether anything can be gained by doing this. Other assumptions are not so easily incorporated, e.g., that individuals do not mate completely at random, or that disease alleles are sometimes preferentially received from the father or from the mother. It would be interesting to investigate the importance of all of these assumptions, and how to incorporate them into the genetic model without jeopardizing the complexity of inference.

In Chapter 16, the blocking Gibbs sampler was applied to a highly complex problem in genetics, analysis of a large, inbred pig pedigree. The blocking Gibbs sampler was compared with the single-site Gibbs sampler, and highly convincing results showed the much better performance of the blocking Gibbs sampler. Due to the fact that a preliminary (to the one in Chapter 9) block selection scheme was used, it can be expected that the performance of the blocking Gibbs sampler is even higher.

Some rules of thumb were deducted from the empirical results of the chapter that bears repeating here. (i) The blocks should be as large as possible without increasing the iteration time significantly, which can basically be translated to, as large as possible given storage constraints. (ii) There should be as few blocks as possible, as more blocks increases the iteration time, but does not yield correspondingly faster convergence. Thus, again, as few blocks as possible should be constructed given the storage constraints. (iii) It was found that of the two block selection methods presented, one was better than the other, however, it is almost certain, though, not examined empirically, that the new block selection method of Chapter 9 is better than either of the two.

Then, in Chapter 17, the blocking Gibbs sampler was applied to another complex problem of genetics, linkage analysis of a relatively large inbred human pedigree. The blocking Gibbs sampler handled the problem relatively easily, and it is clear that the single-site Gibbs sampler would have been useless for this problem and reducible as well. Different ways of computing the maximum likelihood estimators were presented and compared empirically, and also new ways of combining results from different runs were presented and compared empirically. These new methods can also be used with other schemes than blocking Gibbs, and they generally make it possible to make better use of results and obtain higher precision than what has till now been possible.

The linkage analysis performed was two-point only, i.e., considered only two loci. Considering more than two loci at a time entails no new theoretical problems. However, the complexity grows almost exponentially with the number of loci, forcing blocking Gibbs to create smaller blocks. Like the method of sequential imputation (Kong et al. 1993, Irwin et al. 1994) where one locus is treated at a time, blocking Gibbs could create blocks each containing only the variables of a single locus to allow handling these multilocus linkage analysis problems.

Finally, in Chapter 18, the largest remaining problem with the blocking Gibbs sampler was discussed, namely that of finding blocks that guarantee irreducibility. This is so far an unsolved problem and it remains uncertain whether at all it can be solved. In the general case, this problem is NP-hard but it is possible that

a solution can be found in the case of pedigree analysis by exploiting some of the extra structure and information contained in a pedigree. In Chapter 18, a proposed algorithm (Lin et al. 1994) for locating the noncommunicating classes of an MCMC method was analyzed and its limitations were pointed out by a number of simple, realistic pedigrees. Some pointers toward a general algorithm were provided, however, much research still remains before a general algorithm has been found, or, alternatively, the problem has been proven NP-hard. In the latter case, it would be interesting to look for good heuristic methods.

All in all, the blocking Gibbs sampling scheme has been shown through theoretical and practical investigations to be a realistic approach for handling large and complex (looped) Bayesian networks, something no other method currently seems capable of. The basic problem remaining is reducibility. It is uncertain whether a solution can be found, however, as it is usually possible to construct blocks containing more than 90% of the variables, it can be argued that irreducibility will often be fulfilled. Whether this is in fact the case, can easily be checked in genetics problems, as shown in Section 16.7. Also, problems with inherent near-reducibility can be much relieved by the large blocks.

19.1 Directions of Future Research

There are several directions in which possibly fruitful research and development can still be performed.

- A method for finding blocks that guarantee irreducibility should be developed for applications in pedigree analysis. This has been discussed in great detail in Chapter 18, however, not resulting in any solution, and at this point it is uncertain whether it is possible in the general case.
- The new method for computing the reduction in storage requirements obtained when conditioning upon a variable presented in Section 9.2.2 should be implemented, and compared with the one used for the empirical results of Chapters 16 and 17.
- In Section 17.3, it is described how to estimate the recombination fraction in linkage analysis using a maximum likelihood estimator. The recombination fraction can also be estimated by including it as a continuous variable in the blocking Gibbs scheme. This continuous variable would be a parent of the indicator variables $Z^A_{C,d}$ and $Z^B_{C,d}$ in Figure 13.6 on page 93. Thus, in each iteration the recombination fraction would be sampled given the number of recombinations and non-recombinations in the pedigree, and the conditional distributions of the indicator variables would be changed to incorporate the new value of the recombination fraction. This method could also be used to incorporate continuous variables representing the population allele frequencies and penetrance probabilities. In Figure 13.2 on page 89 this would be implemented by inserting a variable representing the population allele frequencies as a parent of the founder genotypes, G_a, G_b, and G_c, and a variable representing the penetrance probabilities as a parent of all phenotype variables, $P_a, \ldots, P_h$. Methods for sampling these continuous variables are described in, e.g., (Thomas et al. 1992, Hansen & Pedersen 1994, Janss, Thompson & Van Arendonk 1995). Usually, the population allele frequencies and penetrance probabilities are specified beforehand when running the blocking Gibbs sampler, making the results dependent on the accuracy of these priors. Implementing continuous variables representing the distributions will yield a more flexible and complete model, independent of specific priors.

- As described in the thesis, not all variables are updated each time a block is visited. This can easily be changed, by applying single-site Gibbs to the remaining variables (the A-set), ensuring that all variables are updated each time a block is visited. Thus, an iteration would now only consist of sampling a single block, instead of visiting all blocks. Also, this would have the benefit that all variables are sampled equally often, though some, of course, are sampled with single-site Gibbs and some with blocking Gibbs. This should be implemented and compared with the pure blocking scheme defined in Chapter 6.

Appendix A

Manual for the Blocking Gibbs Software

During the working period of the thesis, much software associated with the blocking Gibbs sampling algorithm was developed. In this section, the manual of a software package placed into public domain is reproduced. The software package is a special version of the general blocking Gibbs software designed for pedigree analysis, and in particular linkage analysis. The main software program for performing pedigree analysis is denoted *block*. To perform linkage analysis, an auxiliary program must be used, the manual of which is also reproduced here. This program is denoted *theta*. The manuals are almost independent of the thesis, however, a few references to elaborating sections within the main thesis are provided.

A.1 Manual for *block*

A.1.1 Description and Purpose

The *block* program can be used for performing pedigree analysis (including linkage analysis). More specifically, it can be used for :

- Any pedigree analysis involving an arbitrary number of alleles, incomplete penetrance and liability classes. The pedigree may contain an arbitrary number of loops. The number of loops is limited only by memory (but may be large).
- Any two-point linkage analysis involving an arbitrary number of alleles at each locus. Convergence is guaranteed only in the case where both loci have two alleles. In cases with more alleles, convergence can be obtained by specifying user-defined blocks (read more about this later).

To the knowledge of the author, no other programs in the public domain can perform the two above mentioned tasks. Current available programs are very much limited by the number of loops in the pedigree, and is able to handle only very few (10-20 ?). *block* has been successfully running examples in pedigree analysis with thousands of loops.

The program basically functions in the following way :

1. A pedigree is read into memory, and converted to the junction tree representation described in Chapter 4. In the pedigree specification, an initial recombination fraction must be specified, if linkage analysis is performed.

2. A number of blocks are constructed, that can all be sampled exactly. The block selection procedure is described in Chapter 9. Precompiled blocks may also be read from disk.

3. A starting configuration is found according to the method described in Chapter 8.

4. Warm-up is performed. This is usually 10% of the iterations.

5. The specified number of iterations of blocking Gibbs are performed. If linkage analysis is performed, the number of recombinations and nonrecombinations are counted as described in Chapter 17. If linkage analysis is performed, the results can be further processed with the *theta* program, described in Section A.2.

6. The results are stored on disk.

A.1.2 Options

The *block* program is run from the command line, and can be supplied a large number of options. In the following each of these options will be explained. Help can also be obtained with the `-h` option. Most of the options can also be specified within the pedigree specification file. This is done by prepending the option with "`%`", and then using the keywords mentioned with each of the options, and possibly specifying a value for the option. An example where this has been used, can be seen in the supplied example, "`ped_ex5`", see Section A.3.5.

This is the format of the *block* program :

```
block [-hvBEHLQS] [-b#] [-C<conf-file>] [-d<data-file>] [-i#]
[-m#] [-M<substring>] [-n#] [-N#] [-O#] [-r#] [-R#] [-t#] [-w#]
[-x#] [-Z#] netfile
```

The "`netfile`" option contains the name of the file describing the pedigree analysis problem.

The following is a description of all the options :

`-b` This option specifies how to treat the blocks :

0 - Load precompiled blocks from disk.

1 - Construct new blocks, but do not save them (default).

2 - Construct new blocks and save them.

The option can also be set by specifying the option "`block storage = #`" within the pedigree specification file.

`-C` Specify file to load starting configuration from. This can be used to avoid having to find a new starting configuration for each run which may be very time-consuming.

`-D` Specify the method for selecting the blocks. There are three methods to choose from ranging from a slow method providing high quality blocks to a fast method providing blocks of lower quality. The methods are described in more detail in Section 9.2.1.

0 - Slow most optimal method (default).

1 - Faster less optimal method.

2 - Fastest least optimal method.

The option can also be set by specifying the option "`block selection = #`" within the pedigree specification file.

`-E` Attempt to treat the pedigree exactly. This may be possible for smaller pedigrees. In general, this results in obtaining the exact marginal distributions on all variables. For linkage analysis, it results in exact simulation which has not been implemented yet. The option can also be set by specifying the option "`exact`" within the pedigree specification file.

`-f` This option controls the forward sampling of barren variables as described in Chapter 10. All barren variables can be forward sampled instead of being included in the blocking Gibbs sampler. This enables *block* to make the blocks smaller, and thus use less memory. The precision of the estimates seems similar to the one obtained when using blocking Gibbs on all individuals. If `-f` is specified, the forward sampling of barren variables is turned off. The option can also be set by specifying the option "`no forward sampling`" within the pedigree specification file.

`-h` Show the help page.

`-H` Use memory for the backup of the initial junction tree probability tables. These tables must be stored somewhere as they are needed each time the junction tree is initialized. The default is to store them on disk. Depending on your available memory, you may be forced to store the tables on disk (which is the default), but you may also be able to get *block* to run faster, by storing the tables in main memory. The option can also be set by specifying the option "`use memory for backups`" within the pedigree specification file.

`-i` Number of iterations. All blocks are sampled once in each iteration. The option can also be set by specifying "`iteration = #`" within the pedigree specification file.

`-L` Perform linkage analysis. This option may be specified only when option `-N3` is also used. If option `-N3` is used, and `-L` is not specified, a simple inference is performed and the marginal probabilities of all variables (given the specified recombination fraction) are found. The option can also be set by specifying the option "`linkage analysis`" within the pedigree specification file.

`-m` Maximal amount of memory available for blocks, specified in units of 8 bytes (which is the usual storage requirements of a floating point number). Default is 100,000. The option can also be set by specifying the option "`max memory = #`" within the pedigree specification file.

`-M` Specify a list of variables to monitor in two ways :

1. `-M#name`$_1$`,name`$_2$`,...,name`$_n$`#`
2. `-Mname`$_1$`,name`$_2$`,...,name`$_n$

The first way causes *block* to monitor the variables with names identical to one of the specified names. The second causes *block* to monitor variables with names that contain one of the specified names. Thus, you can obtain either exact or substring match. The option can also be set by specifying the option "`monitor = #`" within the pedigree specification file.

`-n` Number of blocks to be constructed. The default is 5. You will notice that there are often constructed more blocks than specified. This is because *block* in many cases must construct extra blocks to ensure irreducibility if any of the problems described in Chapter 14 occur. If a very large and complex problem is being handled, it will most likely be necessary to specify a large number of blocks. First try should be with the default 5 blocks, then 10, 15, etc., can be tried, until *block* is able to construct blocks with sufficiently low storage requirements. The option can also be set by specifying the option "`number of blocks = #`" within the pedigree specification file.

`-N` Type of input file given to *block*:

1. Pedigree 1 format. Pedigree analysis with complete penetrance.
2. Pedigree 2 format. Pedigree analysis with incomplete penetrance.
3. Linkage analysis format.

The input file formats are described in Section A.1.3. The option can also be set by specifying the option "`type of pedigree = #`" within the pedigree specification file.

`-O` Specify number of iterations after which *block* will output the current configuration of all variables. The configuration will be stored in the file :

`work/<pedigree-name>/results/conf.<#iterations>`

The option can also be set by specifying the option "`output configurations at #`" within the pedigree specification file.

`-Q` Run *block* quietly with as little text output as possible.

`-r` The type of representation to use for pedigrees. This option is only for simple pedigree analysis (`-N1` and `-N2`). It has two values :

1. Variables represent genotypes (default).
2. Variables represent single genes.

Descriptions of these representations can be found in Chapter 13. Only the gene representation can be used when running linkage analysis. The genotype representation uses less memory than the gene representation, but the gene representation provides more information than the other. This information is needed when running linkage analysis, thus the genotype representation cannot be used here. The option can also be set by specifying the option "`representation = #`" within the pedigree specification file.

`-R` Force *block* to output intermediary results at the specified iterations. If no value is passed with `-R`, the default is to output results at 100, 200, 500, 1000, 2000, 5000, ..., iterations. A different list of numbers can be specified with :

`-R<#1>,<#2>,<#3>,<#4>, ... ,<#n>`

The intermediary results are stored in the file :

`work/<pedigree-name>/results/results.<#iterations>`

The option can also be set by specifying the option "`output results at #`" within the pedigree specification file.

`-s` Criterion governing the selection of blocks. This criterion specifies the maximum number of blocks that a variable can be removed from. This is further discussed in Section 9.2.1. A variable cannot be removed from all blocks, as

it would then never be sampled. If a very large and complex pedigree is being handled, it may be necessary to remove certain variables from most of the blocks for *block* to be able to update the variables in the blocks jointly. In this case, option `-s2` should most likely be used, as this allows variables to be removed from all blocks except one :

1. #blocks/2+1 (default).
2. #blocks-1.
3. #blocks/4+1.
4. 2·#blocks/3+1.

The option can also be set by specifying the option "`block max criterion = #`" within the pedigree specification file.

`-t` Triangulation method to use on the pedigree. Read more about this in Section 4.5. The default method is usually adequate, but in hard cases, `-t5` should be tried.

0 Default.

1 Minimum number of fill-in edges.

2 Minimum clique size.

3 Minimum clique weight (default).

4 Minimum fill-in weight.

5 Try each of the above 10 times and select the best.

The option can also be set by specifying "`triangulation method = #`" within the pedigree specification file.

`-v` Verbose mode on. When this option is set, a lot of extra information is printed while running. The option can also be set by specifying "`verbose`" within the pedigree specification file.

`-w` Percentage of iterations in the warm-up phase. The default is to perform 10% of the specified number of iterations as warm-up. Thus, if `-i100` is specified, first, 10 iterations of warm-up are performed, and then the 100 main iterations. The option can also be set by specifying "`warm up percentage = #`" within the pedigree specification file.

`-x` Number of extra simulations to perform when each block is visited. This option would be important if it is fast to perform extra simulations compared with stepping from one iteration to the next. This does not seem to be the case here, though. Thus, this option is rarely believed to be useful. It can also be set by specifying "`extra simulations = #`" within the pedigree specification file.

`-Z` Seed option. This option allows the user to use and modify the seed used for computing random numbers in *block*. The seed is stored in the file

`work/<pedigree-name>/general/SEED`.

0 Use old seed in "`SEED`".

1 Find new seed and save it in "`SEED`".

2 Use new seed but do not save it (default).

The option can also be set by specifying "`use seed = #`" within the pedigree specification file.

A.1.3 File Formats

In this section, the formats of the files used by *block* will be described. First, the input files describing the pedigree analysis problem will be described, then the log files, and finally the files containing the results.

Input files

There are three types of input files, declared with either the `-N1`, `-N2` or `-N3` option.

`-N1` Pedigree 1 format. This pedigree format should be used if a simple pedigree analysis with complete penetrance is wanted. Examples of such pedigrees can be found in "`ped_ex1`", Section A.3.1, and "`ped_ex2`", Section A.3.2. This format is very simple :

`#:` Comments can be specified by starting the line with "`#`".

`nalleles:` Number of alleles can be specified with "`nalleles =`".

`palleles:` Allele population frequencies can be specified with "`palleles = (`p_1 ... p_n`)`". If they are not specified, uniform frequencies will be assigned.

`block:` A user-defined block can be specified in one of the following ways (see "`ped_ex2`", Section A.3.2 for an example) :

`block expand:` A list of individual names must be given. A block will then be constructed that contains all the variables corresponding to these individuals.

`block exact:` A list of variable names must be given. A block will then be constructed that contains all these variables. The variables that correspond to an individual are different depending on the representation (`-r1` or `-r2`). For representation 1 (genotype), one variable is created for each individual (with the same name). For representation 2 (gene), the following variables may be created for an individual A :

A^f**:** A's paternal gene.

A^m**:** A's maternal gene.

A_g**:** the genotype of A.

A_x**:** extra variable created to hold evidence if A is heterozygous, e.g., if the genotype is Nn, it must enforce either the configuration $(A^f = N, A^m = n)$ or $(A^f = n, A^m = N)$.

`block:` Similar to `block expand`. In "`ped_ex1`", Section A.3.1, if the variables 1, 2 and 3 were not placed in the same block, the Markov chain would be reducible, and the Gibbs sampler would be stuck in the initial configuration. In "`ped_ex2`", Section A.3.2, there is an example of each block type. Without these two blocks, also this example would be stuck in the initial configuration.

`Pedigree:` The line "`Pedigree:`" must be present in the pedigree file, before the specification of individuals can begin.

individual specifications**:** Then, line after line, the data of individuals can be specified. There is one line for each individual containing the following information :

1. The name of the individual (up to 20 characters).
2. The name of the father ("`0`" if not in the pedigree).

3. The name of the mother ("0" if not in the pedigree). Currently, either both parents must be specified, or none of them.
4. The sex of the individual ("u" - undefined, "m" - male, "f" - female). Alternatively, the syntax ("0" - undefined, "1" - male, "2" - female) can be used.
5. Gene 1 (a number between 1 and `nalleles`, "0" if undefined).
6. Gene 2 (similar).

`-N2` Pedigree 2 format. This pedigree format should be used if a pedigree analysis with incomplete penetrance is wanted. An example of such a pedigree can be found in "`ped_ex3`", Section A.3.3. This format is like `-N1`, but with some extensions and minor changes :

`nphenotypes:` Number of phenotypes can be specified with "`nphenotypes = #`".

`phenotype names:` The phenotype names can be specified with "`phenotype names = (<name`$_1$`> ... <name`$_n$`>)`". See an example of this in "`ped_ex3`", Section A.3.3. The length of these names can be up to 20 characters.

`penetrance:` The penetrance probabilities can be specified with "`penetrance = ...`". As seen in "`ped_ex3`", Section A.3.3, there must be one line for each genotype. First, the genotype is listed, then the probabilities that each phenotype is observed given this genotype.

`block:` For an individual A, there is now also created a variable, A_p, which represents the phenotype of A.

individual specifications: The pedigree specification is much like with the pedigree 1 format. Here, the individual is specified like earlier, but with a phenotype instead of a genotype. `0` specifies an unobserved phenotype.

`-N3` Linkage analysis format. This format should be used if a two-point linkage analysis is wanted. An example of an input file following this format is "`ped_ex4`", Section A.3.4. The format is similar to the previous, but most keywords have been extended, and some new have been introduced to handle two loci :

`nloci:` Number of loci can be specified with "`nloci = #`". Currently this can only be set to 2.

`loci names:` The names of the two loci can be specified with "`loci names = (<name`$_1$`> <name`$_2$`>)`". The length of the names can be up to 20 characters.

`theta:` This is the recombination fraction used under the entire blocking Gibbs sampling run. The results will be produced using this value. It must be between 0 and 0.5.

`nalleles:` The number of alleles is now specified with "`nalleles = (<nalleles at locus 1> <nalleles at locus 2>)`".

`palleles#:` The allele population frequencies are now specified with "`palleles<locus #> = (`p_1 ... p_n`)`".

`use penetrance:` This keyword specifies for each locus whether it has complete or incomplete penetrance. If incomplete penetrance is wanted for a locus, this is specified with a `1`. Thus, this is specified for both loci with "`use penetrance = (<pen`$_1$`> <pen`$_2$`>)`".

`nphenotypes#:` The number of phenotypes at a locus is now specified with "`nphenotypes<locus #> = <no. of phenotypes>`".

`phenotypes names#:` The phenotype names at a specific locus is now specified with "`phenotype names<locus #> = (<name`$_1$`> ... <name`$_n$`>)`".

`penetrance#:` The penetrance probabilities at a locus are now specified with "`penetrance<locus #> = ...`". The actual specification of the probabilities is done similarly as with format 2.

`block#:` A block can now be specified as belonging to a certain locus, i.e., a block containing only variables corresponding with a specific locus is specified as "`block<locus #> = (<name`$_1$`> ... <name`$_n$`>)`".

individual specifications**:** The specification of an individual in the pedigree is much like before. First, the names of the individual itself and its father and mother are given. Then, the sex of the individual is specified, followed by, for each of the two loci, either the two genes or the phenotype depending on whether complete or incomplete penetrance is specified for the locus.

Log files

In this section, the log files output by *block* will be described. The log files reside in "`work/<pedigree name>/log`" if nothing else is mentioned.

`main_log:` This file contains a log from the compilation of the pedigree to the junction tree representation described in Chapter 4. The file contains much information that can be useful, for instance on the cliques that are constructed (the size of them and the variables they contain). The file also contains the total size of the junction tree.

`generations:` This file contains information on the number of generations in the pedigree, and the generation number of each variable.

`complexity_reduction:` This file contains output from the algorithm that finds the optimal blocks, described in Chapter 9. First, the complete junction tree is listed, with the cliques and the separators. Then, the selection algorithm is started. One variable at a time is selected, and each line contains information regarding the block the variable is removed from, the complexity reduction (`c.r.`) of the variable, how much storage requirements remains for the block, and last the size of the largest clique (`lc`), and some information on the largest cliques.

`barren_nodes:` This file lists the barren variables of the pedigree.

`initial_conf:` This file resides in the directory "`general`". It contains the initial configuration for the first block. It is always saved, and can be reused to make *block* start faster (with the `-C` option).

`exact_log:` This file only appears when *block* is able to treat the pedigree in an exact manner. It is a log from the compilation of the pedigree to a junction tree, with the same format as "`main_log`".

`exact.tables:` This file only appears when *block* is able to treat the pedigree exactly. It is used internally by *block* to save information about the junction tree.

`SEED:` This file resides in the directory "`general`". It contains the seed for the random number generator. It can be controlled with the `-Z` option.

`blocks_log`**:** This file contains information about the selected blocks. For each block, the number of variables that have been removed from it is listed, along with the percentage out of the total number of variables that have been removed.

`explode.log`**:** This file is created by the algorithm that finds the starting configuration. It contains the compilation log of the "exploded" junction tree introduced in Section 8.1.

Block files

In this section, the files output for each block are described. They all reside in "`work/<pedigree name>/blocks/block<#>`".

`compile_log`**:** This is a log from the compilation of the pedigree representing this block to a junction tree. It has the same format as "`main_log`".

`<pedigree name>.<block#>.net`**:** This file contains the block represented with a HUGIN specification (Fischer 1990). In this file the names and relations of all variables can be read, as well as their prior probability tables.

`B-set`**:** This file lists the variables that have been removed from the block.

`cut_corrs`**:** This file lists some more information on the variables that have been removed from this block. It lists the names of the variables that are created when breaking the loops of these variables.

`block.bg`**:** This is a storage file for the junction tree corresponding to the block. If option `-b0` is used, this file is loaded and used, instead of creating a new junction tree.

`tables.bg`**:** This is an internal file used by *block*.

Result files

In this section, the files containing results from *block* are described. These files all reside in "`work/<pedigree name>/results`".

`results.<iteration>`**:** This file contains the results for each variable in the pedigree. It lists the name of the variable, and the resulting marginal distribution after `<iteration>` iterations.

`short.<iteration>`**:** This file contains the same as the previous one, but in a shorter format.

`conf.<iteration>`**:** This file is constructed, if option `-O` is used. It contains the configuration of the pedigree after `<iteration>` iterations.

`link,<ped. name>,<`θ_0`>,<iteration>`**:** This file is constructed when doing linkage analysis. At each iteration and each block treated at that iteration, it lists the number of recombinations, the number of nonrecombinations, and a list of estimated recombination probabilities. This file should be used as input to the *theta* program described in Section A.2.

A.1.4 Hints and Tips

Block selection

If analysis is performed on a very large and complex pedigree, *block* may have trouble selecting blocks with sufficiently low storage requirements. Various parameters can be adjusted to help it :

1. The number of blocks can be increased. This is controlled with option `-n`. If *block* is allowed to construct more blocks, it is also able to make the storage requirements of the individual blocks smaller.

2. The number of blocks that a variable can be removed from should be increased. This is controlled by option `-s`. Option `-s2` should be used in hard cases.

3. The most optimal method for selecting blocks should be used. This is controlled with option `-D`.

4. If a simple pedigree analysis is performed, `-r1` should be used, as this representation uses less memory.

5. More triangulations of the initial pedigree should be attempted. This is controlled with option `-t`. `-t5` ensures that each triangulation method is attempted 10 times, and the best triangulation is used.

6. Force *block* to use less memory by using the `-m` option. Using this option does not guarantee, that *block* will use the specified amount of memory, but it will attempt to.

More than 2 alleles - reducibility

block can not always ensure that pedigree analysis with more than 2 alleles are handled correctly, as it does not know how to construct blocks such that irreducibility is guaranteed. It is likely, though, that in many cases with more than 2 alleles, *block* does yield the correct results, as it is able to make the blocks large and sample most of the variables jointly.

To ensure irreducibility, user-defined blocks can be constructed. Examples of this can be seen in the examples, "`ped_ex1`", Section A.3.1, and "`ped_ex3`", Section A.3.3. How to correctly specify these blocks is not easy, but hints can be found in Chapter 14. In smaller linkage studies it is often possible to place all the variables of the first locus in one block, and all the variables of the second locus in another block which ensures irreducibility.

To test whether irreducibility holds, the pedigree analysis can be converted to a diallelic study by representing the $n-1$ least frequent alleles as 1 allele. If this study yields similar probabilities on the remaining allele for all variables, this is a clear indication that irreducibility holds. It is also further described in Section 16.7.

A.1.5 Differences for PC-DOS Version

Due to the limitations of PC-DOS (MS-DOS and variants from other companies), this version uses different names for the various files. Here is a list of the names used under PC-DOS and the files they correspond to :

Name under PC-DOS	Name in other distributions
<ped. name>_<block#>.net	<ped. name>.<block#>.net
data.err	data_errors
data.ld	data_loaded
generati.ons	generations
compl.red	complexity_reduction
initial.con	initial_conf
cut.cor	cut_corrs
link,<iteration>	link,<ped. name>,<θ_0>,<iteration>

A.1.6 Examples

We want to perform a linkage analysis on the supplied pedigree, "ped_ex4", Section A.3.4. First, we set the recombination fraction in the pedigree specification to a good starting value, say, 0.25, i.e., $\theta_0 = 0.25$. We then perform an initial analysis of 100 iterations :

```
block -vL -N3 -i100 ped_ex4
```

Note that you do not have to specify "-i100" in this case, as this is also the default. When the program has finished, we get the results in :

```
work/ped_ex4/results/link,ped_ex4,0.25,100
```

The results are analyzed with *theta* in the following way, e.g. :

```
theta -DV -X1 link,ped_ex4,0.25,100
```

If you have *gnuplot* and *ghostview* present, you will get a graph displayed on the screen. This graph clearly indicates that the most probable recombination fraction is 0. If you want to further examine this, change the recombination fraction in the pedigree to, e.g., 0.05, i.e., $\theta_0 = 0.05$. Then run the linkage study for 1000 (or more) iterations :

```
block -vL -N3 -i1000 ped_ex4
```

A.2 Manual for *theta*

A.2.1 Description and Purpose

The *theta* program can be used to estimate log-likelihood differences for recombination fractions. It is applied to output from the *block* program, and can perform three different types of estimations :

- Given output from *block* obtained at a single run at θ_0, it can estimate the log-likelihood difference of another recombination fraction (θ_1) from θ_0.
- Given output from *block* obtained at two runs with first θ_0, then θ_1, estimate the log-likelihood difference of θ_1 and θ_0 using the square-root estimation method described in Section 17.3.
- Like the above, but using the iterative method also described in Section 17.3.

A two-point linkage analysis can thus be performed by running *block* on a linkage problem, and then analyzing the output with *theta*.

A.2.2 Options

The *theta* program is run from the command line, and can be supplied a number of options. In the following these options will be explained. Help can also be obtained with the `-h` option.

This is the format of the *theta* program :

`theta [-hDPGFVE] [-X<value>] [-s<start>] [-e<end>] <file1> [<file2>] [θ1]`

"`file1`" contains the name of a file with results obtained by running *block* for some time, and must always be given. "`file2`" contains the name of a second file with results from *block* that must be given if option `-X2` or `-X3` is used. If `-X1` is used, only "`file1`" should be specified.

`-D` When this option is given, the program will calculate log-likelihood differences for θ values between 0 and 0.5, and attempt to display the graph with *gnuplot*. If *gnuplot* is not present on your system, you can access the output directly in the "`gnuplot.input`" file. If *gnuplot* is present, you can access the produced output graph (as a PostScript file) in "`graph.<θ0>.ps`".

`-e` When option `-D` or `-P` is used, log-likelihood differences are usually calculated for θ values between 0 and 0.5. With this option and `-s`, it is possible to specify a different starting and ending point. `-e` is obviously used to specify the ending point. Thus, options `-s` and `-e` can be used to magnify pieces of the graph.

`-E` When using option `-D` or `-P`, the endpoints of θ values sometimes produce extreme values, making it hard to see the graph. This option forces the program not to compute log-likelihood differences for these endpoints.

`-F` The numbers that are output to the "`gnuplot.input`" file, are printed using the *C* statement `printf("%f")`. This is the default, and usually works well with *gnuplot*.

`-G` Like the above, except that the numbers are output using the *C* statement `printf("%g")`.

`-h` Print a help page with short descriptions of the options.

`-P` This option is similar to option `-D`, except that the graph will not be displayed.

`-s` Like option `-e`, but specifies the starting point of the graph.

`-V` The standard deviation is computed for each log-likelihood difference. The autocorrelations method of Geyer (1991) is used.

`-X` This option is used for specifying which method to use when estimating the log-likelihood differences :

1. One file is used, containing the output from a linkage problem run with *block* at a specific θ-value (θ_0). A graph over the log-likelihood differences can be obtained using option `-D` or `-P`, or the log-likelihood difference for θ_1 and θ_0 can be found by specifying θ_1.
2. The square-root method described in Section 17.3 is used for computing a log-likelihood difference of θ_1 and θ_0, using the results from runs at both θ_1 and θ_0, supplied as "`file1`" and "`file2`", see above.
3. Like `-X2`, except that the iterative method described in Section 17.3 is used here. Again, results from two runs with *block* must be supplied.

A.2.3 File Formats

In this section, the few files that are used, and output by the *theta* program are described.

`file`$_1$**:** This file must always be specified. It should contain the results from the *block* program when applied to a linkage analysis problem. In the first line of it, the recombination fraction at which it was produced, is listed. In the second line, the number of blocks in this scheme is listed, and then follows the number of recombinations, the number of nonrecombinations, and a list of estimated recombination probabilities for bottom level individuals. This file is located in the "`block`" directory, and is :

`work/<X>/results/link,<X>,<`θ`>,<iterations>`

where `<X>` is the name of the pedigree, θ is the recombination fraction that *block* used for this linkage study, and `iterations` is the number of iterations.

`file`$_2$**:** This file must be specified if option `-D` or `-P` is used. Otherwise, it should not be specified. Like `file`$_1$ it should contain results from *block* when applied to a linkage analysis problem. The results in `file`$_2$ should be produced with a different recombination fraction than those of `file`$_1$, otherwise the results will be worthless.

`gnuplot.input`**:** This file contains the input to the *gnuplot* program. The first column contains the θ_1 value, i.e., the log-likelihood difference is found for θ_1 and θ_0, where the results are obtained by sampling using θ_0. The second column contains the estimated log-likelihood difference for this θ_1 value. If option `-V` is given, three more columns are present. The two first contains the log-likelihood difference minus and plus the standard deviation, and the last contains the standard deviation. The standard deviation is estimated using the autocorrelations method described by Geyer (1991).

`graph.<`θ_0`>.ps`**:** This file contains the PostScript format graph over the log-likelihood differences. The highest point on this curve shows the estimated most likely recombination fraction.

A.2.4 Hints and Tips

The highly useful *gnuplot* program can be obtained by anonymous ftp from, e.g., `prep.ai.mit.edu/pub/gnu`. *ghostview* is also needed, and it can be obtained from the same address. If you are not able to obtain *ghostview* for your architecture, and you have another PostScript viewer available that you would like to use, please write me, and I will make this viewer available as an option in a future revision.

You can also just use your PostScript viewer on the output file from *theta* which is called "`graph.<`θ`>.ps`" where `<`θ`>` is the recombination fraction under which these results were produced.

A.2.5 Examples

We have run a linkage analysis on the pedigree, "`ped_ex4`", Section A.3.4, and we have been using the recombination fraction 0.33, and number of iterations have been 100 (`block -vL -N3 -i100 ped_ex4`). The results for *theta* are then located in the directory :

`work/ped_ex4/results/`

and is called :

```
link,ped_ex4,0.33,100
```

To produce a graph of the log-likelihood differences for these results, do :

```
theta -DV -X1 link,ped_ex4,0.33,100
```

It is possible to increase the precision of part of the graph, by supplying the `-s` and `-e` options :

```
theta -DV -X1 -s0.2 -e0.3 link,ped_ex4,0.33,100
```

Finally, if only a single log-likelihood difference is wanted, this can be done by leaving out the `-D` option, and supplying the θ value that the log-likelihood difference is wanted for :

```
theta -V -X1 link,ped_ex4,0.33,100 0.2
```

In this example, we thus get the log-likelihood difference : $\log_{10}\left(\frac{L(0.2)}{L(0.33)}\right)$.

A.3 Pedigree Examples

In this section, the pedigree examples used with the documentation of *block* and *theta* are presented.

A.3.1 Example 1

The pedigree example presented below corresponds with the one in Figure A.1.

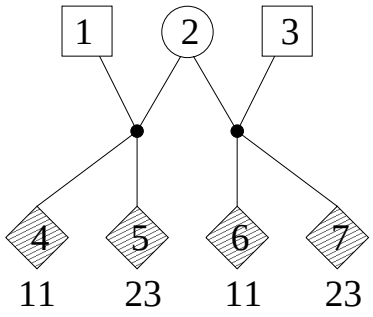


Figure A.1: A small pedigree test example for *block*.

```
# Pedigree example 1
# Alleles must be named: 1,2,3,4,... 0 means undefined/not measured.
# Sex (u = undefined, m = male, f = female).
# The names of individuals can be any strings of length less than 20
# with no spaces.
# If either allele 1 or allele 2 is specified, both must be.
# Number Father Mother Sex Allele-1 Allele-2
nalleles = 3
palleles = (0.25 0.25 0.5)
block = (1 2 3)
Pedigree:
1 0 0 m 0 0
2 0 0 f 0 0
```

```
3 0 0 m 0 0
4 1 2 u 1 1
5 1 2 u 2 3
6 3 2 u 1 1
7 3 2 u 2 3
```

A.3.2 Example 2

The pedigree example presented below corresponds with the one in Figure A.2.

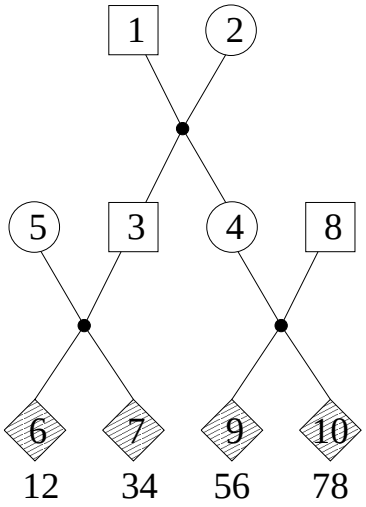


Figure A.2: A small pedigree test example for *block*.

```
# Pedigree example 2
nalleles = 8
block expand = (1 2 3 5)
block exact = (1.f 1.m 1.g 2.f 2.m 2.g 4.f 4.m 4.g 8.f 8.m 8.g)
Pedigree:
1 0 0 m 0 0
2 0 0 f 0 0
3 1 2 m 0 0
4 1 2 f 0 0
5 0 0 f 0 0
6 3 5 u 1 2
7 3 5 u 3 4
8 0 0 m 0 0
9 4 8 u 5 6
10 4 8 u 7 8
```

A.3.3 Example 3

The following pedigree example corresponds with the one in Figure 8.3 on page 52 and so does the ones in Sections A.3.4 and A.3.5.

```
# Pedigree on page 1 in the article:
```

```
#   Augustine Kong: 'Efficient Methods for Computing Linkage Likelihoods
#   of Recessive Diseases in Inbred Pedigrees', Genetic Epidemiology, 1991,
#   vol. 8, pp. 81-103.
# This version with all evidence, as shown in the figure.
# This pedigree is specified in 'pedigree 2 format'. In this format,
# the number of alleles is defined with 'nalleles = #'. Then, the
# probabilities of the <n> alleles are defined, like this:
# 'palleles = (0.2 0.3 0.1 0.01)'. The sum doesn't have to be 1, it
# is simply normalized. Then, the number of phenotypes is defined:
# 'nphenotypes = #'. Then, the names of phenotypes are defined:
# 'phenotypes = (p1 ... p<n>)'. Then, the penetrance probabilities
# are defined:        p1    p<n>
# 'penetrance = 1 1 : (# ... #)
#               1 2 : (# ... #)
#               ...'
# Here again, probabilities don't have to add up to 1.
# Then, finally the pedigree structure and data can be specified.
# Each line should look like this:
# Name Father Mother Sex Phenotype
nalleles = 2
palleles = (0.25 0.25)
nphenotypes = 2
phenotype names = (a n)
penetrance = 1 1 : (1.0 0.0)
             1 2 : (0.0 1.0)
             2 2 : (0.0 1.0)
Pedigree:
1 60 61 m n
2 62 63 f n
3 1 2 f n
4 1 2 m n
5 1 2 m n
6 0 0 f n
7 3 5 f n
8 3 5 f n
9 4 6 m n
10 4 6 m n
11 4 6 m n
12 0 0 m n
13 0 0 f n
14 0 0 m n
15 0 0 f n
16 0 0 f n
17 7 12 m n
18 7 12 m n
19 9 13 f n
20 9 13 f n
21 9 13 f n
22 8 14 m n
23 10 15 f n
24 11 16 m n
25 72 73 m n
26 70 71 f n
27 17 19 u a
```

```
28 17 19 u a
29 17 19 u n
30 17 19 u n
31 17 19 u n
32 17 19 u n
33 17 19 u n
34 17 19 u n
35 17 19 u n
36 17 19 u n
37 17 19 u n
38 17 19 u n
39 18 20 u a
40 18 20 u n
41 21 22 u a
42 21 22 u n
43 21 22 u n
44 21 22 u n
45 21 22 u n
46 21 22 u n
47 21 22 u n
48 21 22 u n
49 21 22 u n
50 21 22 u n
51 21 22 u n
52 21 22 u n
53 21 22 u n
54 23 25 m n
55 24 26 f n
56 54 55 u a
57 54 55 u n
58 54 55 u n
59 54 55 u n
60 0 0 m n
61 0 0 f n
62 0 0 m n
63 0 0 f n
64 60 61 m n
65 62 63 f n
66 0 0 m n
67 64 65 f n
68 1 2 m n
69 0 0 f n
70 68 69 m n
71 66 67 f n
72 3 5 m n
73 0 0 f n
```

A.3.4 Example 4

```
# This pedigree is used to test the 'linkage analysis' format.
# 'nloci' defines the number of loci.
# 'loci names' defines the names of the loci.
# 'theta' defines the starting recombination fraction.
```

```
# 'nalleles' defines the number of alleles in loci 1, loci 2, ...
# 'palleles1' defines the prior allele probabilities for loci 1.
# 'palleles2' does the same for loci 2, etc.
# 'use penetrance' states whether penetrance probabilities are used
#    for the pedigree at loci 1, loci 2, etc.
# 'nphenotypes1' defines the number of phenotypes at loci 1.
# 'nphenotypes2' does the same at loci 2, etc.
#    'nphenotypesx' should only be specified if 'use penetrance' has
#    a 1 at the x'th position.
# 'phenotype namesx' specifies the names of the phenotypes at loci <x>.
# 'penetrancex' specifies the penetrance probabilities at loci <x>.
# 'blockx' specifies the nodes to be blocked at loci <x>.
nloci = 2
loci names = (a d)
theta = 0.33
nalleles = (4 2)
palleles1 = (0.25 0.25 0.25 0.25)
palleles2 = (0.005 0.995)
use penetrance = (0 1)
nphenotypes2 = 2
phenotype names2 = (a n)
penetrance2 = 1 1 : (1 0)
1 2 : (0 1)
2 2 : (0 1)
# blocks for all families
block expand = (60 61 1 64 3 5 7 8 72 26 24 55)
block expand = (62 63 2 65 7 12 17 18 54 55 56 57 58 59)
block expand = (1 2 3 4 68 64 65 67)
block expand = (6 4 9 10 11 72 73 25)
block expand = (68 69 70 9 13 19 20 21)
block expand = (66 67 71 14 8 22)
block expand = (10 15 23 17 19 27 28 29 30 31 32 33 34 35 36 37 38)
block expand = (70 71 26 18 20 39 40)
block expand = (11 16 24 25 23 54)
block expand = (21 22 41 42 43 44 45 46 47 48 49 50 51 52 53)
Pedigree:
1 60 61 m 0 0 n
2 62 63 f 0 0 n
3 1 2 f 0 0 n
4 1 2 m 0 0 n
5 0 0 m 0 0 n
6 0 0 f 0 0 n
7 3 5 f 0 0 n
8 3 5 f 0 0 n
9 4 6 m 1 2 n
10 4 6 m 0 0 n
11 4 6 m 0 0 n
12 0 0 m 0 0 n
13 0 0 f 3 4 n
14 0 0 m 0 0 n
15 0 0 f 0 0 n
16 0 0 f 0 0 n
17 7 12 m 1 3 n
18 7 12 m 1 2 n
```

```
19 9 13 f 1 4 n
20 9 13 f 1 4 n
21 9 13 f 1 3 n
22 8 14 m 1 4 n
23 10 15 f 1 4 n
24 11 16 m 1 3 n
25 72 73 m 2 3 n
26 70 71 f 2 4 n
27 17 19 u 1 1 a
28 17 19 u 1 1 a
29 17 19 u 1 3 n
30 17 19 u 1 3 n
31 17 19 u 1 3 n
32 17 19 u 1 3 n
33 17 19 u 1 4 n
34 17 19 u 1 4 n
35 17 19 u 1 4 n
36 17 19 u 3 4 n
37 17 19 u 3 4 n
38 17 19 u 3 4 n
39 18 20 u 1 1 a
40 18 20 u 2 4 n
41 21 22 u 1 1 a
42 21 22 u 1 3 n
43 21 22 u 1 3 n
44 21 22 u 1 3 n
45 21 22 u 1 3 n
46 21 22 u 1 4 n
47 21 22 u 1 4 n
48 21 22 u 1 4 n
49 21 22 u 1 4 n
50 21 22 u 3 4 n
51 21 22 u 3 4 n
52 21 22 u 3 4 n
53 21 22 u 3 4 n
54 25 23 m 1 3 n
55 24 26 f 1 4 n
56 54 55 u 1 1 a
57 54 55 u 1 3 n
58 54 55 u 1 4 n
59 54 55 u 3 4 n
60 0 0 m 0 0 n
61 0 0 f 0 0 n
62 0 0 m 0 0 n
63 0 0 f 0 0 n
64 60 61 m 0 0 n
65 62 63 f 0 0 n
66 0 0 m 0 0 n
67 64 65 f 0 0 n
68 1 2 m 0 0 n
69 0 0 f 0 0 n
70 68 69 m 0 0 n
71 66 67 f 0 0 n
72 3 5 m 0 0 n
```

```
73 0 0 f 0 0 n
```

A.3.5 Example 5

```
# This pedigree is used to test the 'linkage analysis' format.
# 'nloci' defines the number of loci.
# 'loci names' defines the names of the loci.
# 'theta' defines the starting recombination fraction.
# 'nalleles' defines the number of alleles in loci 1, loci 2, ...
# 'palleles1' defines the prior allele probabilities for loci 1.
# 'palleles2' does the same for loci 2, etc.
# 'use penetrance' states whether penetrance probabilities are used
#    for the pedigree at loci 1, loci 2, etc.
# 'nphenotypes1' defines the number of phenotypes at loci 1.
# 'nphenotypes2' does the same at loci 2, etc.
#    'nphenotypesx' should only be specified if 'use penetrance' has
#    a 1 at the x'th position.
# 'phenotype namesx' specifies the names of the phenotypes at loci <x>.
# 'penetrancex' specifies the penetrance probabilities at loci <x>.
# 'blockx' specifies the nodes to be blocked at loci <x>.

# First, specify the options such that this pedigree can be run simply
# by writing 'block ped_ex5' :

%block storage = 2
%block selection = 1
%no forward sampling
%use memory for backups
%iterations = 1000
%linkage analysis
%number of blocks = 10
%type of pedigree = 3
%triangulation method = 0
%verbose
%warm up percentage = 15
%use seed = 2

nloci = 2
loci names = (a d)
theta = 0.33
nalleles = (4 2)
palleles1 = (0.25 0.25 0.25 0.25)
palleles2 = (0.005 0.995)
use penetrance = (0 1)
nphenotypes2 = 2
phenotype names2 = (a n)
penetrance2 = 1 1 : (1 0)
1 2 : (0 1)
2 2 : (0 1)
# blocks for all families
block expand = (60 61 1 64 3 5 7 8 72 26 24 55)
block expand = (62 63 2 65 7 12 17 18 54 55 56 57 58 59)
block expand = (1 2 3 4 68 64 65 67)
```

```
block expand = (6 4 9 10 11 72 73 25)
block expand = (68 69 70 9 13 19 20 21)
block expand = (66 67 71 14 8 22)
block expand = (10 15 23 17 19 27 28 29 30 31 32 33 34 35 36 37 38)
block expand = (70 71 26 18 20 39 40)
block expand = (11 16 24 25 23 54)
block expand = (21 22 41 42 43 44 45 46 47 48 49 50 51 52 53)
Pedigree:
1 60 61 m 0 0 n
2 62 63 f 0 0 n
3 1 2 f 0 0 n
4 1 2 m 0 0 n
5 0 0 m 0 0 n
6 0 0 f 0 0 n
7 3 5 f 0 0 n
8 3 5 f 0 0 n
9 4 6 m 1 2 n
10 4 6 m 0 0 n
11 4 6 m 0 0 n
12 0 0 m 0 0 n
13 0 0 f 3 4 n
14 0 0 m 0 0 n
15 0 0 f 0 0 n
16 0 0 f 0 0 n
17 7 12 m 1 3 n
18 7 12 m 1 2 n
19 9 13 f 1 4 n
20 9 13 f 1 4 n
21 9 13 f 1 3 n
22 8 14 m 1 4 n
23 10 15 f 1 4 n
24 11 16 m 1 3 n
25 72 73 m 2 3 n
26 70 71 f 2 4 n
27 17 19 u 1 1 a
28 17 19 u 1 1 a
29 17 19 u 1 3 n
30 17 19 u 1 3 n
31 17 19 u 1 3 n
32 17 19 u 1 3 n
33 17 19 u 1 4 n
34 17 19 u 1 4 n
35 17 19 u 1 4 n
36 17 19 u 3 4 n
37 17 19 u 3 4 n
38 17 19 u 3 4 n
39 18 20 u 1 1 a
40 18 20 u 2 4 n
41 21 22 u 1 1 a
42 21 22 u 1 3 n
43 21 22 u 1 3 n
44 21 22 u 1 3 n
45 21 22 u 1 3 n
46 21 22 u 1 4 n
```

```
47 21 22 u 1 4 n
48 21 22 u 1 4 n
49 21 22 u 1 4 n
50 21 22 u 3 4 n
51 21 22 u 3 4 n
52 21 22 u 3 4 n
53 21 22 u 3 4 n
54 25 23 m 1 3 n
55 24 26 f 1 4 n
56 54 55 u 1 1 a
57 54 55 u 1 3 n
58 54 55 u 1 4 n
59 54 55 u 3 4 n
60 0 0 m 0 0 n
61 0 0 f 0 0 n
62 0 0 m 0 0 n
63 0 0 f 0 0 n
64 60 61 m 0 0 n
65 62 63 f 0 0 n
66 0 0 m 0 0 n
67 64 65 f 0 0 n
68 1 2 m 0 0 n
69 0 0 f 0 0 n
70 68 69 m 0 0 n
71 66 67 f 0 0 n
72 3 5 m 0 0 n
73 0 0 f 0 0 n
```

A.4 Availability of Software

The *block* and *theta* software programs can be obtained by anonymous ftp from

ftp.cs.auc.dk/pub/packages/block/current

for the following architectures: Linux, PC-DOS, SPARC-Solaris, RS-6000-AIX3.2.5, SGI Irix64, and DEC Alpha. The package can also be obtained from the associated homepage :

http://www.cs.auc.dk/~claus/block.html

Appendix B

Implementational Aspects

This appendix will discuss some implementational aspects of the blocking Gibbs software presented in Appendix A. The nature of these aspects will be such that they are useful in other applications doing similar computations.

We will discuss the use of the optimized programming structure, the *heap*, in optimization of problems encountered in the blocking Gibbs software, i.e., triangulation and block selection.

B.1 Heaps

A heap is a data structure with its elements partially ordered such that finding either the minimum or the maximum (but not both) of the elements is computationally inexpensive (independent of the number of elements), while both adding a new item and finding each subsequent smallest/largest element can be done in $O(\log n)$ time, where n is the number of elements.

Formally, a heap is a binary tree with a key in each node, such that all the leaves of the tree appear on the two lowest levels; all leaves on the lowest level occur to the left and all levels, except possibly the lowest, are filled. Further, the key in the root is at least as large (or as small) as the keys in its children (if any), and the left and the right subtrees (if they exist) are again heaps.

Heaps are often implemented as one-dimensional arrays. If the goal is to find the minimum quickly, the invariant for heaps is :

`heap[i] <= heap[2*i]` and `heap[i] <= heap[2*i+1]` for all `i`,

where `heap[i]` denotes the key of the `i`th node, `heap[1]` being the first.

Further, changing the key of an existing node can also be done in $O(\log n)$ time.

B.2 Using Heaps when Triangulating

When triangulating with one of the heuristic methods described in Section 4.5, one needs to maintain a list of the variables sorted wrt. one of the criteria minimum fill-in links, minimum clique size, minimum clique weight, or minimum fill-in weight. At each step in the triangulation, the variable that best fulfills the criterion in use, e.g., with the minimum number of fill-in links, is selected, and eliminated from the moralized graph. Then, the number of fill-in links has to be recomputed for each

of the neighbouring variables, and the list must be updated and re-sorted to reflect this.

If an ordinary linked list is used for this, the operation for re-sorting the list after changing the keys of some of the variables takes $O(n \log n)$ if, e.g., quick-sort is used.

As this operation will be performed once for each variable, the algorithm will be done in $O(n^2 \log n)$ time when using a linked list.

On the other hand, if a heap is used, the algorithm will be done in only $O(n \log n)$ time, as the re-sorting of the heap after updating some variables can be done in $O(\log n)$ time.

This gain in time is very significant when large networks are triangulated. For instance, for the network in Figure 8.1 on page 49 with 1614 variables, $n^2 \log n \approx$ 19,000,000 and $n \log n \approx 12{,}000$.

B.3 Using Heaps when Selecting Blocks

For the block selection method 1 described in Section 9.2.1, it is necessary to maintain a list of the variables sorted wrt. their reductions in storage requirements. When a variable is selected for conditioning in a block, some variables need to get their reduction in storage requirements updated, and thus the list of variables needs to be re-sorted.

Again, like with the triangulation, this list can be efficiently implemented with a heap structure, allowing us to obtain a much lower complexity of $O(n \log n)$.

Bilag C

Dansk resumé

Dette resumé af afhandlingen er en oversættelse af indledningen til dansk.

C.1 Inferens i sandsynlighedsbaserede ekspert systemer

Ræsonnement i domæner med usikker viden er en meget almindelig aktivitet for mennesker. I mange år har man forsøgt at formalisere denne aktivitet og udvikle metoder til effektivt at håndtere komplekse problemer med usikker viden på computere. Nogle af de første forsøg på at håndtere disse generelle problemer førte til udviklingen af regel-baserede systemer og fuzzy logik. Selv om regel-baserede systemer i begyndelsen ikke tilbød metoder til håndtering af usikker information, så blev en metode senere udviklet af Shortliffe og Buchanan (1975), der benyttede sig af de såkaldte "certainty factors". Fuzzy logik blev specifikt udviklet til at håndtere de mange problemer med indbygget usikkerhed, se (Zadeh 1983).

Der opstår dog problemer for disse metoder, når flere usikre oplysninger skal kombineres. De er ikke generelt i stand til at håndtere usikkerhed på en fuldstændig konsistent måde. I løbet af de sidste par årtier er der blevet udviklet en bedre måde at håndtere usikkerhed på, hvor usikkerheden bliver modelleret vha. sandsynlighedsteori i grafiske modeller. Dvs. at usikkerheden (nu sandsynligheder af variables tilstande) kan kombineres vha. Bayes' ligning, hvilket giver en konsistent fremgangsmåde. Problemdomænerne modelleres så i de såkaldte grafiske modeller, hvor knuder i grafen repræsenterer variablene, og afhængigheder mellem variablene repræsenteres af forbindelser (kanter) mellem knuderne (Darroch et al. 1980, Wermuth og Lauritzen 1983). En anden fordel ved denne fremgangsmåde er, at de betingede uafhængigheder, som næsten altid er tilstede mellem mange variable, kan udnyttes og bruges til at udvikle effektive inferens metoder.

Disse sandsynlighedsbaserede grafiske modeller har mange navne, men de er oftest blevet kaldt *Bayesianske net* pga. brugen af Bayes' ligning, og det er også dette navn, der vil blive brugt i afhandlingen. De bruges oftest i sammenhæng med ekspert systemer, og de kaldes derfor også *sandsynlighedsbaserede ekspert systemer*. Da Bayesianske net dog modellerer det problem-specifikke domæne, og ikke eksperten (sådan som f.eks. regel-baserede systemer), fremhæver vi denne forskel ved at kalde dem *beslutningsstøtte systemer*, da de kun skal tjene til beslutningsstøtte og ikke skal repræsentere pålidelige eksperter.

I de senere år er mange metoder til inferens i Bayesianske net blevet foreslået. De falder i to kategorier, eksakte og stokastiske. Eksakte metoder har det generelle

problem at inferens i Bayesianske net er NP-hårdt (Cooper 1990), dvs. der er ofte en eksponentiel sammenhæng mellem antal variable og beregningskompleksiteten. Dette begrænser mængden af problemer, for hvilke eksakt inferens er mulig. Derfor er det nødvendigt at anvende stokastiske metoder for problemer, hvor eksakt inferens ikke er mulig.

Til eksakt inferens i Bayesianske net er der blevet udviklet metoder til at håndtere enkeltvis forbundne net (dvs. træer) af Kim og Pearl (1983). Desværre udgør enkeltvis forbundne net kun en lille del af de interessante problemer i den virkelige verden, så det var vigtigt at udvikle metoder til eksakt inferens i generelle net. Metoderne indenfor "*clustering*" (klynge) kategorien har været de mest succesfulde (Pearl 1986b, Lauritzen og Spiegelhalter 1988, Shenoy og Shafer 1990, Lauritzen 1992). Grundlæggende transformerer de nettet til et træ ved at samle variable i klynger (sæt), og udfører inferens i dette såkaldte "klynge træ" ved at sende beskeder indeholdende sandsynlighedsbaseret information om forskellige variable rundt. Det er blevet vist af Shachter et al. (1991) at alle disse "clustering" metoder til eksakt sandsynlighedsbaseret inferens i Bayesianske net fundamentalt set er ækvivalente.

De eksakte metoder er dog ude af stand til at håndtere mange problemer i den virkelige verden, da klyngerne ofte må indeholde mange variable for at transformere nettet til et træ. Da hver klynge indeholder en sandsynlighedstabel, som indeholder sandsynligheden af enhver kombination af værdierne af dens variable, kan disse tabeller vokse til astronomiske størrelser. *Conditioning* metoden af Pearl (1986a) forsøger at reducere størrelserne af klyngetabellerne ved grundlæggende at transformere pladsforbrug til tid. Metoden gør dog kun eksakt inferens mulig i en lidt større mængde af net.

Hvis man vil håndtere problemer af generel størrelse og kompleksitet, er man nødt til at anvende stokastiske metoder, med hvilke det normalt er muligt at opnå resultater med en ønsket præcision. Stokastiske metoder kaldes også Monte Carlo metoder, pga. deres tilfældige natur. Monte Carlo metoder sampler generelt fra den ønskede sandsynlighedsfordeling og tager så gennemsnittet af udfaldene for at approksimere de marginale fordelinger af variable. Generelt er det ikke muligt at producere uafhængige udfald fra den ønskede fordeling, men udfaldene behøver ikke nødvendigvis at være uafhængige. Markov chain Monte Carlo (MCMC) metoder udgør en delmængde af Monte Carlo metoderne, som producerer afhængige udfald ved at benytte en Markov kæde, designet så dens ligevægtsfordeling er lig med den fordeling, vi gerne vil estimere. Udfaldene kan være mere eller mindre afhængige givet den anvendte MCMC metode. MCMC metoder har vist sig som meget praktiske og effektive metoder til inferens i generelle statistiske modeller, og specielt, Bayesianske net. Den første MCMC metode, Metropolis algoritmen, blev foreslået af Metropolis et al. (1953), og denne metode blev senere generaliseret til Metropolis-Hastings algoritmen af Hastings (1970). Gibbs sampleren er et særtilfælde af Metropolis-Hastings algoritmen, men er blevet den mest populære MCMC metode pga. dens intuitive forklaring og simple implementation. Den er dog ikke nødvendigvis det bedste valg i det generelle tilfælde. Interessant nok havde Gibbs sampleren været kendt i den statistiske fysiks litteratur i flere år som *heat bath algoritmen* (Creutz 1979, Ripley 1979) før den blev foreslået af Geman og Geman (1984) til billedbehandling og derigennem blev kendt i bredere kredse.

Der er et antal betingelser, som skal være opfyldte, for at Gibbs sampleren virker. Normalt er den mest kritiske af disse, at den underliggende Markov kæde skal være irreducibel. Hvis dette ikke er opfyldt, indeholder udfaldsrummet for kæden delmængder, der ikke er forbundne (også kaldet *ikke-kommunikerende sæt*), og som Gibbs sampleren derfor ikke er i stand til at bevæge sig imellem. Gibbs sampleren vil derfor aldrig kunne estimere den ønskede fordeling korrekt. Selv om kæden er

irreducibel, kan den også være næsten reducibel. Dette kan få Gibbs sampleren til at bevæge sig så langsomt rundt i udfaldsrummet, at det kræver et astronomisk antal iterationer at komme omkring (det kaldes også, at sampleren *mixer* langsomt).

Til problemer, der udviser disse besværlige egenskaber, er der blevet foreslået flere avancerede MCMC metoder. Disse falder i to kategorier, dem, der forstørrer udfaldsrummet, og dem, der ikke gør det. Blandt metoderne, der forstørrer udfaldsrummet, er en af de mest lovende den såkaldte *simulated tempering* metode (Geyer 1991, Marinari og Parisi 1992, Geyer og Thompson 1995). Simulated tempering vedligeholder et hierarki af Markov kæder, vekslende fra den "varmeste" kæde, som kan bruges til at producere uafhængige udfald, til den "koldeste" kæde, som har den ønskede fordeling som ligevægtsfordeling, og evt. kan være reducibel og/eller mixe meget langsomt. Simulated tempering kan således løse problemer med reducibilitet og langsom mixing i det generelle tilfælde. For at virke i praksis kræver metoden dog konstruktionen af flere "opvarmede" kæder, hvilket kan vise sig at være svært. Desuden kan det være en stor beregningsmæssig byrde at køre de relativt mange kæder samtidig.

En metode, som ikke forstørrer udfaldsrummet, og derfor undgår at producere udfald, som ikke kan anvendes til at estimere den ønskede distribution, er *blocking Gibbs sampling*, som oprindeligt blev præsenteret i Jensen et al. (1995). Denne algoritme kombinerer en bestemt "clustering" metode til eksakt inferens, den såkaldte *junction-træ propageringsmetode* (Lauritzen og Spiegelhalter 1988, Jensen et al. 1990) med Gibbs sampleren. Algoritmen gør det muligt at implementere den generelle Gibbs sampler, hvor komponenterne består af mange variable i stedet for kun en enkelt. Det gør det muligt at opdatere mange variable på én gang (i praksis, mere end 90%), og dette løser ofte problemer med reducibilitet og langsom mixing. I den første del af afhandlingen behandles alle emner, der generelt har med blocking Gibbs sampleren at gøre.

C.2 Applikationer indenfor genetik

Et længe eksisterende problem indenfor genetik, er opdateringen af sandsynligheder i stamtræer, dvs. *stamtræsanalyse*. I stamtræsanalyse repræsenterer variablene f.eks. genotyperne og fænotyperne af medlemmer af stamtræet, og nogle af disse kan være observerede. Til sandsynlighedsbaseret inferens i stamtræer er den eksakte metode *peeling* blevet udviklet af Cannings et al. (1978). Denne metode er endnu en variation af de generelle "clustering" metoder, som interessant nok blev udviklet flere år før de første eksakte metoder for Bayesianske net.

Som de andre eksakte metoder, lider også peeling af problemer med ekstremt store sandsynlighedstabeller, når stamtræet bliver komplekst. Dette problem er indtil nu blevet håndteret på forskellige måder indenfor genetik. For det første har der været en tendens til at undgå komplekse stamtræer og fokusere på mindre stamtræer, f.eks. ved at benytte *sib-pair metoden* af Penrose (1935), som senere blev udviklet til *affected* sib-pair metoden, som kun betragter små stamtræer bestående af enkelte familier. Dernæst har man også forsøgt at undgå problemet ved kun at anvende peeling på stamtræer med meget få løkker, dvs. de er næsten enkeltvis forbundne. En udvidelse af peeling kombineret med conditioning er blevet implementeret i de populære edb-programmer `LINKAGE` (Lathrop og Lalouel 1984, Lathrop et al. 1985) and `FASTLINK` (Cottingham Jr. et al. 1993, Schäffer et al. 1994). Denne metode er kun i stand til at håndtere stamtræer med et meget lavt antal løkker.

Når MCMC metoder anvendes til stamtræsanalyse, opstår der ofte problemer med reducibilitet, pga. de betingede sandsynlighedsfordelinger der er tilstede her, f.eks. penetrans sandsynlighederne. Det er blevet vist, at kun i tilfældet med et lokus

med to alleler, kan irreducibilitet (næsten altid) garanteres (Sheehan og Thomas 1993). Når et lokus med mere end to alleler skal håndteres, kan den underliggende kæde være irreducibel eller reducibel. Der eksisterer ingen generel metode til at fastslå, hvorvidt en Gibbs sampler vil være irreducibel, når den anvendes på et specifikt problem i stamtræsanalyse.

Simulated tempering er også blevet anvendt til stamtræsanalyse og er lovende indenfor området. Også *sequential imputation* (Kong et al. 1993, Irwin et al. 1994) til at foretage *linkage analysis* (estimering af afstanden mellem to gener) virker lovende. Denne metode er ikke iterativ, men håndterer et lokus ad gangen vha. peeling, og er derfor kun i stand til at håndtere stamtræer med meget få løkker.

Blocking Gibbs sampleren er også blevet brugt succesfuldt til stamtræsanalyse. Jensen et al. (1995) anvendte blocking Gibbs sampleren til simpel stamtræsanalyse, og Jensen og Kong (1996) anvendte den til to-punkts linkage analysis. Andel del af afhandlingen behandler således alle emner forbundet med disse genetiske applikationer, og beskriver bl.a. resultaterne fra de to artikler.

C.3 Overblik over afhandlingen

Første del af afhandlingen omhandler alle emner, der er relateret til definitionen og aspekter af blocking Gibbs sampleren. Først, i kapitel 2 gives der en specifik introduktion til denne del af afhandlingen. I kapitel 3 bliver intuitionen og teorien bag Bayesianske net opridset, fulgt af en beskrivelse af junction-træ metoden til eksakt inferens i Bayesianske net i kapitel 4. I kapitel 5 bliver teorien for Markov kæder og MCMC metoder præsenteret, inkluderende definition og andre aspekter af Gibbs sampleren. Så, i kapitel 6 bliver blocking Gibbs sampleren defineret, fulgt af et bevis på dens irreducibilitet i kapitel 7. I kapitel 8 præsenteres en praktisk anvendelig metode til at finde en legal start konfiguration for en Gibbs sampler. Metoden er baseret på conditioning af Pearl (1986a) og vises at være deterministisk i praksis, når den anvendes på eksempler fra genetik, men dette bevises ikke i det generelle tilfælde. I kapitel 9 bliver forskellige metoder til at udvælge blokke for blocking Gibbs sampleren præsenteret. Blok udvælgelses metoderne er hovedsageligt baseret på pladsforbruget af blokkene. Det forsøges at lade blokkene indeholde så mange variable som muligt, og på samme tid holde pladsforbruget på et rimeligt niveau. Blokkene udvælges dog også efter andre kriterier, så som at sikre irreducibilitet af Markov kæden. I kapitel 10 præsenteres forward sampling algoritmen, som anvendes på de såkaldte *barren* variable, som er ikke observerede variable uden observerede efterkommere. Det er en fordel at forward sample disse variable, da uafhængige samples så kan opnås for dem, hvilket resulterer i hurtigere konvergens mod den ønskede fordeling.

Andel del af afhandlingen omhandler alle emner relateret til de genetiske applikationer af blocking Gibbs sampleren. Først, i kapitel 11 gives en specifik introduktion til denne del af afhandlingen. I kapitel 13 forklares det, hvordan stamtræer kan repræsenteres vha. Bayesianske net. Tre repræsentationer gives; i den første repræsenteres genotyper af variable, i den anden repræsenteres enkelte gener af variable, og i den sidste inkluderes der information om rekombination mellem flere loki. I kapitel 14 introduceres der forskellige problemer, der kan fremkalde reducibilitet, og det forklares, hvordan de fleste af dem kan håndteres med blocking Gibbs sampleren. Af og til, og i helt anderledes situationer, opstår der et mere alvorligt problem, når kæden bliver næsten reducibel, hvilket kan forårsage meget langsom mixing. I kapitel 15 præsenteres der to almindelige situationer i stamtræsanalyse, som forårsager en næsten reducibel Gibbs sampler, samt metoder til at håndtere dem med blocking Gibbs sampleren. I kapitel 16 bliver blocking Gibbs sampleren

benyttet til simpel stamtræsanalyse i et stamtræ bestående af 20.000 svin, i et forsøg på at estimere de marginale sandsynligheder af svinenes genotyper givet de observerede fænotyper af en del af svinene. Dernæst, i kapitel 17 bliver blocking Gibbs sampleren benyttet til at foretage to-punkts linkage analysis i et komplekst stamtræ for mennesker, hvoraf nogle har en sjælden hjertesygdom. I kapitel 18 analyseres en algoritme af Lin et al. (1994) til at identificere de ikke-kommunikerende sæt af en MCMC metode anvendt til stamtræsanalyse. Desuden tages der begyndende skridt mod en mere generel algoritme.

Til slut, i kapitel 19 bliver de to dele af afhandlingen samlet i en endelig konklusion med en diskussion og forslag til fremtidig forskning.

Da studiet dels har været teoretisk og praktisk, er der også blevet udviklet en del edb-programmer. Specielt er der blevet udviklet en udgave af blocking Gibbs sampleren til simpel stamtræsanalyse og linkage analysis. Manualen til denne kan forefindes i appendiks A.

Bibliography

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley. Reading, Massachusetts.

Amit, Y., Grenander, U. and Piccioni, M. (1991). Structural image restoration through deformable templates, *Journal of American Statistical Association* **86**: 82–99.

Andersen, S. K., Olesen, K. G., Jensen, F. V. and Jensen, F. (1989). HUGIN — a shell for building Bayesian belief universes for expert systems, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI.

Arnborg, S., Corneil, D. G. and Proskurowski, A. (1987). Complexity of finding embeddings in a k-tree, *SIAM Journal on Algebraic and Discrete Methods* **8**: 277–284.

Bennett, C. H. (1976). Efficient estimation of free energy differences from Monte Carlo data, *Journal of Computational Physics* **22**: 245–268.

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems, *Journal of the Royal Statistical Society, Series B* **36**: 192–236.

Besag, J., Green, P., Higdon, D. and Mengersen, K. (1995). Bayesian computation and stochastic systems, *Statistical Science* **10**(1): 3–66.

Brooks, S. and Roberts, G. O. (1995). Diagnosing convergence of Markov chain Monte Carlo algorithms, *Technical report*, Statistical Laboratory, University of Cambridge, England.

Cannings, C., Thompson, E. A. and Skolnick, M. H. (1976). The recursive derivation of likelihoods on complex pedigrees, *Advances in Applied Probability* **8**: 622–625.

Cannings, C., Thompson, E. A. and Skolnick, M. H. (1978). Probability functions on complex pedigrees, *Advances in Applied Probability* **10**: 26–61.

Charniak, E. (1991). Bayesian networks without tears, *AI Magazine* **12**(4): 50–63.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks, *Artificial Intelligence* **42**: 393–405.

Cottingham Jr., R. W., Idury, R. M. and Schäffer, A. A. (1993). Faster sequential genetic linkage computations, *American Journal of Human Genetics* **53**: 252–263.

Cowles, M. K. and Carlin, B. P. (1995). Markov chain Monte Carlo convergence diagnostics: A comparative review, *Technical report*, Division of Biostatistics, School of Public Health, University of Minnesota, USA.

Creutz, M. (1979). Confinement and the critical dimensionality of space-time, *Physical Review Letters* **43**: 553–556.

Darroch, J. N., Lauritzen, S. L. and Speed, T. P. (1980). Markov-fields and log-linear models for contingency tables, *The Annals of Statistics* **8**: 522–539.

Dawid, A. P. (1992). Applications of a general propagation algorithm for probabilistic expert systems, *Statistics and Computing* **2**: 25–36.

DeGroot, M. H. (1986). *Probability and Statistics.*, Addison-Wesley Publishing Company.

Edwards, A. W. F. (1968). Simulation studies of genealogies, *Heredity* p. 628. Abstract of a presented paper.

Elston, R. C. and Stewart, J. (1971). A general model for the genetic analysis of pedigree data, *Human Heredity* **21**: 523–542.

Feller, W. (1968). *An Introduction to Probability Theory and Its Applications*, 3rd edn, Wiley & Sons, Inc.

Fischer, L. P. (1990). *Reference Manual for the HUGIN Application Program Interface*, 1st edn, Hugin Expert A/S.

Fishman, G. S. (1996). Coordinate selection rules for Gibbs sampling, *The Annals of Applied Probability* **6**(2): 444–465.

Gelfand, A. E. and Smith, A. F. M. (1990). Sampling-based approaches to calculating marginal densities, *Journal of American Statistical Association* **85**(410): 398–409.

Gelman, A. and Meng, X.-L. (1994). Path sampling for computing normalizing constants: Indentities and theory, *Technical Report 376*, Department of Statistics, University of Chicago.

Gelman, A. and Meng, X.-L. (1996). Simulating normalizing constants: From importance sampling to bridge sampling to path sampling, *Technical report*, Department of Statistics, Columbia University, New York.

Gelman, A. and Rubin, D. B. (1992). Inference from iterative simulation using single and multiple sequences (with discussion), *Statistical Science* **7**: 457–511.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**(6): 721–741.

Geyer, C. J. (1991). Markov chain Monte Carlo maximum likelihood, *Computing Science and Statistics, Proceedings of the 23rd Symposium on the Interface*, pp. 156–163.

Geyer, C. J. (1992). Practical Markov Chain Monte Carlo, *Statistical Science* **7**(4): 473–511.

Geyer, C. J. and Thompson, E. A. (1995). Annealing Markov chain Monte Carlo with applications to ancestral inference, *Journal of American Statistical Association* **90**(431): 909–920.

Gibbons, A. (1985). *Algorithmic Graph Theory*, Cambridge University Press.

Gilberg, A., Gilberg, L., Gilberg, R. and Holm, M. (1978). *Polar Eskimo Genealogy*, Vol. 203 No. 4 of *Meddelelser om Grønland*, Nyt Nordisk Forlag, Arnold Busck, København.

Gilks, W. R., Richardson, S. and Spiegelhalter, D. J. (eds) (1996). *Markov Chain Monte Carlo in Practice*, Chapman & Hall, London, UK.

Hansen, B. and Pedersen, C. B. (1994). *Analysing complex pedigrees using gibbs sampling - a theoretical and empirical investigation*, Technical report, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Hartl, D. L. (1988). *A Primer of Population Genetics*, Sinauer Associates, Inc., Sunderland, Massachusetts.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications, *Biometrika* **57**(1): 97–109.

Heath, S. C. (1997). Generating consistent genotypic configurations for multi-allelic loci and large complex pedigrees, Unpublished manuscript. Submitted to Human Heredity.

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling, *Uncertainty in Artificial Intelligence 2*, North-Holland, Amsterdam.

Hrycej, T. (1990). Gibbs sampling in Bayesian networks, *Artificial Intelligence* **46**: 351–363.

Irwin, M., Cox, N. and Kong, A. (1994). Sequential imputation for multilocus linkage analysis, *Proceedings of the National Academy of Sciences of the USA*, pp. 11684–11688.

Janss, L. L. G., Thompson, R. and Van Arendonk, J. A. M. (1995). Application of Gibbs sampling for inference in a mixed major gene-polygenic inheritance model in animal populations, *Theoretical and Applied Genetics* **91**(6/7): 1137–1147.

Jensen, C. S. (1996a). A simple method for finding a legal configuration in complex Bayesian networks, *Technical Report R-96-2046*, Department of Computer Science, Aalborg University, Denmark.

Jensen, C. S. and Kong, A. (1996). Blocking gibbs sampling for linkage analysis in large pedigrees with many loops, *Technical Report R-96-2048*, Department of Computer Science, Aalborg University, Denmark.

Jensen, C. S., Kong, A. and Kjærulff, U. (1995). Blocking-Gibbs sampling in very large probabilistic expert systems, *International Journal of Human-Computer Studies* **42**: 647–666.

Jensen, C. S. and Sheehan, N. (1997). Problems with the determination of the noncommunicating classes for MCMC applications in pedigree analysis, *Technical Report R-97-5004*, Department of Computer Science, Aalborg University, Denmark.

Jensen, F. V. (1988). Junction trees and decomposable hypergraphs, *Research report*, Judex Datasystemer A/S, Aalborg, Denmark.

Jensen, F. V. (1996b). *An Introduction to Bayesian Networks*, UCL Press, University College Limited.

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

Kalos, M. H. and Whitlock, P. A. (1986). *Monte Carlo Methods, Volume I: Basics*, Wiley & Sons, Inc.

Kim, J. H. and Pearl, J. (1983). A computational model for causal and diagnostic reasoning in inference systems, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 190–193.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by simulated annealing, *Science* **220**: 671–680.

Kjærulff, U. (1990). Triangulation of graphs — algorithms giving small total state space, *Technical Report R 90-09*, Department of Mathematics and Computer Science, Aalborg University, Denmark.

Kjærulff, U. (1993). Approximation of Bayesian networks through edge removals, *Research Report IR-93-2007*, Department of Computer Science, Aalborg University, Denmark.

Kong, A. (1989). Monte Carlo methods for approximating linkage likelihoods, *Technical Report 254*, University of Chicago, Department of Statistics, Chicago, Illinous 60637, USA.

Kong, A. (1991). Efficient methods for computing linkage likelihoods of recessive diseases in inbred pedigrees, *Genetic Epidemiology* **8**: 81–103.

Kong, A., Cox, N., Frigge, M. and Irwin, M. (1993). Sequential imputation and multipoint linkage analysis, *Genetic Epidemiology* **10**: 483–488.

Lange, K. and Elston, R. C. (1975). Extensions to pedigree analysis. I. Likelihood calculations for simple and complex pedigrees, *Human Heredity* **25**: 95–105.

Lathrop, G. M. and Lalouel, J.-M. (1984). Easy calculations of lod scores and genetic risks on small computers, *American Journal of Human Genetics* **36**: 460–465.

Lathrop, G. M., Lalouel, J.-M., Julier, C. and Ott, J. (1985). Multilocus linkage analysis in humans: Detection of linkage and estimation of recombination, *American Journal of Human Genetics* **37**: 482–498.

Lauritzen, S. L. (1992). Propagation of probabilities, means and variances in mixed graphical association models, *Journal of American Statistical Association* **87**(420): 1098–1108.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N. and Leimer, H.-G. (1990). Independence properties of directed Markov fields, *Networks* **20**(5): 491–505. Special Issue on Influence Diagrams.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems, *Journal of the Royal Statistical Society, Series B* **50**(2): 157–224.

Lin, S. (1993). *Markov Chain Monte Carlo Estimates of Probabilities on Complex Structures*, PhD thesis, University of Washington, Seattle.

Lin, S. (1995). A scheme for constructing an irreducible Markov chain for pedigree data, *Biometrics* **51**: 318–322.

Lin, S. (1996). Multipoint linkage analysis via Metropolis jumping kernels, *Biometrics* **52**(4): 1417–1427.

Lin, S., Thompson, E. and Wijsman, E. (1993). Achieving irreducibility of the Markov chain Monte Carlo method applied to pedigree data, *IMA Journal of Mathematics Applied in Medicine & Biology* **10**: 1–17.

Lin, S., Thompson, E. and Wijsman, E. (1994). Finding non-communicating sets for Markov chain Monte Carlo estimations on pedigrees, *American Journal of Human Genetics* **54**: 695–704.

Liu, J. S., Wong, W. H. and Kong, A. (1994). Covariance structure of the Gibbs sampler with applications to the comparisons of estimators and augmentation schemes, *Biometrika* **81**(1): 27–41.

MacCluer, J. W., Vandeburg, J. L., Read, B. and Ryder, O. A. (1986). Pedigree analysis by computer simulation, *Zoo Biology* **5**: 147–160.

Marinari, E. and Parisi, G. (1992). Simulated tempering: A new Monte Carlo scheme, *Europhysics Letters* **19**: 451–458.

Mendel, G. (1866). Versuche über Pflanzen-Hybriden, *Verh. Naturforschung Ver. Brünn* **4**: 3–47.

Meng, X.-L. and Schilling, S. (1996). Fitting full-information item factor models and an empirical investigation of bridge sampling, *Journal of American Statistical Association* **91**(435): 1254–1267.

Meng, X.-L. and Wong, W. H. (1996). Simulating ratios of normalizing constants via a simple identity: A theoretical exploration, *Statistica Sinica* **6**(4): 831–860.

Mengersen, K. L. and Tweedie, R. L. (1996). Rates of convergence of the Hastings and Metropolis algorithms, *The Annals of Statistics* **24**(1): 101–121.

Metropolis, N., Rosenbluth, A., Rosenbluth, M. and Teller, A. (1953). Equations of state calculations by fast computing machines, *Journal of Chemistry and Physics* **21**: 1087–1091.

Morton, N. E. (1955). Sequential tests for the detection of linkage, *American Journal of Human Genetics* **7**: 277–318.

Neal, R. M. (1993). Probabilistic inference using Markov chain Monte Carlo methods, *Technical Report CRG-TR-93-1*, Department of Computer Science, University of Toronto.

Ott, J. (1989). Computer-simulation methods in human linkage analysis, *Proceedings of the National Academy of Sciences of the USA* **86**: 4175–4178.

Ott, J. (1991). *Analysis of Human Genetic Linkage*, John Hopkins University Press Ltd., London.

Pearl, J. (1986a). A constraint-propagation approach to probabilistic reasoning, *in* L. M. Kanal and J. Lemmer (eds), *Uncertainty in Artificial Intelligence*, North-Holland, Amsterdam, pp. 357–370.

Pearl, J. (1986b). Fusion, propagation, and structuring in belief networks, *Artificial Intelligence* **29**: 241–288.

Pearl, J. (1987). Evidential reasoning using stochastic simulation of causal models, *Artificial Intelligence* **32**: 245–257.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Series in Representation and Reasoning, Morgan Kaufmann Publishers, Inc.

Penrose, L. S. (1935). The detection of autosomal linkage in data which consists of pairs of brothers and sisters of unspecified parentage, *Ann. Eugen.* **6**: 133–138.

Ploughman, L. M. and Boehnke, M. (1989). Estimating the power of a proposed linkage study for a complex genetic trait, *American Journal of Human Genetics* **44**: 543–551.

Polson, N. G. (1995). Convergence of Markov chain Monte Carlo algorithms, *in* J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith (eds), *Bayesian Statistics 5*, Oxford: Oxford University Press.

Ripley, B. D. (1979). Simulating spatial patterns: Dependent samples from a multivariate density, *Journal of the Royal Statistical Society, Series C* **28**: 109–112.

Roberts, G. O. and Sahu, S. K. (1997). Updating schemes, correlation structure, blocking and parameterization for the Gibbs sampler, *Journal of the Royal Statistical Society, Series B* **59**(2): 291–317.

Roberts, G. O. and Tweedie, R. L. (1995). Geometric convergence and central limit theorems for multidimensional Hastings-Metropolis algorithms, *Technical report*, Department of Statistics, Colorado State University.

Rosenthal, J. S. (1992). Rates of convergence for the Gibbs sampler and other Markov chains, *Technical report*, Department of Mathematics, Harvard University.

Schäffer, A. A., Gupta, S. K., Shriram, K. and Cottingham Jr., R. W. (1994). Avoiding recomputation in linkage analysis, *Human Heredity* **44**: 225–237.

Shachter, R. D., Andersen, S. K. and Szolovits, P. (1991). The equivalence of exact methods for probabilistic inference on belief networks, *Technical report*, Department of Engineering-Economic Systems, Stanford University.

Sheehan, N. (1990). *Genetic Restoration on Complex Pedigrees*, PhD thesis, University of Washington.

Sheehan, N. (1992). Sampling genotypes on complex pedigrees with phenotypic constraints: the origin of the B allele among the Polar Eskimos, *IMA Journal of Mathematics Applied in Medicine and Biology* **9**: 1–18.

Sheehan, N. and Thomas, A. (1993). On the irreducibility of a Markov chain defined on a space of genotype configurations by a sampling scheme, *Biometrics* **49**: 163–175.

Shenoy, P. P. and Shafer, G. R. (1990). Axioms for probability and belief-function propagation, *in* R. D. Shachter, T. S. Levitt, L. N. Kanal and J. F. Lemmer (eds), *Uncertainty in Artificial Intelligence 4*, Elsevier Science Publishers B. V. (North-Holland), Amsterdam, pp. 169–198.

Shortliffe, E. H. and Buchanan, B. (1975). A model for inexact reasoning in medicine, *Mathematical Biosciences* **23**: 351–379.

Smith, A. F. M. and Roberts, G. O. (1993). Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods, *Journal of the Royal Statistical Society, Series B* **55**(1): 5–23.

Spiegelhalter, D. J., Dawid, A. P., Lauritzen, S. L. and Cowell, R. G. (1993). Bayesian analysis in expert systems (with discussion), *Statistical Science* **8**: 219–247 and 247–283.

Swendsen, R. H. and Wang, J. S. (1987). Nonuniversal critical dynamics in monte carlo simulations, *Phys. Rev. Lett.* **58**: 86–88.

Tarjan, R. E. and Yannakakis, M. (1984). Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs, *SIAM Journal on Computing* **13**: 566–579.

Thomas, A. (1988). Drawing pedigrees, *IMA Journal of Mathematics Applied in Medicine & Biology* **5**: 201–213.

Thomas, A., Spiegelhalter, D. J. and Gilks, W. R. (1992). BUGS: A program to perform Bayesian inference using Gibbs sampling, *in* J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith (eds), *Bayesian Statistics 4*, Clarendon Press, Oxford, UK, pp. 837–842.

Thompson, E. A. (1986). *Pedigree Analysis in Human Genetics.*, John Hopkins University Press.

Tierney, L. (1994). Markov chains for exploring posterior distributions (with discussion), *Annals of Statistics* **22**: 1701–1762.

Wermuth, N. and Lauritzen, S. L. (1983). Graphical and recursive models for contingency tables, *Biometrika* **70**(3): 537–552.

Wright, S. and McPhee, H. C. (1925). An approximate method of calculating coefficients of inbreeding and relationship from livestock pedigrees, *Journal of Agricultural Research* **31**: 377–383.

Yiannoutsos, C. T. and Gelfand, A. E. (1994). Simulation approaches for calculations in directed graphical models, *in* S. Gupta and J. Berger (eds), *Statistical Decision Theory and Related Topics*, Vol. V, Springer-Verlag, N.Y., pp. 441–452.

York, J. (1992). Use of the Gibbs sampler in expert systems, *Artificial Intelligence* **56**: 115–130.

Zadeh, L. A. (1983). The role of fuzzy logic in the management of uncertainty in expert systems, *Fuzzy Sets and Systems* **11**: 199–228.

Index