Markov Chain Monte Carlo Methods for Exact Tests in Contingency Tables

KHEDRI, SHILER

Markov Chain Monte Carlo Methods for Exact Tests in Contingency Tables

Shiler Khedri

A thesis presented for the degree of
Doctor of Philosophy

Statistics Group
Department of Mathematical Sciences
University of Durham
England

July 2012

Dedicated to

Dr Peter Craig

Markov Chain Monte Carlo Methods for Exact Tests in Contingency Tables

Shiler Khedri

Submitted for the degree of Doctor of Philosophy
July 2012

Abstract

This thesis is mainly concerned with conditional inference for contingency tables, where the MCMC method is used to take a sample of the conditional distribution. One of the most common models to be investigated in contingency tables is the independence model. Classic test statistics for testing the independence hypothesis, Pearson and likelihood chi-square statistics rely on large sample distributions. The large sample distribution does not provide a good approximation when the sample size is small. The Fisher exact test is an alternative method which enables us to compute the exact p-value for testing the independence hypothesis. For contingency tables of large dimension, the Fisher exact test is not practical as it requires counting all tables in the sample space. We will review some enumeration methods which do not require us to count all tables in the sample space. However, these methods would also fail to compute the exact p-value for contingency tables of large dimensions. Diaconis and Sturmfels (1998) introduced a method based on the Grobner basis. It is quite complicated to compute the Grobner basis for contingency tables as it is different for each individual table, not only for different sizes of table. We also review the method introduced by Aoki and Takemura (2003) using the minimal Markov basis for some particular tables. Bunea and Besag (2000) provided an algorithm using the most fundamental move to make the irreducible Markov chain over the sample space, defining an extra space. The algorithm is only introduced for $2 \times J \times K$ tables using the Rasch model. We introduce direct proof for irreducibility of the Markov chain achieved by the Bunea and Besag algorithm. This is then used to prove that Bunea and Besag (2000) approach can be applied for some tables of higher dimensions, such as $3 \times 3 \times K$ and $3 \times 4 \times 4$. The efficiency of the Bunea and Besag approach is extensively investigated for many different settings such as for tables of low/moderate/large dimensions, tables with special zero pattern, etc. The efficiency of algorithms is measured based on the

effective sample size of the MCMC sample. We use two different metrics to penalise the effective sample size: running time of the algorithm and total number of bits used. These measures are also used to compute the efficiency of an adjustment of the Bunea and Besag algorithm which show that it outperforms the the original algorithm for some settings.

Declaration

The work in this thesis is based on research carried out at the Statistics Group, the Department of Mathematical Sciences, Durham University, England. No part of this thesis has been submitted elsewhere for any other degree or qualification and it all my own work unless referenced to the contrary in the text.

Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor, Dr. Peter Craig whose encouragement, guidance and support from the start to finish enabled me to develop an understanding of the subject. I am also thankful to my second supervisor, Professor David Wooff, who gave me the opportunity to be able to expand my experience as a statistician.

I gratefully acknowledge the Mathematical Sciences department, for providing the support and equipment I needed to produce and complete my thesis. I would like to show my gratitude to Sharry Borgan, kind father of the department, for giving me the opportunity to teach during my PhD, as well as his on going support through out my studies. I have received a lot of support from IT specialists in the Mathematical Sciences department: Brian Rogers, Bernard Piette, and Mark Short, for which I am truly grateful. I would also like to thank our cleaning staff, Pamela and Michael, whose friendly smiles and warm manner always brightened up our day.

I would like to thank my parents and brother for all their love and encouragement. It is thanks to them that I was raised with a love of science and their endless support helped me in my academic development. I offer special thanks to my very close friend, Amin, who encouraged and supported me all the way through my PhD and has been a great friend during four years of living and studying in Durham. I would like to offer my regards to my very good friends Mohammad Al Turkestani and Dima Smayra and all the other friends for all their support. I should also mention my cool friend John Holmes for keeping me as strong as possible to deal with all the difficulties during the last year of my study. Finally a special thanks to Dr Khalil Rashidian who kindly encouraged me to continue my education at Durham University.

Shiler Khedri

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Categorical data and modelling options

Since the early years of the development of statistical methods, part of the effort has been devoted to providing methods for analysis of categorical data. A categorical variable is qualitative data that takes a value from one of several possible categories. Categorical data, part of an observed data set which consists of categorical variables, are common in many different sciences such as social, medical, biological, behavioural and public health. There are three types of measurement scale from which categorical variables can be derived: nominal, ordinal and interval. Nominal variables have categories without any natural order (such as sex or race). An ordinal variable assumes a natural order for different levels of a variable (e.g. educational level or injury level). Interval variables are created by categorising a continuous scale (e.g. the level of income or age categories). A primary aim of the analysis of categorical data, similar to many other statistical procedures, is to select a model which reasonably describes the data. This is one of the fundamental tasks of statistical analysis. The best model should be selected from among all possible models based on mathematical analysis. To continue, notation has to be introduced. The notation which is used throughout this thesis is described in Table 1.1.

1.1.1 Frequency and contingency tables

Analysis of categorical data generally involves the use of data tables. A univariate categorical variable can be represented through a frequency table which summarizes the number

Table 1.1: Notation of two-way contingency tables

Notation	Description
X	A categorical variable which is represented in rows of a contingency table
Y	A categorical variable which is represented in columns of a contingency table
I	The number of rows of a contingency table
J	The number of columns of a contingency table
T	A contingency table corresponding to the joint categorical random variable (X, Y)
T'	The transpose of a contingency table, produced by substituting rows and columns
T^o	The observed table, where o is an abbreviation for observed
t_{ij}	The frequency for the cell of a contingency table
$t_{.j}$	The marginal total over rows of a contingency table
$t_{i.}$	The marginal total over columns of a contingency table
$t_{..}$	Grand total over all cells of a contingency table
$\boldsymbol{i}$	A vector contains the marginal total over rows, $\boldsymbol{i} = (t_{.1}, t_{.2}, \cdots, t_{.J})$
$\boldsymbol{j}$	A vector contains the marginal total over columns, $\boldsymbol{j} = (t_{1.}, t_{2.}, \cdots, t_{I.})$
t_s	The vector of sufficient statistics of parameters in a contingency table
π_{ij}	The joint probability (X, Y) occurs in the cell (i, j) of a two-way contingency table
$\pi_{i.}$	The marginal distribution of categorical variable X, $\pi_{i.} = \sum_j \pi_{ij}$
$\pi_{.j}$	The marginal distribution of categorical variable Y, $\pi_{.j} = \sum_i \pi_{ij}$
$\pi_{j\|i}$	The conditional distribution of categorical variable Y given X, $\pi_{j\|i} = \pi_{ij}/\pi_{i.}$
$\mathcal{S}$	The set of all contingency tables with the same marginal totals as the observed table, known as the reference set

of occurrences for each level of the categorical variable. For example, suppose that a simple random sample of 50 students are asked their favourite colour between red, blue and green. All the answers can be tabulated in the following frequency table, Table 1.2

Table 1.2: Frequency table of students' favourite colour example

	Favourite colour			
	Red	Blue	Green	Total
Frequency	14	19	17	50
Proportion	0.28	0.38	0.34	1.00

A contingency table is a tabular representation of categorical data when there are two or more categorical variables. A two-way contingency table, first introduced by Karl Pearson in 1904 (Agresti, 1996), can be used to represent the frequency of events in two dimensions, where rows represent different levels of the first variable and columns correspond to different levels of the other variable. Each cell of the table contains the observed frequency for a different combination of the levels of the two categorical variables. In order to represent more than two categorical variables multi-dimensional contingency tables can be applied. Let X and Y denote two categorical variables, X with I categories and Y with J categories. A contingency table with I rows and J columns is called an $I \times J$ table. Classifications of subjects on both variables have $I \times J$ possible outcomes. For example, consider the 50 students classified based on their favourite colour and suppose their sex was also recorded. The results can be represented in a two-way contingency table by considering sex in rows and favourite colour in columns, see Table 1.3

Table 1.3: A two-way contingency table for students' favourite colour example

	Favourite colour			
Sex	Red	Blue	Green	Total
Male	8	11	4	23
Female	11	7	9	27
Total	19	18	13	50

There are two approaches to analysing categorical data: *randomisation* which compute statistics based on tables defined by a limited number of discrete values to investigate the association between the variables. The second is *modelling* where we can check the association by modelling a categorical response variable. The following section illustrates the different sampling methods for contingency tables in the modelling approach.

1.1.2 Sampling distribution for two-way contingency tables

The observed counts in the cells of a contingency table can be treated as random variables with non-negative integer values. The probability of observing counts in a table depends on the assumed sampling distribution. There are three different distributions to represent this probabilistic behaviour: Poisson, full multinomial, and the product of multinomial distributions (Agresti, 1990). Throughout this section we focus on two-way contingency tables.

Poisson distribution

As the frequencies can accept only non-negative integers a simple distribution which has its mass in this range is the Poisson distribution. This way, we assume that the frequency of each cell comes from the Poisson distribution with parameter μ_{ij}, where $\mu_{ij} \propto \pi_{ij}$ and each cell is independent of the rest. It should be noted that the joint probability distribution for a table T with frequencies $T = \{t_{ij}, i = 1, 2, \ldots, I \text{ and } j = 1, 2, \ldots, J\}$ is given by the product of distribution for each cell:

$$\Pr(T) = \prod_{i,j} \frac{\exp(-\mu_{ij})\mu_{ij}^{t_{ij}}}{t_{ij}!} \tag{1.1}$$

where μ_{ij} is the expected value of the t_{ij}. The Poisson distribution is suitable for sampling in which the total sample size is not fixed. This happens when we count the number of events occurring over a period of time (or location). An example of this could be the result of a new test for all patients who arrive at a hospital for a blood test over the period of one year. The result could be represented be classification of test results and blood type. In this case, the sample size is not known before the data collection is completed.

Multinomial distribution

When we sample a fixed number of units of the population a suitable distribution for the counts of the table is given by the multinomial distribution. Suppose that a simple random sample of individuals is chosen from the population and each is assigned into one of $I \times J$ categories of a two-way contingency table. Then the probability of observing table T with counts $T = \{t_{ij}, i = 1, 2, \ldots, I \text{ and } j = 1, 2, \ldots, J\}$ follows the multinomial distribution with probability distribution function as follows:

$$\Pr(T) = \frac{t_{..}!}{\prod_{i,j} t_{ij}!} \prod_{i,j} \pi_{ij}^{t_{ij}} \tag{1.2}$$

where π_{ij} are unknown probabilities of a randomly chosen individual belonging to the category $\{ij\}$. The maximum likelihood (ML) estimate for π_{ij} is the sample proportion, $p_{ij} = t_{ij}/t_{..}$. Note that, the multinomial model can also arise as the conditional distribution of each cell given the total sample size. That is, if $t_{ij} \sim \text{Poisson}(\mu_{ij})$ then

$$\left(\{t_{ij}\} \,\Big|\, \sum_{ij} t_{ij} = t_{..} \right) \sim \text{Multinomial} \left(t_{..} \,,\, \frac{\mu_{ij}}{\sum_{ij} \mu_{ij}} \right).$$

Product of multinomial distributions

With fixed sample size, defined by the level of the row factor, the frequencies of the table cannot be modelled using a multinomial distribution over all cells of the table. This is because the marginal totals of the table corresponding to a sub-population are not random but known quantities. Each row of a two-way table is sampled from the corresponding sub-population. This way, observation of each row of the table is independent, having a probability distribution with probabilities $\{\pi_{1|i}, \pi_{2|i}, \ldots, \pi_{J|i}\}$. Then the counts in each row follow the multinomial distribution, and the probability distribution for table T, the entire set of frequencies, is given by the product of the multinomial distributions:

$$\Pr(T) = \prod_i \left(\frac{t_{i.}!}{\prod_j t_{ij}!} \right) \prod_j \pi_{j|i}^{t_{ij}}. \tag{1.3}$$

Similarly to the multinomial case, if we assume the Poisson distribution over cells, the conditional distribution of the counts on the cells given the row marginal totals $t_{i.}$ produce the same distribution as given in (1.3).

1.2 Independence in two-way contingency tables

A common question in two-way contingency tables is whether the column variable depends on the row variable. In the example of students given in Table 1.3 , the question is whether the favourite colour of students is the same for both males and females; in other words, whether the favourite colour of students is independent of their gender. In the case of independence of rows and columns, the sufficient statistics are the marginal totals. In this case, regardless of the sampling distribution, we have the same inference for the table, as it implies that the probability distribution of the frequencies conditioned on the values of sufficient statistics, is given by the hyper-geometric distribution for 2×2 contingency

table. In the case of increasing the number of levels of rows or columns, the distribution will generalise to the multivariate hyper-geometric distribution.

For multinomial sampling with probabilities $\{\pi_{ij}\}$ the null hypothesis of statistical independence is $H_\circ : \pi_{ij} = \pi_{i.}\pi_{.j}$ for all i and j. In the case of product of independent multinomial distributions, independence corresponds to homogeneity of probabilities among the rows. If the sample size is large enough, the central limit theory can provide an approximation using the normal distribution for computing the probability distribution of the test statistics for the specified statistical model. The Pearson χ^2 and the likelihood ratio are two possible tests for the independence hypothesis in two-way contingency tables. It should be noted that for all three sampling assumptions of the contingency tables the Pearson χ^2 test is the same.

1.2.1 Large-sample independence tests

The large-sample approximation has been common for analysing contingency tables for decades. When the sample size is large enough, one may use classical methods which essentially work based on the chi-square distribution. The well-known Pearson chi-squared test of independence uses the fact that the expected frequency for cell (i, j) of the contingency table in the case of independence is $\mu_{ij} = t_{..}\pi_{i.}\pi_{.j}$, where $\pi_{i.}$ and $\pi_{.j}$ are unknown marginal probabilities of the joint distribution. The ML estimate of the marginal probabilities are sample marginal proportions $\widehat{\pi}_{i.} = t_{i.}/t_{..}$ and $\widehat{\pi}_{.j} = t_{.j}/t_{..}$. Equivalently the estimate of the expected frequency is $\widehat{\mu}_{ij} = t_{..}\widehat{\pi}_{i.}\widehat{\pi}_{.j}$. Then the Pearson chi-squared statistic is given by:

$$X^2 = \sum_i \sum_j \frac{(t_{ij} - \widehat{\mu}_{ij})^2}{\widehat{\mu}_{ij}}$$

which asymptotically follows the χ^2 distribution with degrees of freedom $(I-1)(J-1)$.

The likelihood ratio test produces another test statistic under the null hypothesis of independence. For multinomial sampling the likelihood of a two-way contingency table is

$$\mathcal{L}(\pi_{ij}|\{t_{ij}\}) = \prod_{i=1}^{I}\prod_{j=1}^{J} \pi_{ij}^{t_{ij}}.$$

Under the null hypothesis of independence, $\widehat{\pi}_{ij} = t_{i.}t_{.j}/t_{..}$, whilst in general $\widehat{\pi}_{ij} = t_{ij}/t_{..}$. Hence, the likelihood ratio statistic equals

$$\Lambda = \frac{\prod_i \prod_j (t_{i.}t_{.j})^{t_{ij}}}{t_{..}^{t_{..}} \prod_i \prod_j t_{ij}^{t_{ij}}}.$$

The likelihood-ratio chi-squared statistic is $-2\log\Lambda$, usually denoted by G^2, and has the following form

$$G^2 = 2\sum_i \sum_j t_{ij} \log\left(t_{..}\frac{t_{ij}}{t_{i.}t_{.j}}\right),$$

where G^2 for large sample size follows the χ^2 distribution with degrees of freedom equal $(I-1)(J-1)$.

When a contingency table contains cells with small expected frequencies, even for a large sample size, the large-sample approximation is not reliable. In the case of small sample-size the chi-square distribution does not provide a good approximation for the distribution of the test. When the sample size is small, all possible combinations of the data can be evaluated to compute what is known as the exact p-value. The best known example of an exact test is given by R. Fisher (1934) in which an exact distribution of a table is used rather than the large-sample approximation.

1.2.2 Conditional tests

The exact tests considered in this thesis are all what are known as conditional tests.

Conditional inference refers to the methods in which the inference is based on the distribution of parameters conditioned on their sufficient statistics (Anderson, 1974). In this way, the distribution of the parameter is a function of the sufficient statistics. The realisation of sufficient statistics is considered as a value for the nuisance parameter in the modelling. A nuisance parameter is any parameter which is not of immediate interest but which must be accounted for in the analysis of those parameters which are of interest. A well-known example of a nuisance parameter is the variance of a normal distribution, when the mean is of primary interest.

Suppose we want to test the independence assumption between rows and columns in a two-way contingency table. Conditional inference uses the conditional distribution of the sufficient statistics under the full model, $\{t_{ij}\}$, given sufficient statistics for the model under the null hypothesis. Here, we illustrate that the conditional test for independence provides the multiple hyper-geometric distribution. We illustrate this for the general case of $I \times J$ contingency tables under multinomial sampling, the result of the conditional test is the same for the Poisson and product of multinomial sampling. Under the null hypothesis of independence, $H_\circ : \pi_{ij} = \pi_{i.}\pi_{.j}$, the probability function of t_{ij} (1.2) simplifies

to

$$\begin{aligned}\Pr(\{t_{ij}\}) &= \frac{t_{..}!}{\prod_i \prod_j t_{ij}!} \prod_i \prod_j \pi_{ij}^{t_{ij}} \\ &= \frac{t_{..}!}{\prod_i \prod_j t_{ij}!} \prod_i \prod_j (\pi_{i.}\pi_{.j})^{t_{ij}} \\ &= \frac{t_{..}!}{\prod_i \prod_j t_{ij}!} \left(\prod_i \pi_{i.}^{t_{i.}}\right)\left(\prod_j \pi_{.j}^{t_{.j}}\right). \end{aligned} \tag{1.4}$$

Hence, the minimal sufficient statistics based on H_0 will be $\boldsymbol{t}_s = \{t_{i.}, t_{.j}\}$ and the nuisance parameters are $\{\pi_{i.}\}$ and $\{\pi_{.j}\}$. On the other hand, the statistics $\{t_{i.}\}$ and $\{t_{.j}\}$ under the null hypothesis are independent and follow the multinomial distributions:

$$\begin{aligned}\{t_{i.}\} &\sim \text{Multinomial}(t_{..}\,,\, \{\pi_{i.}\}) \\ \{t_{.j}\} &\sim \text{Multinomial}(t_{..}\,,\, \{\pi_{.j}\}) \end{aligned} \tag{1.5}$$

The joint probability distribution of the sufficient statistics $\{t_{i.}\}$ and $\{t_{.j}\}$ is given by:

$$\Pr(\boldsymbol{t}_s) = \frac{t_{..}!}{\prod_i t_{i.}!} \prod_i \pi_{i.}^{t_{i.}} \frac{t_{..}!}{\prod_j t_{.j}!} \prod_j \pi_{.j}^{t_{.j}}. \tag{1.6}$$

Now we need to find the conditional probability distribution of $(\{t_{ij}\} \mid \boldsymbol{t}_s)$. The joint probability function of $\{t_{ij}\}$ and $\boldsymbol{t}_s$ is the same as the probability function of $\{t_{ij}\}$, (1.4), as $\{t_{ij}\}$ determines the sufficient statistics $\{t_{i.}\}$ and $\{t_{.j}\}$. Hence the conditional distribution of the sufficient statistics based on the full model given the sufficient statistics under the null hypothesis is obtained by (1.4) divided by (1.6):

$$\Pr(\{t_{ij}\} \mid \boldsymbol{t}_s) = \frac{\prod_i t_{i.}! \prod_j t_{.j}!}{t_{..}! \prod_i \prod_j t_{ij}!}. \tag{1.7}$$

This is the called the multivariate hypergeometric distribution. If $I = J = 2$ this is the standard hypergeometric distribution. The same reasoning can show that the conditional test for Poisson sampling results in the multivariate hypergeometric distribution. see Agresti (1990) for the product of multinomial sampling.

1.2.3 Fisher's exact test

In the case of having a small sample size, the chi-square distribution is not a valid approximation for the distribution of the test statistic. An alternative method for this is to find the exact distribution instead of approximation of large samples, known as small-sample

tests. Fisher's exact test is one of the small-sample tests of independence for 2×2 contingency tables. Another example is the randomisation test of independence. It should be noted that for 2×2 contingency tables, independence is equivalent to $H_0 : \theta = 1$, where $\theta = \frac{\pi_{1|1}\pi_{2|2}}{\pi_{1|2}\pi_{2|1}}$ is the odds ratio. The p-value for this test is the sum of certain hypergeometric probabilities. The notation used to show examples of Fisher's exact test is as follows. The cell frequencies are denoted by t_{ij}, $i, j = 1, 2$. Let the contingency tables be denoted by

$$T = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \tag{1.8}$$

and the observed contingency table by $T^o = \{t^o_{ij}\}$.

Given the marginal totals, t_{11} determines the other three cell counts with the range of possible values $m_- \leq t_{11} \leq m_+$, where $m_- = \max(0, t_{1\cdot} + t_{\cdot 1} - t_{\cdot\cdot})$ and $m_+ = \min(t_{1\cdot}, t_{\cdot 1})$. Fisher (1934) showed that under the null hypothesis of independence the probability of obtaining any such set of values given the marginal totals follows the hypergeometric distribution

$$\Pr(\{t_{ij}\} \mid \{t_{i\cdot}\}, \{t_{\cdot j}\}) = \Pr(t_{11}) = \frac{\binom{t_{1\cdot}}{t_{11}}\binom{t_{2\cdot}}{t_{\cdot 1} - t_{11}}}{\binom{t_{\cdot\cdot}}{t_{\cdot 1}}}.$$

The formula above provides the exact probability for this particular arrangement of the data, conditional on the marginal totals. Computation of Fisher's exact p-value depends on the form of the alternative hypothesis:

One-sided p-value: In the case of a one-sided alternative hypothesis test, say $H_1 : \theta > 1$, for given marginal totals, tables with larger t_{11} have larger sample odds ratios which correspond to stronger evidence in favour of H_1. Hence, the p-value equals $\Pr(t_{11} \geq t^o_{11})$, where t^o_{11} is the observed value for the first cell of contingency table (Fisher, 1934; Yates, 1934). Similarly, for $H_1 : \theta < 1$, the p-value is computed through summation over all tables in which $\Pr(t_{11} \leq t^o_{11})$.

Two-sided p-value: Two-sided tests are more common than one-sided tests in applications. For a two-sided hypothesis test, there are four different criteria to compute p-values.

- The most common approach is to find all tables for which $\Pr(s) \leq \Pr(t^o_{11})$ and then computing the p-value as the sum of the probability of the tables.

- Another way is to consider $\Pr(t_{11}^o)$ for tables that are farther from H_0. This is identical to computing $\Pr(\chi^2 \geq \chi_o^2)$, where χ_o^2 is the observed Pearson statistic

- Another approach computes the p-value as follows

$$\text{p-value} = 2 \min[\Pr(t_{11} \geq t_{11}^o), \Pr(t_{11} \leq t_{11}^o)]$$

  A disadvantage of this method is the possibility of having a p-value greater than 1.

- A fourth kind of p-value takes p-value= $\min[\Pr(t_{11} \geq t_{11}^o), \Pr(t_{11} \leq t_{11}^o)]$ plus the probability in the other tail that is as close as possible to that one-tail probability.

1.3 Enumeration methods

In this section we illustrate the class of enumeration methods for exact tests in contingency tables. The exact test requires working with all possible tables, called the reference set, with the same marginal totals as the observed table. The number of possible tables increases factorially fast as row, column or the total sample size increases and so using Fisher's methods becomes difficult as generating all tables is infeasible. In order to solve this problem, computational algorithms have been proposed. The following sections discuss two algorithms for computing the exact p-value; the algorithm introduced by Pagano and Halvorsen (1981) and the network algorithm proposed by Mehta and Patel (1983).

1.3.1 Pagano and Halvorsen's algorithm

Pagano and Halvorsen (1981) provide a computational algorithm for calculating the exact significance value for two-way contingency tables. The advantage of this algorithm is that the computation does not need the enumeration of all tables with the specified marginal totals, which makes it faster than other algorithms. The p-value is the sum of probabilities of all tables belonging to the reference set, $\mathcal{S}$, for which the probability of the table is less than the probability of the observed table, T^o, where the probability of each table is given by (1.7).

In this algorithm the probability of each table is expanded as a product of hyper-geometric conditional probabilities as follows:

$$\Pr(T) = \prod_{j=1}^{J-1} \prod_{i=1}^{I-1} p_{ij} \tag{1.9}$$

where p_{ij} is the probability of the value in cell ij given the values in the "previous" cells (those in rows, $1, \ldots i-1$ or in row i and colums $1, \ldots j-1$) . Note that $p_{I.}$ and $p_{.J}$ do not appear in this expression as they are trivial due to the fact that all tables must have the same marginal totals as the observed table.

It is reasonably straightforward to show that each conditional probability is hypergeometric and is given by

$$p_{ij} = \prod_{j=1}^{J-1} \prod_{i=1}^{I-1} \frac{\binom{J_j - z_{ij}}{t_{ij}} \binom{n - v_{ij} - J_j + z_{ij}}{I_i - r_{ij} - t_{ij}}}{\binom{n - v_{ij}}{I_i - r_j}}$$

where

$$\begin{aligned}
I_i &= \sum_{h=1}^{J} t_{ih} \\
J_j &= \sum_{g=1}^{I} t_{gj} \\
z_{ij} &= \sum_{g=1}^{i-1} t_{gj} \\
r_{ij} &= \sum_{h=1}^{j-1} t_{ih} \\
v_{ij} &= \sum_{g=1}^{i-1} I_g + \sum_{h=1}^{j-1} J_h - \sum_{g=1}^{i-1} \sum_{h=1}^{j-1} t_{gh}
\end{aligned}$$

When enumerating the possible values in cell ij given the values in previous cells, the bounds are given by $l_{ij} \leq t_{ij} \leq u_{ij}$ where

$$\begin{aligned}
l_{ij} &= \max(0\ ,\ v_{ij} + J_j - z_{ij} + I_i - r_{ij} - n) \\
u_{ij} &= \min(J_j - z_{ij}\ ,\ I_i - r_{ij}).
\end{aligned}$$

The overall algorithm is more easily described using linearised indices $q = (j-1)(I-1)+i$ so that, for example, t_{ij} becomes t_q. A table is then equivalent to the vector $t = (t_1, ..., t_m)$ where $m = (I-1)(J-1)$. The structure of the algorithm for enumerating all tables satisfying the constraints, known as G-algorithm, is shown in Figure 1.

The advantage of this algorithm is that computing the p-value requires enumerating only some subsets of cells in tables which satisfy the constraints. The following theorem helps to find a subset of the reference set which is large enough to find the exact p-value.

Theorem 1.1 *Suppose that for some $T \in \mathcal{S}$ there exist $d \leq m$ such that $\prod_{q=1}^{d} p_q < \Pr(T^o)$, and the cell entries $\boldsymbol{t}$ are $t_1, t_2, \ldots, t_d$. Then*

$$\sum_{T \in D} \Pr(T) = \prod_{q=1}^{d} p_q$$

where $\mathcal{D}$ is the set of all tables $\in \mathcal{S}$ for which the first d components of the table are $t_1, t_2, \ldots, t_d$.

This theorem helps because all tables in D contribute to the p-value and their total contribution is $\prod_{q=1}^{d} p_q$. Therefore we do not need to enumerate them all individually and can proceed immediately to considering the next possible value for cell t_d.

1.3.2 Mehta and Patel network algorithm

There is an alternative algorithm proposed by Mehta and Patel (1983) which extends the bounds of computational feasibility relative to Pagano and Halvorsen's algorithm. Before explaining the network algorithm we need to formulate some notation.

Let T be a $I \times J$ contingency table and each cell is shown by $t_{kk'}$ which is the intersection of row k and column k'. Marginal totals are defined as follow:

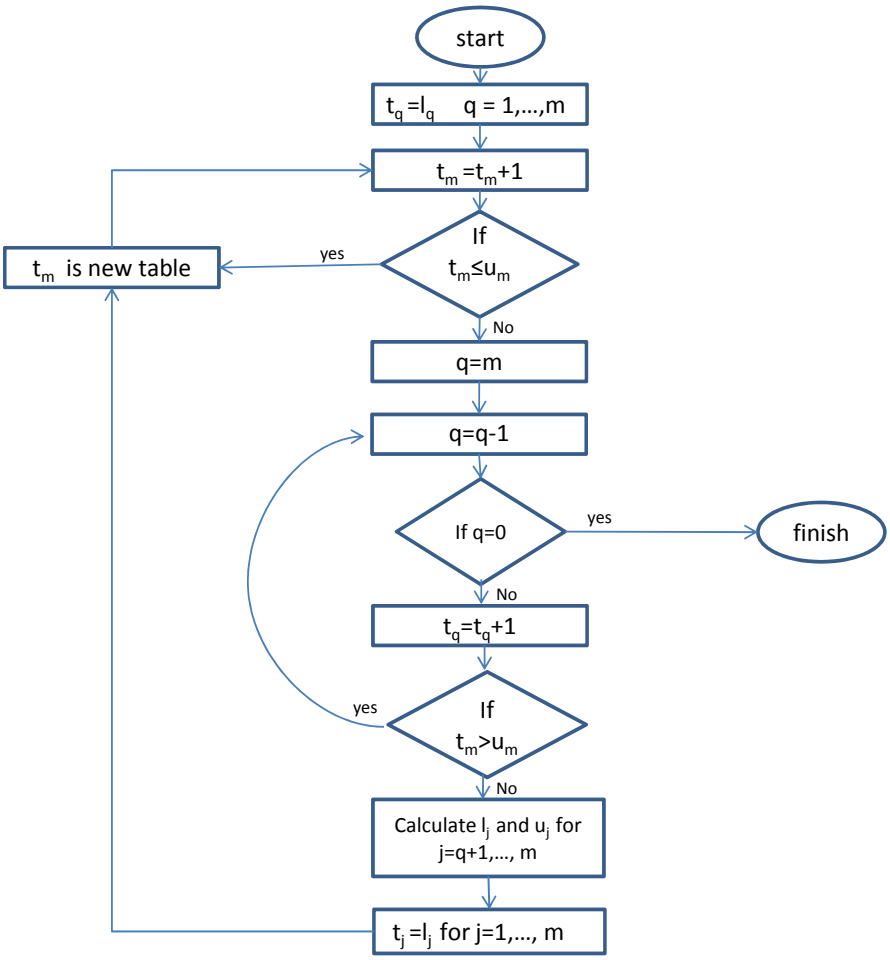
$$I_k = \sum_{k'=1}^{J} t_{kk'} \qquad J_{k'} = \sum_{k=1}^{I} t_{kk'}.$$

The reference set is shown by:

$$\mathcal{S} = \{T : T \text{ is } I \times J \text{ table}, \sum_{k'=1}^{J} t_{kk'} = I_k, \sum_{k'=1}^{I} t_{kk'} = J_{k'}.\}$$

Under the null hypothesis, the probability of each table of the reference set is expressed as a product of multinomial coefficients as follows:

Figure 1.1: *G-algorithm flowchart.*



$$\Pr(T) = \frac{\left(\prod_{k'=1}^{J} \frac{J_{k'}!}{t_{1k'}!\cdots t_{Ik'}!}\right)}{\frac{\omega!}{I_1!\cdots I_I!}}, \tag{1.10}$$

where $\omega = \sum_{k=1}^{I} I_k$. For later use ϕ is defined as

$$\phi = \frac{\omega!}{I_1!\, I_2! \ldots I_I!}.$$

A network representation consists of nodes and arcs, constructed in $J+1$ stages. At any stage τ there exists a set of nodes each labelled by a unique vector $(\tau, I_{1\tau}, \ldots, I_{I\tau})$ where $I_{k\tau}$ is defined to be $\sum_{i=1}^{k} t_{i\tau}$. The range of $I_{k\tau-1},\ k = 1, \ldots, I$ for these successor nodes is given by

$$\max\left\{0, I_{k\tau} - J_\tau + \sum_{l=1}^{k-1}(I_{1\tau} - I_{1\tau-1})\right\} \leq I_{k\tau-1} \leq \min\left\{I_{k\tau}, \delta_{\tau-1} - \sum_{l=1}^{k-1} I_{l\tau-1}\right\}$$

where $\delta_{k'} = \sum_{l=1}^{k'} J_l$. The length of an arc from a node in stage τ to a node in stage $\tau - 1$ is defined to be equal to

$$\frac{J_\tau!}{(I_{1\tau} - I_{1\,\tau-1}!)\cdots(I_{I\tau} - I_{I\,\tau-1}!)}.$$

so that the total length of a path from stage J to stage 0 is the numerator in equation (1.10). Suppose that T^o is the observed table. We aim to identify and to sum all paths whose lengths do not exceed $\phi.p(T^o)$. But this is usually computationally infeasible. So we may use the following alternative method; first, we define the following two items:

SP(τ, $I_{1\tau}$, …, $I_{I\tau}$) = The length of the shortest subpath from node τ to node 0.
LP(τ, $I_{1\tau}$, …, $I_{I\tau}$) = The length of the longest subpath from node τ to node 0.

Let,

$$PAST = \prod_{k'=\tau+1}^{J}\left\{\frac{J_k'!}{(I_{1k'} - I_{1k'-1})!\ \cdots\ (I_{Ik'} - I_{Ik'-1})!}\right\}$$

We can enumerate the paths implicitly if either of the following two conditions holds:

$$(1) \quad PAST \times LP(\tau, I_{1\tau}, \ldots I_{I\tau}) \leq \phi \times \Pr(T^o)$$

or

$$(2) \quad PAST \times SP(\tau, I_{1\tau} \ldots I_{I\tau}) > \phi \times \Pr(T^o)$$

If (1) is satisfied we know immediately that every path is suitable and these paths contribute to the p-value. If (2) is satisfied we know that every path exceeds $\phi.\Pr(T^o)$ and

they are not suitable so none of these paths can contribute to the p-value and they are dropped.

1.4 Exact method using Monte Carlo simulation

Bernard (1963) proposed the Monte Carlo approach for computing exact p-values. For a general log-linear model, enumerating the test distribution has been computationally infeasible and a Monte Carlo exact test is necessary (Smith et al., 1996). This approach does not need complete enumeration or asymptotic approximation for computing the exact p-value. Assume that u^o is the value of a particular chosen test statistic for an observed table T^o, which test statistic to use is a decision to be made by the person applying the test.

We generate a sample of size n of tables from the reference distribution, $T^{(1)}, T^{(2)}, \ldots, T^{(n)}$. The sequence of statistics for the sampled tables is denoted by $u^{(1)}, u^{(2)}, \ldots, u^{(n)}$, corresponding values for these tables. Assuming u^o is a sample from reference distribution, we have $n+1$ random sample from the distribution. If we ignore the possibility of ties, the rank of u^o among all $n+1$ values is drawn from a uniform distribution, $\mathcal{U}\{1, 2, \ldots, n\}$. A large value of u^o compared to the sampled values shows that the observed statistic is not from the reference distribution. To compute the p-value, the random sample statistics are ranked. If the rank of observed statistics u^o is the $q-$th largest among $u^{(1)}, u^{(2)}, \ldots, u^{(n)}$, then the approximate exact p-value is reported as $q/(n+1)$. For more detailed discussion see Besag and Clifford (1989).

Although the Monte Carlo methods is well defined for testing independence, there are many hypothesis of interest in multi dimensional contingency tables for which the Monte Carlo test has not been constructed. The Monte Carlo test relies on repeated random sampling from an exact probability distribution of the test statistic under the null hypothesis. In the absence of a reference distribution, a Monte Carlo Markov chain approach can be applied which is based on generating samples from the conditional distribution. So when simple Monte Carlo tests are not available, in some situations an MCMC procedure can help as an alternative.

1.5 Markov chain Monte Carlo (MCMC)

In this section we illustrate the Markov chain Monte Carlo (MCMC) approach to analyse contingency tables. MCMC methods have many applications in statistics; for a general introduction see Gamerman and Lopes (2006). To perform an exact test the null distribution of an appropriate test statistic is required. For this, we define the concept of *move* and *degree of a move*, then the notation will be introduced. Furthermore we define some concepts and finally explain how MCMC works.

Definition (*Move*): A move for an observed table, T^o, of dimension $I \times J \times K$ is a table with the same dimension with entries of $-1, 0, 1$ such that all two-way margins are equal to zero (Bunea and Besag, 2000).

Definition (*Basic Move*): The simplest type of move, with the fewest non zero entries, m is called a *basic move* which for some $i', i'', j', j'', k', k''$, where $i' \neq i''$, $j' \neq j''$, $k' \neq k''$ has

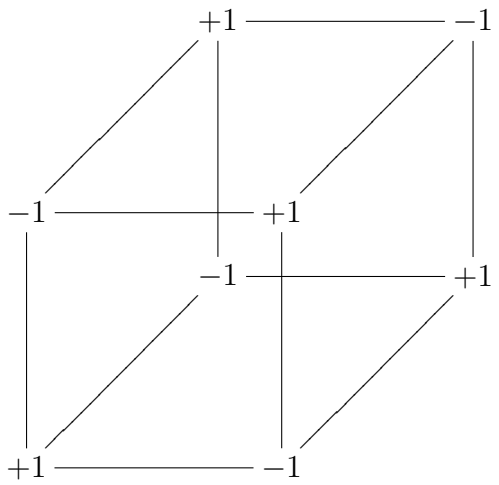
$$m_{i'j'k'} = m_{i'j''k''} = m_{i''j'k''} = m_{i''j''k'} = +1$$

$$m_{i'j'k''} = m_{i'j''k'} = m_{i''j'k'} = m_{i''j''k''} = -1$$

and all remaining elements are zero.

Definition (*Degree of a Move*): If we denote the positive part of a move, $m \in M$, by m^+ where $m^+_{ijk} = \max(m_{ijk}, 0)$ and the negative part by $m^-_{ijk} = \max(-m_{ijk}, 0)$ then $m = m^+ - m^-$. So the degree of m is defined by $\deg(m) = \sum_{i,j,k} m^+_{ijk} = \sum_{i,j,k} m^-_{ijk}$.

Throughout this chapter, we use m as notation for a move. A set of possible moves will be denoted by M. Each move has a degree which will be denoted by $\deg(m)$. A basic move has the smallest possible degree which is 4, and the set of all basic moves will be denoted by M_4. An example of a three dimensional view of a move in M_4 is shown in the following picture:



All the other moves of higher degrees can be written as linear combinations, with integer coefficients, of moves from M_4 (Aoki and Takemura, 2003).

The difference, D, of two tables in a reference set has the same dimension as the observed tables but with the marginal totals equal to zero because T and T^* have the same marginal totals.

1.5.1 Markov chains and Metropolis-Hastings

A Markov chain with states S is defined as follows: The process starts in one of these states and moves successively from one state to another. Each move is called a step. If the chain is currently in state s_k, then it moves to state $s_{k'}$ at the next step with a probability denoted by $p_{kk'}$, where this probability does not depend upon which states the chain was in before the current state. The probabilities $p_{kk'}$ are called transition probabilities. The process can remain in the state it is in, and this occurs with probability p_{kk}. An initial probability distribution, defined on S, specifies the starting state. A Markov chain is said to be irreducible if its state space is a single communicating class, in other words, if it is possible to get to any state from any state.

When direct methods are not available for generating from a multivariate distribution of interest, MCMC sampling can often be used. To implement this approach we need methods for constructing a Markov chain with the required equilibrium distribution. One of the methods used for this purpose is the Metropolis-Hastings algorithm. The Metropolis-Hastings method is an MCMC procedure for generating samples from arbitrary multivariate distributions:

Let $f(T)$ be the desired distribution.

- given current T, generate T_1 from some transition proposal distribution: a family of conditional probability distributions $q(T_1 \mid T)$ which are defined for all valid T and T_1;
- accept T_1 with probability $a(T, T_1)$,

$$a(T, T_1) = \min\left\{\frac{f(T_1)q(T \mid T_1)}{f(T)q(T_1 \mid T)}, 1\right\}, \tag{1.11}$$

  where it is required that $f(T)q(T_1 \mid T) > 0$.
- otherwise, retain T.

When applying Metropolis-Hastings to our problem, T is a table and we propose a new table T_1 by selecting one or more sub-tables of T and by adding a randomly chosen move from a suitable set changing the cell counts in these sub-tables so that the sufficient statistics for the nuisance parameters of the model being considered are maintained (Smith et al., 1996). For example, in the independence model the sub-tables are all tables with the same dimension and the same two-way marginal totals.

1.5.2 MCMC approach for exact test in contingency tables

One of the problems occurs when using MCMC approach to test no three-way interaction effect in $2 \times J \times K$ contingency tables is maintaining the irreducibility of the chain. In testing for no three-way interaction, the sufficient statistics are all margins of the form $t_{ij.}, t_{i.k}, t_{.jk}$ in which the conditional distribution $f(T)$ is given by:

$$f(T) \propto \frac{1}{\prod_{ijk} t_{ijk}!}, \qquad T \in \mathcal{S}.$$

Let $T^o = \left\{t^o_{ijk}\right\} \in Z^{IJK} \geq 0$ be an $I \times J \times K$ contingency table, when the reference set is:

$$\begin{aligned} S\left(\left\{t_{ij.}\right\}, \left\{t_{i.k}\right\}, \left\{t_{.jk}\right\}\right) &= \left\{T \mid t_{ij.} = t^o_{ij.}, t_{i.k} = t^o_{i.k}, t_{.jk} = t^o_{.jk}, t_{ijk} \in \mathbb{N},\right. \\ &\quad \left. i \in (1, \ldots, I), j \in (1, \ldots, J), k \in (1, \ldots, K)\right\} \end{aligned} \tag{1.12}$$

Our aim is to construct a Markov chain over the reference set of three-way contingency tables. For any given set of moves $\mathcal{M}$ the associated transition is defined as:

1. Let $T \in \mathcal{S}$ denote the current state of the chain.
2. Choose $m \in \mathcal{M}$ at random
3. Define $T_{new} = T + m$
4. If $T_{new} \in \mathcal{S}$, select T_{new} as the next state of the chain with probability min $\left\{1, \frac{f(T_{new})}{f(T)}\right\}$, otherwise retain T

It should be noted that because we are using a symmetric transition proposal distribution $q(T \mid T_1) = q(T_1 \mid T)$ the acceptance probability in (1.11) has simplified.

Example: To illustrate the method described above, consider the following observed table, T^o

0	1	1
1	1	1
1	0	1

1	0	1
0	1	0
1	1	1

which is a $2 \times 3 \times 3$ table. The two large squares correspond respectively to $i = 1$ and $i = 2$. In each large square, j and k respectively index rows and columns.

Now we choose uniformly a random move from $\mathcal{M}$, say $m_1 \in \mathcal{M}$ and we produce the next table T_{new} using $T_{new} = T^o + m_1$.

Let suppose that m_1 chosen from M is

0	+	−
0	0	0
0	−	+

0	−	+
0	0	0
0	+	−

and hence $T_{new} = T^o + m_1$ is given by:

0	2	0
1	1	1
1	−1	0

1	−1	2
0	1	0
1	2	0

As we observe, T_{new} has some negative cells. Since $T_{new} \notin S$ then we retain T^o as the next table sampled. Now suppose we draw the next move $m_2 \in M_2$

+	−	0
0	0	0
−	+	0

−	+	0
0	0	0
+	−	0

$T_{new} = T^o + m_2$ is

1	0	1
1	1	1
0	1	1

0	1	1
0	1	0
2	0	1

As $T_{new} \in \mathcal{S}$ then we keep it as the next state with probability $\alpha = \min\left\{1, \frac{f(T_{new})}{f(T)}\right\}$. That is, if $\alpha = 1$ then T_{new} will be chosen for the next state. If $\alpha \leq 1$ then number U should be selected uniformly so that

$$\begin{array}{llll} \text{if} & U < \alpha \rightarrow & \text{select} & T_{new} \\ \text{if} & U > \alpha \rightarrow & \text{retain} & T^o \end{array}$$

Now, $f(T_{new}) = k\frac{1}{2}$ and $f(T^o) = k$, so $\alpha = \min\left\{1, \frac{f(T_{new})}{f(T^o)}\right\} = \frac{1}{2}$. Now, assume that U has been selected randomly as 0.23. Then $U < \alpha$ and as a result, T_{new} is selected as the next table. The algorithm will continue by starting the transition proposal process with T_{new} as the current state.

Chapter 2

Algorithms for 2×J×K tables

2.1 Review of Bunea and Besag (2000)

In this chapter first we review the Markov chain Monte Carlo approach for exact tests introduced by Bunea and Besag (2000) for assessing the goodness of fit of probability models to observed datasets in $2 \times J \times K$ tables. The problem with their algorithm for an arbitrary I in a table of $I \times J \times K$ is that the irreducibility of the chain may fail because it has not been shown that their algorithm can communicate between all tables in the reference set when $I > 2$. So, the focus is on $I = 2$ as it helps to use the known results of the Rasch model.They also show that, when $t_{.jk}$ are all positive in a $2 \times J \times K$ contingency table, it is not necessary to use their algorithm for testing for the absence of three-way interaction as the ordinary algorithm using basic moves leads to an irreducible Markov chain. It should be noted that in section 2.1 we do not present any new idea or proof, but we simply introduce our own version of Bunea and Besag's algorithm. However, in section 2.2 we offer a new proof for the irreducibility of the Markov chain using the specific set of moves, known as M^*. Finally, we provide a new proof in section 2.3 that using only basic moves while tables are allowed to have a single -1 entry can still lead to irreducibility of the Markov chain over $\mathcal{S}$.

2.1.1 Bunea and Besag's algorithm

Some definitions are required when introducing this algorithm. The reference set $\mathcal{S}$ is defined as the set of all tables with non-negative entries with the same marginal totals as

the observed table. We also define a new set, $\mathcal{S}'$, as the space of all $2 \times J \times K$ tables with the same two-way margins as the observed table, having all non-negative entries except for at most a single -1 (Bunea and Besag, 2000). The set of all moves M^*, for three-way tables is defined as (Diaconis and Sturmfels, 1998):

$$M^* = M_4 \cup M_6 \cup \cdots \cup M_{2\min(J,K)}. \tag{2.1}$$

Bunea and Besag (2000) provide the MCMC algorithm for sampling $2 \times J \times K$ tables which may move outside $\mathcal{S}$ into $\mathcal{S}'$ but later return inside $\mathcal{S}$.

The current state of the chain is shown by $T \in S$ and the proposed subsequent state is shown by T_s. Their modified version of the Metropolis-Hastings algorithm described in chapter 1 is as follows:

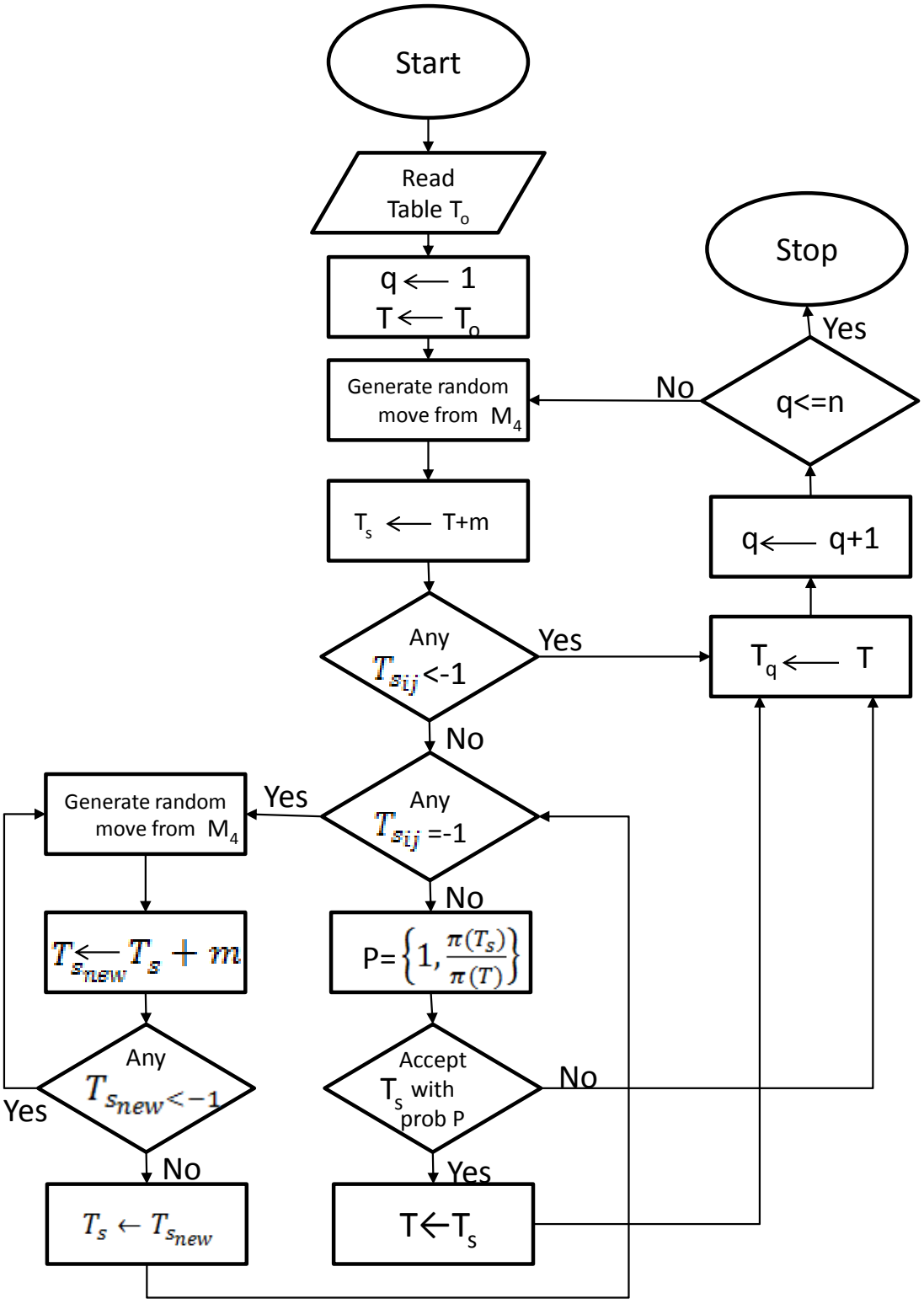
- Start with the observed table, T^o, as the current state.
- Randomly choose a basic move, $m \in M_4$.
- Define $T_s = T^o + m$.
- If the cell entries of T_s are all non-negative then T_s can be the next state with acceptance probability of P which is defined as follows

  $$P = \left\{1, \frac{f(T_s)}{f(T)}\right\};$$

  if not accepted the current state becomes the next state.
- If T_s has a single -1 entry then we keep on choosing $m \in M_4$ to add on, rejecting any which produce a cell entry less than -1 and continue until all entries of T_s are non-negative. We then apply the same acceptance test as in the previous bullet point.
- If T_s has any cell entry less than -1, T_s will be rejected and the current state becomes the next state.

The algorithm is also described in the following flowchart

Figure 2.1: *Bunea and Besag algorithm flowchart.*



It can be shown that this algorithm defines an irreducible Markov chain for the following reasons:

(1) Using M^* in the algorithm introduced in chapter 1 leads to an irreducible Markov chain. Any two tables in S are connected by a path in S formed by making moves from M^*

(2) Any move in a path in (1) is decomposable to basic moves in such a way that replacing it by the basic moves may take the path outside S but not outside S'.

2.1.2 Rasch model

Consider J categorical variables which are tabulated in the table T. Subject i responds to item j for $j = 1, 2, \ldots, J$ and the answer is coded to $t_{ij} = 0, 1$. For example, the response for item i can be a disagreement versus agreement with comment j, the failure or success at performing task j, presence or absence of symptom or feature j, etc. It is often wise to separate subject (row) effect from item (column) effect in the matrix of response. One of the earliest models to do this was developed by Rasch (1980), in which he specifies row and column effect in an additive logistic model for the matrix of responses (Erosheva et al., 2002).

As an example, when there is a student i with ability θ_i and question j with difficulty β_j then the Rasch probability of a correct answer is equal to

$$\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}.$$

Now, when we consider $I = 2$, we are able to use results for the Rasch model. A key result of the Rasch model, Bunea and Besag (2000) citing Ryser (1963), is that any two binary $J \times K$ tables with the same row and column totals can be connected by a sequence of moves of the type depicted in a single layer of basic moves.

The following propositions are the results of Bunea and Besag (2000):

Proposition 2.1 *Any $m \in M^*$ can be decomposed into basic moves.*

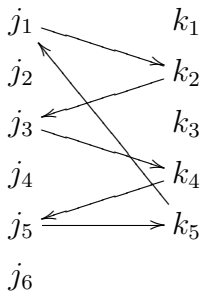
Proof we may prove this proposition by taking an arbitrary move in M^*, $m \in M^*$, and showing that there exists a set of basic moves, by which the move can be constructed. Let $m \in M^*$, then $m \in M_\nu$, where $2 \leq \nu \leq \min(J, K)$. Here, J is the number of rows and K

is the number of columns in our table of $2 \times J \times K$. Any $m \in M_\nu$ can be depicted through a circuit in a bipartite graph on J and K nodes. A bipartite graph is a graph whose nodes can be divided into two independent disjoint sets so that each edge connects a node in one set with a node in the other. Here, one set is j for $j = 1, 2, \ldots, J$ corresponding to the rows and the other set is by k for $k = 1, \ldots, K$ corresponding to the columns. A circuit is a closed walk which means it is a path which doesn't have a repeated vertex except for the first and last. Any circuit in the bipartite graph can be identified with a particular move. For example, consider the following move in a $2 \times 6 \times 5$ table.

	+			−
	−		+	
			−	+

	−			+
	+		−	
			+	−

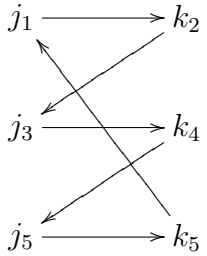
the above move can be represented by the following bipartite graph.



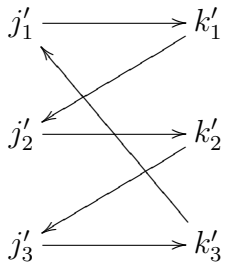
where arrows from left to right correspond to + and right to left correspond to −. This move can be written as follows:

$$(j_1, k_2)(k_2, j_3)(j_3, k_4)(k_4, j_5)(j_5, k_5)(k_5, j_1)$$

and is an element of M_6. This means that for each of the rows and columns just three levels have been changed. We make a sub-graph of the corresponding bipartite graph from those levels, in our example:



For simplicity of notation we use the new notation of j' and k' with ordinal indices of $1, 2, 3, \cdots, \nu$. So, we present the sub-graph by



which corresponds to the following move:

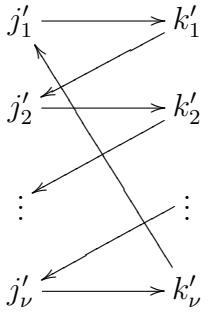
+		−
−	+	
	−	+

−		+
+	−	
	+	−

Now, this circuit can be divided into the following basic moves each of which involves just two rows and two columns.

$$\begin{aligned}(j'_1,k'_1)(k'_1,j'_2)(j'_2,k'_2)(k'_2,j'_3)(j'_3,k'_3)(k'_3,j'_1) &= (j'_1,k'_1)(k'_1,j'_2)(j'_2,k'_2)(k'_2,j'_1)+\\ &\quad (j'_1,k'_2)(k'_2,j'_3)(j'_3,k'_3)(k'_3,j'_1)\end{aligned}$$

In the same way, we can consider any move $m \in M_\nu$ and produce the corresponding sub-graph



This sub-graph can be written with the circuit

$$(j'_1,k'_1)(k'_1,j'_2),\cdots,(j'_s,k'_s)(k'_s,j'_{s+1}),\cdots,(j'_\nu,k'_\nu)(k'_\nu,j'_1)$$

This circuit can be explicitly decomposed by the $\nu-1$ following basic moves:

$$\begin{array}{cccc}(j'_1,k'_1) & (k'_1,j'_2) & (j'_2,k'_2) & (k'_2,j'_1)\\ (j'_1,k'_2) & (k'_2,j'_3) & (j'_3,k'_3) & (k'_3,j'_1)\\ \vdots & \vdots & \vdots & \vdots\\ (j'_1,k'_\nu) & (k'_\nu,j'_{\nu+1}) & (j'_{\nu+1},k'_{\nu+1}) & (k'_{\nu+1},j'_1)\end{array}$$

Hence for any given $m \in M^*$, we can find the decomposition, as illustrated, in terms of basic moves which form the main move.

Proposition 2.2 *If $t^{(1)}_{.jk} \geq 1$, for all j and k, then using M_4 as the set of moves in the algorithm described in chapter* 1 *leads to an irreducible Markov chain.*

Proof We need to prove that for a realisation T in which $t_{.jk} \geq 1$ and $T + m \in \mathcal{S}$, where $m \in M^*$, there exists a sequence of basic moves in M_4 which moves T to $T + m$, while all intermediate tables produced by $T + m^i$, $m^i \in M_4$ remain non-negative, since a negative element is equivalent to exiting from S.

We define a new table T' from the m and T in the following way:

$$t'_{ijk} = \begin{cases} +1 & \text{if } m_{ijk} = -1 \\ \min(1, t_{ijk}) & \text{if } m_{ijk} = 0 \text{ and } t_{1jk}\, t_{2jk} = 0 \\ 0 & \text{if } m_{ijk} = 1 \end{cases} \tag{2.2}$$

For the remaining cells undefined yet, we set $t'_{1jk} = 1$ and $t'_{2jk} = 0$. So T' has binary entries and $t'_{.jk} = 1$. Considering the above explanation, T' corresponds to Rasch table and so does $T' + m$. T' has no negative entries as we define $t'_{ijk} = 1$ when $m_{ijk} = -1$. Applying the result from Ryser(1963), there is a sequence of basic moves taking us from T' to T'+m while preserving non-negativity along the way. As we have $t_{ijk} \geq t'_{ijk}$, the same sequence of basic moves can be applied to connect T and $T + m$ while staying in S.

□

Example:

For more illustration consider the following $2 \times 3 \times 3$ table, T.

1	2	2
4	2	3
0	2	1

1	1	3
3	1	2
1	4	5

The marginal sum $t_{.jk}$ is

2	3	5
7	3	5
1	6	6

Clearly we have $t_{.jk} \geq 1$. Now, consider the following move

-1	1	0
1	0	-1
0	-1	1

1	-1	0
-1	0	1
0	1	-1

The corresponding T', as defined in (2.2) is

1	0	1
0	1	1
0	1	0

0	1	0
1	0	0
1	0	1

The T' has been defined in such a way that $t'_{.jk} = 1$, for all j and k. This guarantees that T' is a Rasch table. In our example the Rasch table representation of the $2 \times J \times K$ table T' is as follows.

	k	i=1	i=2
j=1	1	1	0
	2	0	1
	3	1	0
j=2	1	0	1
	2	1	0
	3	1	0
j=3	1	0	1
	2	1	0
	3	0	1

Since $m \in M^*$, so $T' + m$ keeps the row and column marginal sums unchanged, therefore $T' + m$ is a Rasch table as well. Also non-negativity holds in $T' + m$, since we choose $t'_{ijk} = +1$ whenever $m_{ijk} = -1$. In our example $T' + m$ is

0	1	1
1	1	0
0	0	1

1	0	0
0	0	1
1	1	0

Ryser's result tells us that we can decompose m into a sequence of basic moves taking us from T' to T'+m without creating a negative entry. In our example $m \in M_6$ and so it can be decomposed by a sequence of length 2 of basic moves m^1 and m^2 so that $m = m^1 + m^2$ and $T' + m^1$ has no negative entries.

-1	1	0
1	-1	0
0	0	0

1	-1	0
-1	1	0
0	0	0

And m^2 is as follows

0	0	0
0	1	-1
0	-1	1

0	0	0
0	-1	1
0	1	-1

It is easily seen that the same two basic moves take us from T to $T+m$ without leaving $\mathcal{S}$.

□

Proposition 2.3

If $T, T+m \in \mathcal{S}$, where $m \in M^*$ then there exists a path, using moves in M_4, that connects T to $T+m$ and that does not leave $\mathcal{S}'$.

Proof If we consider a move, $m \in M^*$, proposition 2.1 tells us that we can decompose it into a sequence of basic moves. In the proof of proposition 2.1 it has been shown that the sequence looks like the following circuit

$$(j_1', k_1')(k_1', j_2')(j_2', k_2')(k_2', j_1')$$

$$(j_1', k_2')(k_2', j_3')(j_3', k_3')(k_3', j_1')$$

$$\vdots$$

$$(j_1', k_{\nu-1}')(k_{\nu-1}', j_\nu')(j_\nu', k_\nu')(k_\nu', j_1')$$

Note that (k_2', j_1') may cause -1 which will be returned to 0 by (j_1', k_2'), and again (k_3', j_1') may cause -1 which will changed to 0 by the next one and this might happen several times in the circuit.

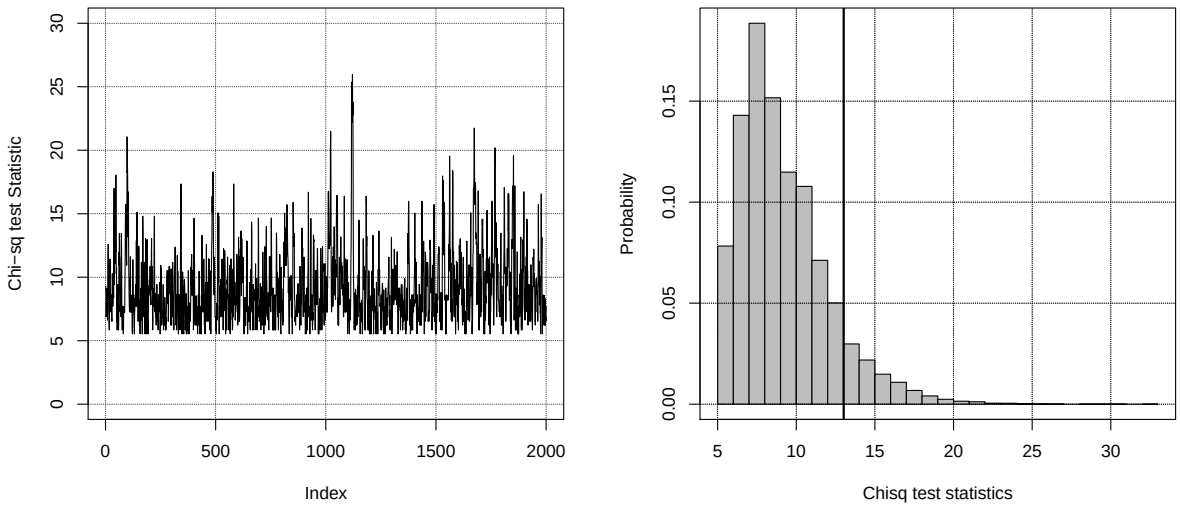
We write each basic move as

$$(j_1', k_s')(k_s', j_{s+1}')(j_{s+1}', k_{s+1}')(k_{s+1}', j_1')$$

Consider two sequential basic moves

$$(j_1', k_{s-1}') \quad (k_{s-1}', j_s') \quad (j_s', k_s') \quad (k_s', j_1')$$
$$(j_1', k_s') \quad (k_s', j_{s+1}') \quad (j_{s+1}', k_{s+1}') \quad (k_{s+1}', j_1')$$

Figure 2.2: *The trace plot of the chi-squared statistics computed for the generated tables by Bunea and Besag's algorithm (left); Histogram of the computed the chi-squared statistics for generated tables, where the vertical line is the statistics for the observed table (right).*



The top layers of the basic moves can be represented as follow

	+1		-1			
	-1		+1			

			+1		-1	
			-1		+1	

As we have shown, the intermediate table could have at worst a single -1 which gets modified later.

□

Example: By the definition of S', we consider the example using the Bunea and Besag's algorithm to illustrate how this method works for a specific table with the `R` software. Consider the following observed $2 \times 3 \times 3$ table. We aim to test the hypothesis that there

is no three-way interaction.

6	2	4
4	6	1
3	1	4

2	4	6
5	3	4
3	7	3

We produced an `R` function, see appendix A.1, which computes the p-value for the hypothesis of no three-way interaction using Bunea and Besag 's algorithm. We run the algorithm which allows moves to go into $\mathcal{S}'$. Running the algorithm for 100,000 iterations gives us the estimated p-value= 0.0930, weak evidence of a three-way interaction, and also Figure 2.2 (left) shows the trace plot of the chi-squared statistics computed for the generated tables. Figure 2.2 (right) shows the histogram of the computed chi-squared statistics. The vertical line in the plot depicts the chi-squared statistic for the observed table.

2.2 Direct proof that M^* is a Markov basis

Bunea and Besag (2000) used M^* as an irreducible set of moves for $\mathcal{S}$ but they cite Diaconis and Sturmfels(1998) which uses a Grobner basis argument which does not easily generalise to $I \times J \times K$ tables with $I > 2$.

Here, it is shown directly that M^* is a Markov basis. Since the difference, D, of two tables is non zero, $D \neq 0$ and

$$d_{i.k} = \sum_k d_{ij.} = 0 \tag{2.3}$$

there exist some j and k so that $d_{1jk} > 0$. We relabel the rows and columns so that:

$$d_{111} > 0$$

and we start to draw a picture of the pattern of cells in D:

$$+$$

According to the constraint (2.3), $d_{11.} = 0$ and so there exists $k > 1$ so that $d_{11k} < 0$. By relabelling columns 2 to K so that column k becomes column 2:

$$d_{112} < 0$$

$$+ \qquad -$$

Again, based on the constraint (2.3) $d_{1.2} = 0$, there exists $j > 1$ so that $d_{1j2} > 0$.

By relabelling rows 2 to J so that row j becomes row 2.

$$\begin{array}{cc} d_{122} > 0 & \\ + & - \\ d_{121} & + \end{array}$$

Now, consider d_{121}; if $d_{121} < 0$, as we already have $d_{111}, d_{122} > 0$ and $d_{112} < 0$, we have half a basic move and because $d_{1ij} = -d_{2ij}$, thus we have found $m \in M_4$:

$$\begin{array}{cc} + & - \\ - & + \end{array}$$

If not then $d_{121} \geq 0$, which gives:

$$\begin{array}{cc} + & - \\ \geq & + \end{array}$$

Since the sum of the first two entries in row 2 is positive, there exists $k > 2$ such that $d_{12k} < 0$ to satisfy the constraint (2.3). By relabeling columns 3 to k we have:

$$\begin{array}{lccc} d_{123} < 0 & + & - & d_{113} \\ & \geq & + & - \end{array}$$

The same approach is now applied to d_{113}: if $d_{113} > 0$, we have $m \in M_4$ based on d_{112}, d_{113}, d_{122} and d_{123}. Otherwise $d_{113} \leq 0$ and we have

$$\begin{array}{ccc} + & - & \leq \\ \geq & + & - \end{array}$$

Since, $d_{1.3} = 0$ there exist $j > 2$ so that $d_{1j\,3} > 0$ and relabelling we have

$$\begin{array}{ccc} + & - & \leq \\ \geq & + & - \\ d_{131} & d_{132} & + \end{array}$$

The same argument applied to d_{131} and d_{132} either gives us a move or they are both greater than or equal to zero. If $d_{131} < 0$ then we have found $m \in M_6$; if $d_{132} < 0$ then we have found $m \in M_4$; otherwise $d_{131} \geq 0$ and $d_{132} \geq 0$ and the situation looks like:

$$\begin{array}{ccc} + & - & \leq \\ \geq & + & - \\ \geq & \geq & + \end{array}$$

Based on the same argument, in the next step D looks like:

$$\begin{array}{cccc} + & - & \leq & \leq \\ \geq & + & - & \leq \\ \geq & \geq & + & - \\ d_{141} & d_{142} & d_{143} & + \end{array}$$

d_{144} must be positive for constraint 2.3 to hold. The same argument is applied to d_{14t} where $t = 1, 2, 3$ so:

$$\begin{aligned} d_{143} < 0 &\implies m \in M_4 \\ d_{142} < 0 &\implies m \in M_6 \\ d_{141} < 0 &\implies m \in M_8 \end{aligned}$$

otherwise $d_{14t} \geq 0$ where $t = 1, 2, 3$.

The same approach can be applied to generalise the argument in top-left $m \times m$ square of D. If we haven't found a move in the top left $m \times m$ square of D then we have the following pattern:

$$\begin{array}{ccccccc}
+ & - & \leq & \leq & \cdots & & \leq \\
\geq & + & - & \leq & \cdots & & \leq \\
\geq & \geq & + & - & \cdots & & \leq \\
\geq & \geq & \geq & + & \cdots & & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\
 & & & & & & - \\
\geq & \geq & \cdots & \geq & \cdots & & d_{1mm} > 0
\end{array}$$

By adding another column there should be $-$ in d_{1mm+1}. Now, we consider all d_{1jm+1} for $j \leq m$. Depending on which one is $+$ there exists a move with different degree, if none are $+$ there is no move, then $d_{1jm+1} \leq 0$ for $j \leq m$ so by relabelling rows $d_{1m+1m+1} > 0$. Considering d_{1m+1k} for $k \leq m$ will provide a move if any d_{1m+1k} is $-$ so we assume $d_{1m+1k} \geq 0$ for $k \leq m$ which means we haven't found a move in the top left $(m+1) \times (m+1)$ square of D.

Here we would start again with $(m+1) \times (m+1)$ top left square. Since $m \leq \min(J, K)$, there must be a move.

The following two lemmas show that, each time a move is added to D, it gets smaller in at least eight cells, by adding the move to make the table $T^{\#} = T + m$, and it doesn't go out of $\mathcal{S}$. Consequently, there is a path in $\mathcal{S}$ from T to T^* made by a finite sequence of moves in M^*.

Lemma 2.4 *Suppose that there is some $m \in M^*$ such that: $d_{ijk} > 0$ if $m_{ijk} > 0$ and $d_{ijk} < 0$ if $m_{ijk} < 0$. Then $T^{\#} \in S$ where $T^{\#} = T + m$.*

Proof If $m_{ijk} > 0$, $m_{ijk} = +1$, and $d_{ijk} > 0$. Hence, $t^*_{ijk} \geq 0$ and $t^{\#}_{ijk} = t_{ijk} + 1$ implies $t^{\#}_{ijk} \geq 0$. On the other hand, if $m_{ijk} < 0$, $m_{ijk} = -1$, and $d_{ijk} < 0$. Therefore

$$t^*_{ijk} < t_{ijk} \tag{2.4}$$

and so

$$0 \leq t^*_{ijk} \leq t_{ijk} - 1 = t^{\#}_{ijk} \tag{2.5}$$

Having $0 \leq t^{\#}_{ijk}$ provides $T^{\#} \in S$.

□

Lemma 2.5 *Writing $D^* = T^* - T^{\#} = D - m$:*

- *If* $D^* = 0$ *then we have a basic move from* T *to* T^*.
- *If* $D^* \neq 0$ *then* $\sum \mid d^*_{ijk} \mid \leq \sum \mid d_{ijk} \mid -8$ *and so the the size of* D*, which can not be negative, decreases by at least* 8

Proof When $D^* = 0$ then $T^* = T^{\#}$ so $T^* = T + m$ hence there exist a move which makes connection between two tables. In the case of $D^* \neq 0$, since $D^* = D - m$, for every cell $d^*_{ijk} = d_{ijk} - m_{ijk}$. When $m_{ijk} > 0$, $m_{ijk} = +1$ and $d_{ijk} > 0$. Hence, $|d^*_{ijk}| = |d_{ijk} - 1| = d_{ijk} - 1$ and so $|d^*_{ijk}| = |d_{ijk}| - 1$. If $m_{ijk} < 0$, $m_{ijk} = -1$ and $d_{ijk} < 0$. Hence

$$|d^*_{ijk}| = |d_{ijk} + 1| = -(d_{ijk} + 1) = |d_{ijk} \mid -1.$$

Having proved $|d^*_{ijk}| = |d_{ijk}| - 1$ for all cells where $m_{ijk} \neq 0$,

$$\sum_{ijk} |d^*_{ijk}| = \sum_{ijk} |d_{ijk}| - \sum_{ijk} |m_{ijk}| \tag{2.6}$$

$$\leq \sum_{ijk} |d_{ijk}| - 8 \tag{2.7}$$

since m has at least 8 non-zero cells.

□

2.3 Direct justification of Bunea and Besag using only basic moves

In $2 \times J \times K$ tables, it is always possible to find a path in S' from T to T^* using only basic moves. In this section a direct proof will be given that the Bunea and Besag algorithm creates an irreducible Markov chain over $\mathcal{S}$. Note that M^*, results from Diaconis and Sturmfels and the Rasch Model are not used in our proof. The following constraints have been considered in this proof:

1. $t_{.jk} \geq 0$ and $t_{i.k} \geq 0$ and $t_{ij.} \geq 0$.
2. $\sum_i t^*_{ijk} \geq 0$ and $\sum_i t^*_{ijk} \geq 0$ and $\sum_i t^*_{ijk} \geq 0$.
3. $\sum_i d_{ijk} = \sum_j d_{ijk} = \sum_k d_{ijk} = 0$

Theorem 2.6 *In $2 \times J \times K$ tables, it is always possible to find a path in S' from T to T^* using only basic moves using the following Algorithm*

(1) If $D = 0$ then algorithm stops.

(2) Find a positive entry, $d_{ijk} > 0$ in D.

(3) According to constraint (3), there should be a negative cell $d_{ijk'}$ in the same row and slice as d_{ijk}. The constraint also implies one other negative cell in the same column and slice as d_{ijk} located in $d_{ij'k}$. Now, we set a basic move according to these three mentioned cells in D such that $m_{ijk} = +1$, $m_{ijk'} = -1$ and $m_{ij'k} = -1$ which implies that $m_{ij'k'} = +1$, $m_{i'jk} = -1$, $m_{i'jk'} = +1$, $m_{i'j'k} = +1$ and $m_{i'j'k'} = -1$, where i' is the other slice.

(4) Let $T' = T + m$ and $D' = D - m$.

(5) Different possibilities might happen to $t_{i'j'k'}$.

- *If $t_{i'j'k'} > 0$ then $T + m \in S$ so go back to step (1) with $T = T'$ and $D = D'$.*
- *If $t_{i'j'k'} = 0$ then $T + m \in S' \setminus S$, ($t'_{i'j'k'} = -1$). In this case $d'_{i'j'k'} > 0$. So go back to step (3) with $T = T'$, $D = D'$ and $d'_{i'j'k'}$ as the positive entry.*

We need to prove the following statements.

- If $t'_{i'j'k'} = -1$ then $d'_{i'j'k'} > 0$.
- All the other cells in T' are non-negative.
- The algorithm stops.

Proof • $t'_{i'j'k'} = -1$ when $t_{i'j'k'} = 0$ and $m_{i'j'k'} = -1$ so

$$\begin{aligned} d'_{i'j'k'} &= d_{i'j'k'} - m_{i'j'k'} \\ &= t^*_{i'j'k'} - t_{i'j'k'} - m_{i'j'k'} \\ &= t^*_{i'j'k'} + 1 \end{aligned}$$

and because $T^* \in S$ then $d'_{i'j'k'}$ is always greater than zero.

- The only cells which need to be considered are $t'_{i'jk}$, $t'_{ijk'}$ and $t'_{ij'k}$ because the corresponding cells in m are -1. But the corresponding cells in D are negative which means:

$$0 \leq t^*_{i'jk} < t_{i'jk} = t'_{ijk} + 1 \Rightarrow t_{i'jk} \geq 0$$

  and similarly for $t_{ijk'}$ and $t_{ij'k}$

- Because sign of m matches sign of D for certain in their 6 cells so each time the move is subtracted from D the cells d_{ijk}, $d_{ij'k}$, $d_{ijk'}$, $d_{i'jk}$, $d_{i'j'k}$ and $d_{i'jk'}$ get smaller. It may cause to increase or decrease the values of the cells, $d_{ij'k'}$ and $d_{i'j'k'}$ so that

$$\mid D' \mid \leq \mid D \mid -6 + 2$$

$$= \mid D \mid -4$$

  Note that $\mid D' \mid \geq 0$ so that it takes a maximum of $\lfloor \frac{|D|}{4} \rfloor$ steps for algorithm to stop.

□

2.4 Conclusion

Bunea and Besag (2000) provided an algorithm using the M^* introduced by Diaconis and Sturmfels (1998), to make an irreducible Markov chain over the sample space, defining an extra space, S'. The algorithm is only introduced for $2 \times J \times K$ tables using the Rasch model idea. To prove why the chain is irreducible they have referred to Diaconis and Sturmfels (1998). We have shown the reason why M^* is a Markov basis and we have also proved that the connected Markov chain can be constructed using only basic move allowing a single -1 in the tables which helps to avoid the need for more complicated moves of higher degrees.

Chapter 3

Algorithms for 3×3×K and 3×4×K tables

3.1 Introduction

Diaconis and Sturmfels (1998) introduced a general algorithm for computing a Markov basis based on the existence of a Grobner basis. Their approach is extremely appealing as it can be used for any dimension. However, for two main reasons, using their approach is limited. The more important is the computational complexity of computing Grobner basis, as the Grobner basis is different for each table not just each size of table. The other is that it involves many redundant basis elements and a reduced basis lacks symmetry.

Therefore, Aoki and Takemura (2003) suggested using the unique minimal Markov basis which is defined in the following section.

Before continuing, by introducing Aoki and Takemura (2003)'s minimal Markov basis, we will prove the following theorem which shows the difference between any two $I \times J \times K$ tables can be made from basic moves.

Theorem 3.1 *The difference of two tables, $\mathcal{D}$, can be made from a set of basic moves.*

Proof Consider the following constraint

$$d_{.jk} = d_{i.k} = d_{ij.} = 0$$

In section 2.2, direct proof that M^* is Markov basis, it has been shown that we are always able to find in the first layer of D a pattern of $+$ and $-$ entries which matches one layer

of a move $m \in M^*$. It is also the case that to satisfy the constraint there must be a $-$ in of the other layers which matches the $+$ in d_{111}.

If we complete the half-move by filling in the opposite signs in the layer where the $-$ was found, we obtain a move m in M^*. The sign of D matches m in more than half of the non-zero cells of m. Consequently, the size of D decreases when we make the move.

As we know each $m \in M^*$ is decomposable in to a set of basic moves. Hence, by subtracting each basic move from D we can make the size of D smaller. We can then repeat the process of finding half a move in M^* until the first layer becomes zero. The same argument is applied to the other layers. So by subtracting the sequence of basic moves D becomes zero.

So it has been shown that $\mathcal{D}$ is made from a set of basic moves.

□

3.2 Aoki and Takemura approach

In this section we briefly review Aoki and Takemura (2003) and introduce their minimal Markov basis for different tables. A Markov basis is defined as follows:

Definition *Markov basis*

A Markov basis is a set $\mathcal{B} = \{m_1, \cdots m_L\}$ of $I \times J \times K$ integer arrays $m_L \in M$,where $\mathcal{M}$ is the set of all moves, such that for any $\{t_{ij.}\}$, $\{t_{i.k}\}$, $\{t_{.jk}\}$ and $T, T' \in \mathcal{S}$, there exist $A > 0$, $(\varepsilon_1, m_{q_1}), \cdots, (\varepsilon_A, m_{q_A})$ with $\varepsilon_s = \pm 1$ and $m_{q_s} \in \mathcal{B}$ such that

$$T' = T + \sum_{s=1}^{A} \varepsilon_s m_{q_s} \quad \text{and} \quad T + \sum_{s=1}^{a} \varepsilon_s m_{q_s} \in \mathcal{S} \quad \text{for} \quad 1 \leq a \leq A.$$

A Markov basis $\mathcal{B}$ is minimal if no proper subset of $\mathcal{B}$ is a Markov basis. A minimal Markov basis is said to be unique if there exists only one minimal Markov basis.

If a Markov basis is obtained, a connected Markov chain over $\mathcal{S}$ is easily constructed.

Aoki and Takemura (2003) derived the unique minimal Markov basis, described as a set of different types of move, for $3 \times 3 \times K$ tables where $K = 3$, $K = 4$ and $K \geq 5$. Before introducing the theorems, a brief explanation of different types of move will be given.

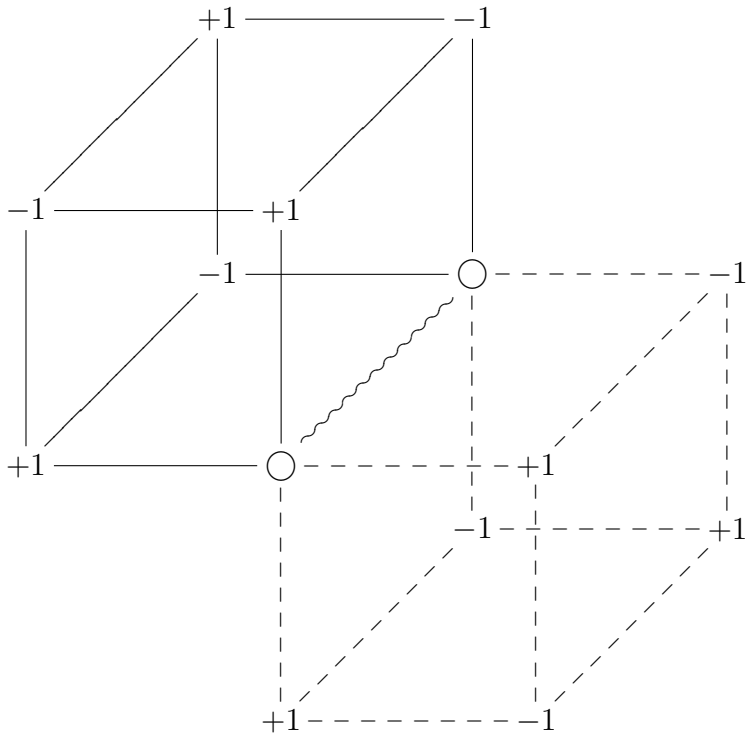
The most elementary type of move, already discussed in the introduction, is the basic move. Here we use the notation $m_4(i_1i_2; j_1j_2; k_1k_2)$ which indicates that the move is made by choosing two layers, two rows and two columns in which all the non-zero cell entries lie: $m_{i_1j_1k_1} = m_{i_1j_2k_2} = m_{i_2j_1k_2} = m_{i_2j_2k_1} = +1$ and $m_{i_2j_1k_1} = m_{i_1j_2k_1} = m_{i_1j_1k_2} = m_{i_2j_2k_2} = -1$.

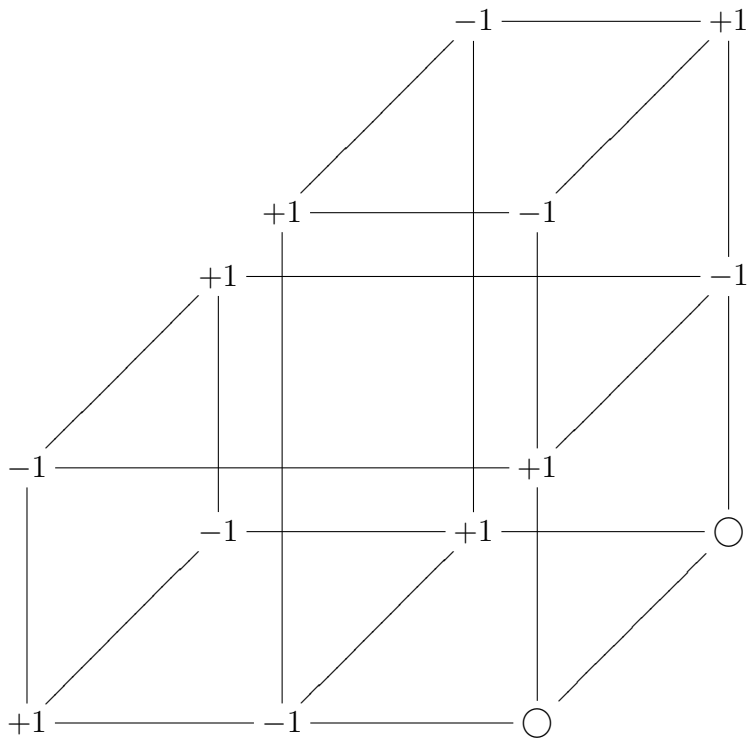
One type of move of degree 6, having non-zero cells in only two layers and three rows and columns, is a $3 \times 3 \times K$ integer array $m_6^I(i_1i_2; j_1j_2j_3; k_1k_2k_3)$ with elements

$$m_{i_1j_1k_1} = m_{i_1j_2k_2} = m_{i_1j_3k_3} = m_{i_2j_1k_2} = m_{i_2j_2k_3} = m_{i_2j_3k_1} = 1,$$

$$m_{i_1j_1k_2} = m_{i_1j_2k_3} = m_{i_1j_3k_1} = m_{i_2j_1k_1} = m_{i_2j_2k_2} = m_{i_2j_3k_3} = -1,$$

Similarly, there exist $m_6^J(i_1i_2i_3; j_1j_2; k_1k_2k_3)$ and $m_6^K(i_1i_2i_3; j_1j_2j_3; k_1k_2)$ which have non-zero entries respectively in just two rows and two columns. These moves are called two-step moves because they can be implemented as as two basic moves, as indicated in the following graphical representations of moves in M_6:



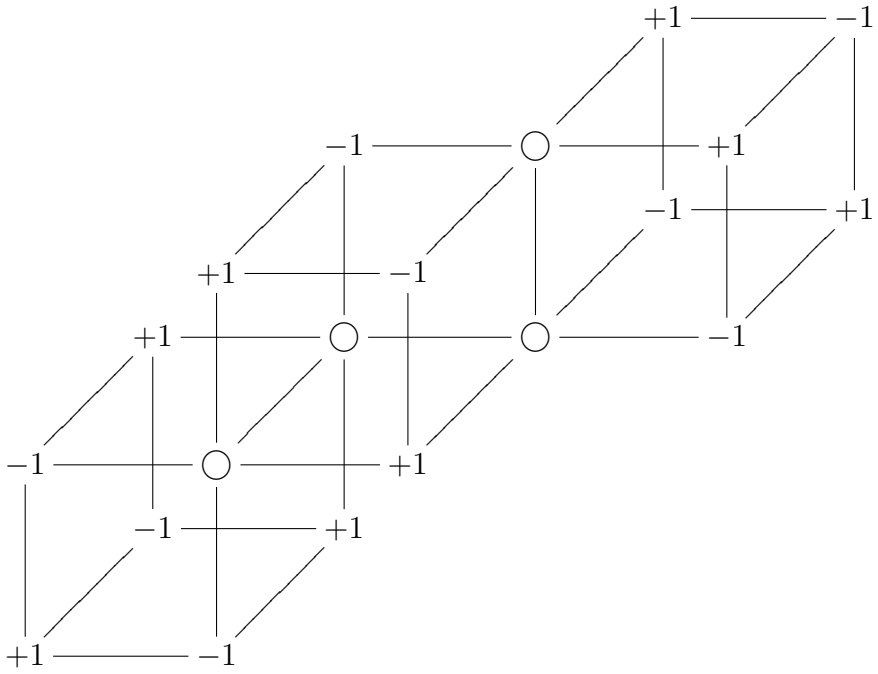


The next type of move is a move of degree 8, which is a three-step move. For the case of a general $I \times J \times K$ contingency table, there are several versions of such a move. However, for the $3 \times 3 \times K$ case, Aoki and Takemura need only the following type which has non-zero cells in three layers, three rows and four columns: $m_8(i_1 i_2 i_3; j_1 j_2 j_3; k_1 k_2 k_3 k_4)$ where

$$m_{i_1 j_1 k_1} = m_{i_1 j_2 k_2} = m_{i_2 j_1 k_3} = m_{i_2 j_2 k_1} = m_{i_2 j_3 k_4} = m_{i_3 j_1 k_2} = m_{i_3 j_2 k_4} = m_{i_3 j_3 k_3} = 1,$$

$$m_{i_1 j_1 k_2} = m_{i_1 j_2 k_1} = m_{i_2 j_1 k_1} = m_{i_2 j_2 k_4} = m_{i_2 j_3 k_3} = m_{i_3 j_1 k_3} = m_{i_3 j_2 k_2} = m_{i_3 j_3 k_4} = -1,$$

The following figure shows a three dimensional view of one such move, indicating how it can be constructed from three basic moves:

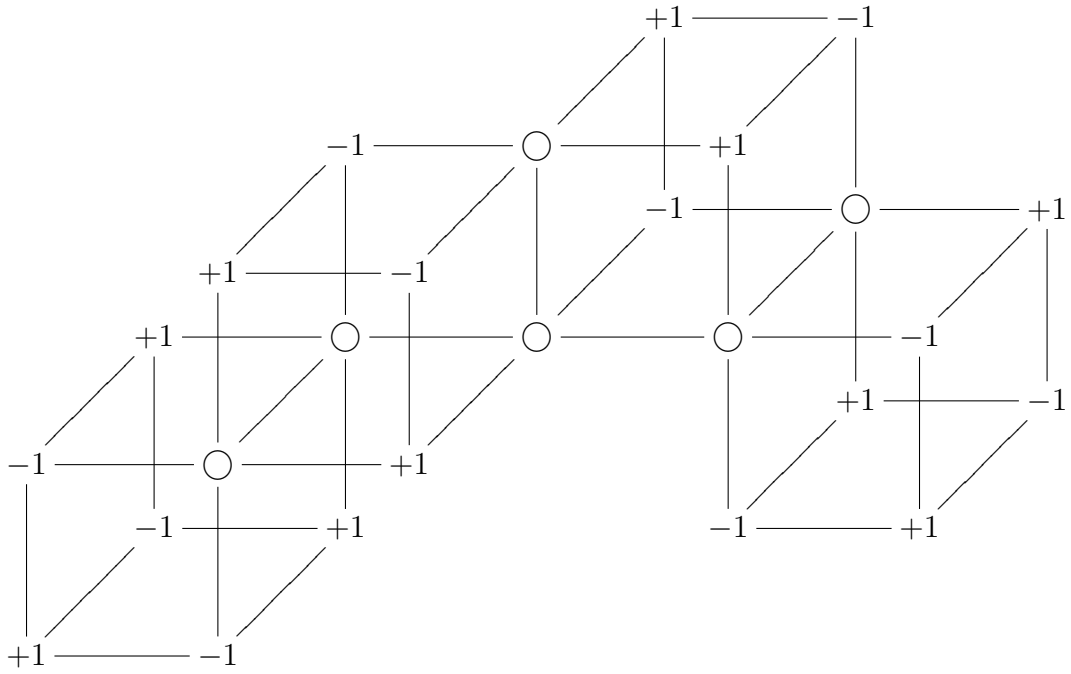


Finally, Aoki and Takemura require one type of move of degree 10, $m_{10}(i_1i_2i_3; j_1j_2j_3; k_1k_2k_3k_4k_5)$ which has non-zero cells in three layers, three rows and 5 columns:

$$m_{i_1j_1k_1} = m_{i_1j_2k_2} = m_{i_1j_2k_5} = m_{i_1j_3k_4} = m_{i_2j_1k_3} = m_{i_2j_2k_1} = m_{i_2j_3k_5} = m_{i_3j_1k_2} = m_{i_3j_2k_4} = m_{i_3j_3k_3} = 1,$$

$$m_{i_1j_1k_2} = m_{i_1j_2k_1} = m_{i_1j_2k_4} = m_{i_1j_3k_5} = m_{i_2j_1k_1} = m_{i_2j_2k_5} = m_{i_2j_3k_3} = m_{i_3j_1k_3} = m_{i_3j_2k_2} = m_{i_3j_3k_4} = -1,$$

The following three dimensional view of one such move shows that these are four-step moves constructed from four basic moves:



The following theorems from Aoki and Takemura (2003) determerine the unique minimal Markov basis for $3 \times 3 \times K$ tables for the cases $K = 3$, $K = 4$ and $K \geq 5$.

Theorem 3.2 *The set of all basic moves* $m_4(i_1i_2, j_1j_2, k_1k_2)$ *and the three different types of move of degree* 6*,*

$$m_6^I(i_1i_2, j_1j_2j_3, k_1k_2k_3), m_6^J(i_1i_2i_3, j_1j_2, k_1k_2k_3), m_6^K(i_1i_2i_3, j_1j_2j_3, k_1k_2)$$

constitute the unique minimal Markov basis for $3 \times 3 \times 3$ *tables.*

Theorem 3.3 *The set of all basic moves* $m_4(i_1i_2, j_1j_2, k_1k_2)$*, the three different types of move of degree* 6

$$m_6^I(i_1i_2, j_1j_2j_3, k_1k_2k_3), m_6^J(i_1i_2i_3, j_1j_2, k_1k_2k_3), m_6^K(i_1i_2i_3, j_1j_2j_3, k_1k_2)$$

and one type of move of degree 8*:*

$$m_8(i_1i_2i_3, j_1j_2j_3, k_1k_2k_3k_4)$$

constitute the unique minimal Markov basis for $3 \times 3 \times 4$ *tables.*

Theorem 3.4 *The set of all basic moves* $m_4(i_1i_2, j_1j_2, k_1k_2)$*, the three different types of move of degree* 6*,*

$$m_6^I(i_1i_2, j_1j_2j_3, k_1k_2k_3), m_6^J(i_1i_2i_3, j_1j_2, k_1k_2k_3), m_6^K(i_1i_2i_3, j_1j_2j_3, k_1k_2)$$

one type of move of degree 8*,*

$$m_8(i_1i_2i_3, j_1j_2j_3, k_1k_2k_3k_4))$$

and one type of move of degree 10*,*

$$m_{10}(i_1i_2i_3, j_1j_2j_3, k_1k_2k_3k_4k_5)$$

constitute the unique minimal Markov basis for $3 \times 3 \times K$ *tables for* $K \geq 5$*.*

In the following sections, we show that connected Markov chains can be constructed over $\mathcal{S}$ using only basic moves, by allowing the chain to visit tables outside $\mathcal{S}$ having up to two cells containing -1.

3.3 Construction of a connected Markov chain over 3×J×K tables

It will be shown that without using the unique minimal Markov basis introduced by Aoki and Takemura (2003), we can still construct an irreducible Markov chain over $\mathcal{S}$. This is done by extending the idea of Bunea and Besag (2000). They allow tables to have at most a single -1 cell; we must allow them to have at most two.

Theorem 3.5 *The Bunea and Besag algorithm applied to* $3 \times 3 \times K$ *tables creates an irreducible Markov chain on* S' *provided we re-define* S' *to include tables having at most two negative cells which are* -1*.*

Proof Let $T \in S'$ have at most a single -1 cell. As before, $T^* \in S$, $D = T^* - T$ and we refer to $\sum_{ijk} |d_{ijk}|$ as the size of D.

The idea of the proof is to show that we can always find either a single basic move or a sequence of two basic moves which, when applied to T, reduce the size of D and leave $T \in S'$ and having at most a single -1 cell. We do so by appropriately relabelling rows, columns and layers and then exhaustively considering possible configurations of positive and negative cells in the first few rows and columns of D. The consequence is that there must be a path, made from basic moves, in the re-defined S' between any two tables in S.

For the first part of the proof we work with general J rather than $J = 3$ so that we can use it as the starting point of the proof of the subsequent theorem concerning $3 \times 4 \times 4$ tables.

(1) If T has a negative cell, we bring it to t_{111} by relabelling rows, columns and layers and then d_{111} must be positive; otherwise, find any positive cell in D and bring it to d_{111} by relabelling.

(2) The constraint implies a negative cell in row 1 of layer 1 and we bring it to d_{112} by relabelling columns 2 to K. The constraint also implies a negative cell in column 1 in layer 1 and we bring it to d_{121} by relabelling rows 2 to J. Finally the constraint implies a negative cell in row 1 and column 1 of another layer and we bring it to d_{211} by relabelling layers 2 and 3. Writing $+$ for a positive cell and $-$ for a negative cell. the cells we have identified in the first two layers of D look like

$$\begin{array}{cc|c} + & - & - \\ - & & \end{array}$$

(3) If $d_{122} > 0$ then we can make the move

$$\begin{array}{cc|cc} + & - & - & + \\ - & + & + & - \end{array}$$

and the size of D gets smaller as there are at least five cells where d_{ijk} has the same sign as m_{ijk} and at most three where the sign can differ.

Here, the only possible cell in T which might become negative is t_{222}, as it is the cell which corresponds to the unmatched $-$. Moreover, if t_{111} was negative, it ceases to be. Therefore, having made the move, $T \in S'$ and has at most one -1 cell. We return to step (1) with the new T.

(4) Now suppose that $d_{122} \leq 0$ so that D looks like

$$\begin{array}{cc|c} + & - & - \\ - & \leq 0 & \end{array}$$

The same argument that we used in step (2) applies to d_{221} and d_{212}. Therefore the only case which can cause us difficulties is where we also we have $d_{221} \geq 0$ and $d_{212} \geq 0$ so that D looks like

$$\begin{array}{cc|cc} + & - & - & \leq 0 \\ - & \leq 0 & \leq 0 & \end{array}$$

(5) If $d_{222} < 0$ then we can simply make the same basic move as we end up with no negative cells in T and then $T \in S$. The size of D gets smaller because the signs of at least five cells match. We return to step (1) with the new T.

(6) The only remaining case is where $d_{222} \geq 0$ so that D now looks like

$$\begin{array}{cccc} + & - & - & \leq 0 \\ - & \leq 0 & \leq 0 & \geq 0 \end{array}$$

Here, the constraint implies a positive cell in the second row of layer one to keep the marginal total equal to zero . By relabelling columns 3 to K, we can bring it to d_{123} so that D looks like

$$\begin{array}{cccccc} +(+) & - & (-) & -(-) & \leq 0 & (+) \\ -(-) & \leq 0 & +(+) & \leq 0(+) & \geq 0 & (-) \end{array}$$

where the signs in the brackets indicate the move we plan to make.

If $d_{113} < 0$ we can make that basic move and the size of D gets smaller because the signs of at least five cells match. Here the only possible cell which might become negative is t_{223} and t_{111} cannot stay negative. We return to step (1) with the new T.

(7) When $d_{113} \geq 0$, D looks like

$$\begin{array}{ccccc} + & - & \geq 0 & - & \leq 0 \\ - & \leq 0 & + & \leq 0 & \geq 0 \end{array}$$

If $d_{223} < 0$ then we can make the basic move with -1 in m_{121}, m_{113}, m_{211} and m_{223} and the size of D gets smaller because at least five signs match. The only possible cell in T which can be left negative is t_{113}. We return to step (1).

(8) When $d_{223} \geq 0$, D looks like

$$\begin{array}{ccccccc} + & - & \geq 0 & - & \leq 0 & \\ - & \leq 0 & + & \leq 0 & \geq 0 & \geq 0 \end{array}$$

Here, the constraint implies that there is a positive cell in the second column of the first layer and by relabelling rows 3 to J that cell is d_{132}. D now looks like:

$$\begin{array}{ccccccc} +(+) & -(-) & \geq 0 & -(-) & \leq 0(+) & \\ - & \leq 0 & + & \leq 0 & \geq 0 & \geq 0 \\ (-) & +(+) & & (+) & (-) & \end{array}$$

Now, if $d_{131} < 0$ then we can make the move and the size of D gets smaller and the only possible cell to remain negative in T is t_{232}. We return to step (1).

(9) When $d_{131} \geq 0$, D looks like

$$\begin{array}{cccccc} + & - & \geq 0 & - & \leq 0 & \\ - & \leq 0 & + & \leq 0 & \geq 0 & \geq 0 \\ \geq 0 & + & & & & \end{array}$$

If $d_{232} < 0$ then we can make the basic move with -1 in m_{131}, m_{112}, m_{211} and m_{232} and the size of D gets smaller. The only cell in T which can remain negative is t_{131} and we return to step (1).

(10) When $d_{232} \geq 0$, D looks like

$$\begin{array}{cccccc} + & - & \geq 0 & - & \leq 0 & \\ - & \leq 0 & + & \leq 0 & \geq 0 & \geq 0 \\ \geq 0 & + & & & \geq 0 & \end{array}$$

Here, there are no more conclusions in the first and second layer to make. Since there are only three layers, we can use the constraint to draw conclusions about some cells in the third layer and the three layers of D now look like

$$\begin{array}{ccccccccc} + & - & \geq 0 & - & \leq 0 & & & + & \\ - & \leq 0 & + & \leq 0 & \geq 0 & \geq 0 & + & & - \\ \geq 0 & + & & & \geq 0 & & & - & \end{array}$$

(11) Here, for $J > 3$ we can't make any statement but for $J = 3$ we can determine the sign of d_{133} and D looks like

$$\begin{array}{ccccccccc} + & - & \geq 0 & - & \leq 0 & & & + & \\ - & \leq 0 & + & \leq 0 & \geq 0 & \geq 0 & + & & - \\ \geq 0 & + & - & & \geq 0 & & & - & \end{array}$$

We can see that, with the exception of d_{311}, we have the right pattern of negative cells in the first and third layers of D to match an M_6 move. The corresponding M_6 looks like

$$\begin{array}{ccccccccc} + & - & & & & & - & + & \\ - & & + & & & & + & & - \\ & + & - & & & & & - & + \end{array}$$

Therefore, if $d_{311} < 0$, we can make that M_6 move and the size of D gets smaller as signs match in at least 11 out of 12 cells. We would then return to step (1).

(12) When $d_{311} \geq 0$, D looks like

$+$	$-$	≥ 0	$-$	≤ 0		≥ 0	$+$	
$-$	≤ 0	$+$	≤ 0	≥ 0	≥ 0	$+$		$-$
≥ 0	$+$	$-$		≥ 0			$-$	

The M_6 move described before can be decomposed to the two following basic moves.

0	0	0				0	0	0
0	$-$	$+$				0	$+$	$-$
0	$+$	$-$				0	$-$	$+$

and

$+$	$-$	0				$-$	$+$	0
$-$	$+$	0				$+$	$-$	0
0	0	0				0	0	0

If we make the second of these two moves first, it can leave negative cells in t_{311} and t_{322} whereas if we make the other first it can leave the original negative cell in t_{111} and a further negative cell in t_{122}. Either way when we follow up with the other basic move, only one negative cell can remain in T in cell t_{311}. The size of D will have decreased as the M_6 move matches D in at least 10 out of 12 cells. We then return to step (1).

We have shown that there exists a path from T to T^* in the re-defined S' using only basic moves. Moreover, the path is finite as the size of D is a non-negative integer which decreases with each move.

□

Theorem 3.6 *The Bunea and Besag algorithm applied to* $3 \times 4 \times 4$ *tables creates an irreducible Markov chain on* $\mathcal{S}'$ *where* $\mathcal{S}'$ *is defined as in Theorem 3.5.*

Proof The proof of Theorem 3.5, as far as the end of step (10), applies to $3 \times J \times K$ tables for $J \geq 3$ and $K \geq 3$. We start this proof with D which looks like:

$$\begin{array}{ccccccccccc} + & & - & & \geq 0 & & - & & \leq 0 & & + \\ - & & \leq 0 & & + & & \leq 0 & & \geq 0 & \geq 0 & & + & & - \\ \geq 0 & & + & & & & & & \geq 0 & & & - & & \end{array}$$

(1) If either $d_{133} < 0$ or $d_{311} < 0$, we can make the M_6 move described at the end of the proof of Theorem 3.5, leaving at most a single -1 entry in T and decreasing the size of D. We would then return to step (1) of Theorem 3.5.

(2) Taking $d_{133} \geq 0$ and $d_{311} \geq 0$, D looks like:

$$\begin{array}{ccccccccccc} + & & - & & \geq 0 & & - & & \leq 0 & & \geq 0 & & + & \\ - & & \leq 0 & & + & & \leq 0 & & \geq 0 & \geq 0 & & + & & - \\ \geq 0 & & + & & \geq 0 & & & & \geq 0 & & & & - & \end{array}$$

Since $J = 4$, the constraint implies that $d_{143} < 0$. Without yet restricting to $K = 4$, the constraint implies that we can relabel columns 4 to K so that $d_{134} < 0$. Now D looks like:

$$\begin{array}{cccccccccc} + & - & \geq 0 & & - & \leq 0 & & \geq 0 & + & \\ - & \leq 0 & + & & \leq 0 & \geq 0 & \geq 0 & + & & - \\ \geq 0 & + & \geq 0 & - & & \geq 0 & & & - & \\ & & - & & & & & & & \end{array}$$

If $d_{341} < 0$ then we can make the basic move which has -1 in cells m_{121},m_{143},m_{341} and m_{323}; the size of D gets smaller and we create no additional negative entries in T. We would return to step (1) of Theorem 3.5.

A similar argument applies if $d_{314} < 0$.

(3) When $d_{341} \geq 0$ and $d_{314} \geq 0$ then D looks like:

$$\begin{array}{cccccccccc} + & - & \geq 0 & & - & \leq 0 & & \geq 0 & + & & \geq 0 \\ - & \leq 0 & + & & \leq 0 & \geq 0 & \geq 0 & + & & - & \\ \geq 0 & + & \geq 0 & - & & \geq 0 & & & - & & \\ & & - & & & & & & \geq 0 & & \end{array}$$

Here, the constraint and the fact that $J = 4$ imply that $d_{331} < 0$ and so D looks like:

$$\begin{array}{cccc} + & - & \geq 0 \\ - & \leq 0 & + \\ \geq 0 & + & \geq 0 & - \\ & & - \end{array} \qquad \begin{array}{ccc} - & \leq 0 \\ \leq 0 & \geq 0 & \geq 0 \\ & \geq 0 \end{array} \qquad \begin{array}{ccc} \geq 0 & + & \geq 0 \\ + & & - \\ - & - \\ \geq 0 \end{array}$$

Now, if $d_{324} < 0$ then we can make the basic move which has -1 in m_{121}, m_{134}, m_{324} and m_{331}; the size of D gets smaller and we create no additional negative entries in T. We would return to step (1) of Theorem 3.5.

(4) When $d_{324} \geq 0$, D looks like:

$$\begin{array}{cccc} + & - & \geq 0 \\ - & \leq 0 & + \\ \geq 0 & + & \geq 0 & - \\ & & - \end{array} \qquad \begin{array}{ccc} - & \leq 0 \\ \leq 0 & \geq 0 & \geq 0 \\ & \geq 0 \end{array} \qquad \begin{array}{ccc} \geq 0 & + & \geq 0 \\ + & & - & \geq 0 \\ - & - \\ \geq 0 \end{array}$$

Now, if $d_{344} < 0$, we have, in layers 1 and 3, 7 of the negative entries required to match a move in M_8. Those negative entries are in d_{112}, d_{121}, d_{323}, d_{332}, d_{134}, d_{143} and d_{344} and the missing entry is d_{411}. The move in M_8 is made from three basic moves: (i) with -1 in m_{112}, m_{211}, m_{311} and m_{322}; (ii) with -1 in m_{323}, m_{332}, m_{122} and m_{133}; (iii) with -1 in d_{134}, d_{143}, d_{333} and d_{344}. Implementing the M_8 move decreases the size of D as the signs of at least 12 non-zero entries out of 16 in the move match D. Implementing the M_8 move using basic moves may create temporarily a second negative entry in T but at the end the only possible negative entry would be in t_{411}. We would return to step (1) in Theorem 3.5.

(5) When $d_{344} \geq 0$, D looks like

$$\begin{array}{cccc} + & - & \geq 0 \\ - & \leq 0 & + \\ \geq 0 & + & \geq 0 & - \\ & & - \end{array} \qquad \begin{array}{ccc} - & \leq 0 \\ \leq 0 & \geq 0 & \geq 0 \\ & \geq 0 \end{array} \qquad \begin{array}{ccc} \geq 0 & + & \geq 0 \\ + & & - & \geq 0 \\ - & - \\ \geq 0 & & \geq 0 \end{array}$$

The constraint and $J = 4$ now imply that $d_{334} \leq 0$ so that D looks like

$$\begin{array}{cccc} + & - & \geq 0 & \\ - & \leq 0 & + & \\ \geq 0 & + & \geq 0 & - \\ & & - & \end{array} \qquad \begin{array}{ccc} & - & \leq 0 \\ \leq 0 & \geq 0 & \geq 0 \\ & \geq 0 & \end{array} \qquad \begin{array}{cccc} \geq 0 & + & & \geq 0 \\ + & & - & \geq 0 \\ - & - & & \leq 0 \\ \geq 0 & & & \geq 0 \end{array}$$

At this point, we now use the fact that $K = 4$ together with the constraint. We deduce that $d_{313} < 0$ and $d_{333} > 0$ which then implies that $d_{233} < 0$. D now looks like

$$\begin{array}{cccc} + & - & \geq 0 & \\ - & \leq 0 & + & \\ \geq 0 & + & \geq 0 & - \\ & & - & \end{array} \qquad \begin{array}{ccc} & - & \leq 0 \\ \leq 0 & \geq 0 & \geq 0 \\ & \geq 0 & - \end{array} \qquad \begin{array}{ccccc} \geq 0 & + & - & \geq 0 \\ + & & - & \geq 0 \\ - & - & + & \leq 0 \\ \geq 0 & & & \geq 0 \end{array}$$

We see that we can make the basic move with -1 in m_{211}, m_{233}, m_{313} and m_{331} and that we create no additional negative entry in T. Enough signs match to ensure that the size of D decreases and we return to step (1) of Theorem 3.5.

For any configuration of the tables, we have shown that we can find one or more basic moves which reduce the size of D and leave at most a single negative entry in T and which temporarily create at most one additional negative entry in T.

□

3.4 Conclusion

As previously mentioned, Aoki and Takemura have found a unique minimal Markov basis for $3 \times 3 \times K$ tables which includes the different types of move from degree 4 to 10. They have also found the list of indispensable moves for $3 \times 4 \times K$ tables which includes different types of move from degree 4 to 16.

In this chapter, we have proved the novel result that constructing the connected Markov chain over $\mathcal{S}$ for $3 \times 3 \times K$ and $3 \times 4 \times 4$ tables is possible by using only basic moves, allowing tables to go to $\mathcal{S}'$, redefined to allow at most two cell entries equal to -1. This makes implementation easier, than for MCMC using the Aoki and Takemura Markov bases, as we can avoid producing more complicated moves of higher degree.

Chapter 4

Efficiency of MCMC exact methods

4.1 Introduction

Throughout this chapter we assess the efficiency of MCMC methods introduced in Chapters 2 and 3 for computing the exact p-value. We first use an example of a $2 \times 3 \times 3$ table to introduce the methodology proposed by Bunea and Besag (2000). We have extended their approach to an algorithm which accepts random moves from M_6 as well as random basic moves from M_4. It should be remembered that Bunea and Besag (2000) use only basic moves. We also implement the MCMC method based on the method introduced by Diaconis and Sturmfels (1998), where moves will be selected from M^*, the set of moves of different degrees. This comparison needs us to define a measure for the efficiency of the algorithms. Hence, we introduce two different types of measure of efficiency for the algorithms by which the efficiency of the algorithms will be investigated.

It also provides a comparison between the efficiency of each algorithm and the alternatives. For algorithms which depend on a parameter the efficiency metric is computed over the range of parameter values to determine whether there is an optimum value for the parameter.

The comparison study will be carried out on tables with small, moderate, and large sample sizes. That is, we choose tables with small, moderate, and large values for cell entries. We are also interested in evaluating the performance of different algorithms for tables with larger dimension. For this purpose, we run the algorithms over a table with the larger dimension of $2 \times 5 \times 6$.

Bunea and Besag (2000) proved their method only for contingency tables with two layers, that is $2 \times J \times K$. In chapter 3 we proved the algorithm which only uses the basic moves to provide an irreducible Markov chain over tables of $3 \times J \times K$ where $J = 3, 4$. Aoki and Takemura (2003) introduced the list of indispensable set of basic moves which construct the unique minimal Markov basis over contingency tables of dimension $3 \times J \times K$, for small $J = 3, 4$. We will evaluate the performance of algorithm introduced in chapter 3, as well as Aoki and Takemura's algorithm on contingency tables of dimension $3 \times 3 \times 5$ and $3 \times 4 \times 5$.

4.2 Computing efficiency

This section explains the methods we used to compare the efficiency of the algorithm. The efficiency measure will be computed based on two components: effective sample size; computational cost.

Effective sample size

A common problem of using MCMC methods is that the values $\theta^{(t)}$ generated at iteration t look much like the $\theta^{(t+1)}$ and this similarity continues for all iterations, or in technical terms there is auto-correlation between the samples. Therefore, the contribution of each additional sample to the quality of inferences about the posterior density can be small. Hence, the sampler requires a longer run of the MCMC algorithm to reach a sample size equivalent to n independent samples.

We can compute the effective sample size from an MCMC sample by considering the auto-correlation of the generated MCMC samples. The formula for the effective sample size is given by

$$N_{eff} = \frac{n}{1 + \sum_k \rho_k}\,, \tag{4.1}$$

where ρ_k is the auto-correlation of lag k (Gamerman and Lopes, 2006) and n is MCMC sample size. We also need to decide which variable to use to compute the auto-correlation in (4.1). One option is to use the sequence of indicator values, I_t, which equals 1 if the generated table has a probability less than the probability of the observed table and zero otherwise. Note that it might seem natural to use this indicator, as the p-value can be obtained by averaging this indicator. We tried this indicator for several examples along with other measures and we observed that this does not reveal the autocorrelation

between sequence of generated tables, i.e. a disadvantage of using this indicator is that it removes much information about the generated table. For this reason we have not used this metric.

An alternative is to use the probability of the generated table. Instead of the original probability we prefer to use the logarithm of probability. This way we spread out small values over a continuous range which suitably reveals the variability. Let us denote the probability of observing the table generated at time t by π_t; $\ln(\pi_t)$ is used for the efficiency analysis.

For a fixed MCMC sample length, we prefer an algorithm that provides a larger effective sample size for the same cost - the higher the effective sample size, the higher the efficiency. We denote the effective sample size for the logarithm of the probability of the table by n_{eff}. Note that effective sample size cannot represent the efficiency of the algorithm alone, as time spent to run an algorithm is a factor to be considered. In simple terms, consider two algorithms with two different running times for a fixed number of iterations. It is obvious that for a fixed time period the faster algorithm can produce more MCMC samples. For this reason, the time of running an algorithm needs to be considered as a cost factor in measuring the efficiency of the algorithm. We define the effective sample size divided by the CPU running time of the algorithm as an efficiency measure

$$\text{eff}_T = \frac{N_{eff}}{\text{Running Time}} \tag{4.2}$$

This measure will be computed for several examples later in this chapter.

Cost of an algorithm using number of memory bits

In assessing the efficiency of an algorithm, the cost can be measured by the total number of bits used in an algorithm. Devroye (1986) shows how this measure can be computed and used as the cost factor in comparing two algorithms. Devroye (1986) shows how many bits are needed to generate a discrete random variable. The exact number of bits required varies depending on what happens in the process, each time we generate a value. So instead of CPU running time one can use the expected number of bits used in the algorithm. Devroye (1986) also shows that optimal algorithm and the expected number of bits depend on both the number of possible values taken by the random variable and on the probabilities, in a complex way. A simple, close to optimal algorithm for choosing

from any collection of n equally likely objects gives

$$C = E[\#\text{ Bits}] = 2^m \frac{m}{n} \tag{4.3}$$

where $m = \lceil \log_2 n \rceil$, and n is the number of possible values. We use this calculation as the basis of counting the cost of choosing random moves. The $\lceil\ \rceil$ denotes the ceiling function.

In our case, n is the number of possible moves. So this value varies depending on the type of move and dimension of the table. For example, assume a table of dimension $2 \times J \times K$. The number of possible basic moves of degree 4 is given by:

$$\begin{aligned} n &= \frac{1}{2!} J(J-1)K(K-1) && (4.4)\\ &= 2\binom{J}{2}\binom{K}{2} && (4.5) \end{aligned}$$

as we need to select two rows and two columns at random which can happen with $J(J-1)K(K-1)$ possibilities. However, if we swap both the rows and columns we get the same move. Similarly, we can calculate the number of moves of degree 6. This time, it is necessary to select three rows and three columns, but there exist 3! permutations of rows and columns which give the same moves:

$$\begin{aligned} n &= \frac{1}{3!} J(J-1)(J-2)K(K-1)(K-2) && (4.6)\\ &= 6\binom{J}{3}\binom{K}{3}. && (4.7) \end{aligned}$$

By induction we can show that for a table of dimension $2 \times J \times K$ a move of degree $2m$ the number of possible moves is:

$$\begin{aligned} n &= \frac{1}{m!} J(J-1)\cdots(J-m+1)K(K-1)\cdots(K-m+1) && (4.8)\\ &= m!\binom{J}{m}\binom{K}{m}. && (4.9) \end{aligned}$$

We also count the number of times a uniform random number is used to accept or reject a table. We refer to this cost as Bernoulli Cost and denote it by C_b in this chapter.

The cost of an algorithm in random bit model for generating a Bernoulli sample is between 1 and 2. If the parameter of a Bernoulli distribution, $p = 0.5$ it only needs one random bit. For $p \neq 0.5$ the expected number of bits required is not greater than 2. For illustration consider a binary expansion of the Bernoulli parameter $p = 0.p_1p_2p_3\cdots$. For generating the Bernoulli value it needs to generate binary random bits B_i until the first $B_i \neq p_i$. This

way, the probability of exiting after i bits is $1/2^i$. So the expected value for the number of bits is calculated by

$$E(\#\text{bits}) = \sum_{i=1}^{\infty} i \left(\frac{1}{2}\right)^i = 2. \tag{4.10}$$

For this reason we use $C + 2 \times C_b$ as the upper bound for the total cost of the algorithm.

So the lower and upper bound for the cost is given by

$$(C_L\ ,\ C_U) = (\ C + 1 \times C_b\ \ ,\ \ C + 2 \times C_b\ ).$$

Now, considering the effective sample size and the cost of an algorithm (the total number of bits required) we can define a lower and upper bound for the efficiency measure, denoted by eff_L and eff_U respectively. A simple idea is to divide the effective sample size for a fixed number of MCMC samples by the total number of bits used to run the algorithm.

$$\text{eff}_L = \frac{N_{eff}}{C_U} \tag{4.11}$$

$$\text{eff}_U = \frac{N_{eff}}{C_L} \tag{4.12}$$

4.3 A case study for 2×J×K tables

In this section we aim to compute and compare the efficiency of MCMC methods in Chapter 2 over a table of dimensions $2 \times 3 \times 3$. We choose different tables of small, moderate, and large values for cell entries to compute the p-value and the efficiency throughout this chapter. We also study how the efficiency of algorithms is affected by the dimensions of the tables. We first introduce three algorithms: M2BB, M23BB and M23.

Table 4.1: Notation used to represent different algorithms

Notation	Description
M2BB	Using only moves of degree 4, M_4, allowing tables to leave $\mathcal{S}$.
M23BB	Using moves of degree 4 and 6, allowing tables to leave $\mathcal{S}$.
M23	Using moves of degree 4 and 6, staying in $\mathcal{S}$ all the time.
MMs	Using moves from M^*, staying in $\mathcal{S}$ all the time.
MMsBB	Using moves from M^*, allowing tables to leave $\mathcal{S}$.

Algorithm M2BB uses the Bunea and Besag idea to generate tables using basic moves of degree 4. We extend this algorithm to enable the generation of moves of degree 6 as well

as basic moves of degree 4 and we call the resulting algorithm M23BB. M23 is the same algorithm without possibility to go in S' and is equivalent to the method introduced in Diaconis and Sturmfels (1998). This is followed by an explanation of these algorithms in more detail and then we apply them to a table of dimension $2 \times 3 \times 3$. As this table has small numbers in the cells, with maximum values 5 and total value 50, it will be denoted by T_S. We use this table later in this section along with tables with larger values to investigate the performance of these algorithms.

T_S :

2	2	4
4	1	1
3	1	4

2	4	2
5	3	4
3	2	3

We found the complete reference set for the table T_S to compute the exact p-value for the no three-way interaction model for table T_S. This enables us to investigate how the computed p-value from algorithms M2BB, M23BB and M23 compare to the true p-value. For this purpose we solved the the system of equations produced for table T_s conditional on the marginal totals. An `R` code, shown in Appendix A.2 was used to find the set of all solution of the system of equations for which the corresponding table does not contain any negative cell entries. The 216 tables in the reference set are listed in Appendix A.2, where each row represents a table. The exact p-value is found to be 0.9190594.

In what follows, we will present results of various algorithms applied to a number of different contingency tables, including T_S. The results will be presented mainly as tables. A short description of the notation used in the tables is given in Table 4.2

M2BB-Algorithm

In this algorithm we use only the basic moves to generate tables of the same dimension and with the same margins. This algorithm allows the tables to have the proposals from $\mathcal{S}'$ (Bunea and Besag, 2000), that is, the proposal can accept tables with cells of -1. Because we only use the basic moves from M_4 the number of possible moves is:

$$n = 2\binom{3}{2}\binom{3}{2} = 18$$

and equivalently the average number of bits required each time a new move is generated is 8.889. We also consider the cost of the number of bits for generating a Bernoulli sample which is used in acceptance sampling of MCMC.

Table 4.2: Notation used to represent results of MCMC samples

Notation	Description
(p_i, p_j)	The probability distribution for choosing basic moves, where p_i shows the probability of choosing a basic move of degree i.
p-value	The average of the p-values computed for each MCMC chain.
N_{eff}	The effective sample size, computed based on logarithm of the probability of the observed table.
Eff Lower	The average of the lower bands for the efficiency of the algorithm
Eff Upper	The average of the upper band for the efficiency of the algorithm
Time (Sec)	The average of the CPU running time spent for the algorithm
Eff Time	The average of the efficiency measure computed based on running time
Cost Lower	The average of the lower cost computed for the each algorithms
Cost Upper	The average of the upper cost computed for an algorithm
N-bits Total	The average of total number of bits used in an algorithm
N-bits Bernoulli	The total number of Bernoulli bits counted for during algorithm (Average)
N-bits Sp	The average number of bits used while the algorithm is in $\mathcal{S}'$
N-bits So	The average number of bits used, while in $\mathcal{S}'$ but outside $\mathcal{S}$, making proposals which would take the chain outside $\mathcal{S}'$

We produced the `R` function `mcmc.m2bb` to produce the MCMC sample based on Bunea and Besag (2000) which is shown in the Appendix A.1. The algorithm has been applied over a table of dimension $2\times3\times3$ as an example. The estimated p-value for no interaction effect is $0.919 \pm 3 \times 0.0006$. The effective sample size is 1008. The efficiency computed for the algorithm is $Eff_L = 0.00516$ with the upper limit of $Eff_U = 0.00537$. The result will be presented in Table 4.3 and Table 4.4 along with results of the other algorithms.

The p-value in Table 4.3 is the average of 100 estimated p-values obtained by 100 times running an MCMC of sample size 10,000. The value underneath the metric in each cell shows the standard deviation of the metric. So for a new MCMC run using M2BB algorithm we have approximately 95% probability for the p-value being in the interval of $0.919 \pm 1.96 \times 0.006$. The 95% confidence interval for the true p-value is given by $0.919 \pm 1.96 \times 0.006/\sqrt{100}$. We also show the average of the effective sample size over the 100 MCMC runs, as well as the corresponding standard deviation (see the formula for effective sample size in (4.1)).

The upper and lower limit for efficiency of each algorithm is calculated based on (4.11) for each MCMC run. The averages and standard deviations of these limits are reported

in Table 4.3. The average running time and corresponding efficiency are also reported. In Table (4.4), we report the cost related metrics.

Table 4.3: The efficiency of the algorithms for a table of dimension $2 \times 3 \times 3$, with small values for cell entries, T_S. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6)	Estimated p-value	N_{eff}	Eff Lower	Eff Upper	Time (Sec)	Eff Time
M2BB	(1.00,0.00)	0.919	1,008	0.00516	0.00537	2.68	375.79
		0.006	75	0.00040	0.00042	0.06	29.33
M23	(0.01,0.99)	0.924	847	0.01154	0.01487	1.63	520.06
		0.025	67	0.00091	0.00118	0.04	41.00
M23	(0.10,0.90)	0.919	856	0.01101	0.01396	1.59	539.01
		0.011	81	0.00104	0.00131	0.04	55.41
M23	(0.50,0.50)	0.920	873	0.00896	0.01079	1.54	568.81
		0.007	61	0.00063	0.00076	0.02	40.48
M23	(0.90,0.10)	0.919	874	0.00746	0.00869	1.55	563.53
		0.006	70	0.00059	0.00069	0.04	46.41
M23	(0.99,0.01)	0.919	887	0.00730	0.00845	1.53	581.08
		0.007	68	0.00056	0.00065	0.02	46.18
M23BB	(0.01,0.99)	0.920	1,026	0.00720	0.00904	3.36	306.00
		0.020	63	0.00049	0.00062	0.09	20.11
M23BB	(0.10,0.90)	0.920	1,047	0.00689	0.00852	3.31	316.63
		0.009	75	0.00051	0.00064	0.09	25.22
M23BB	(0.50,0.50)	0.921	1,050	0.00553	0.00651	3.13	335.11
		0.008	65	0.00038	0.00045	0.08	23.15
M23BB	(0.90,0.10)	0.919	1,033	0.00453	0.00518	3.04	339.86
		0.006	72	0.00037	0.00042	0.08	27.21
M23BB	(0.99,0.01)	0.918	1,019	0.00433	0.00491	3.00	339.45
		0.006	75	0.00034	0.00039	0.08	27.79

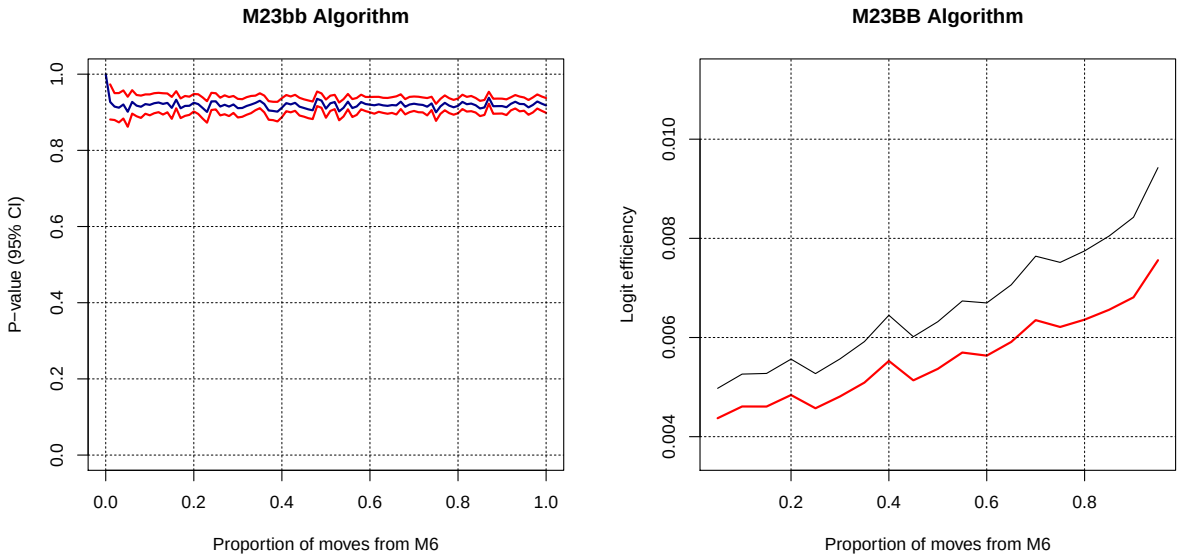
† The second value in each cell is the standard deviation of the metric

Table 4.4: The cost of the algorithms for a table of dimension $2 \times 3 \times 3$, with small values for cell entries, T_S. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6)	Cost Lower	Cost Upper	N-Bits Total	N-Bern	N-Bits Sp	N-Bits So
M2BB	(1.00,0.00)	187,703	195,425	179,980	7,722	64,304	26,787
		4,861	4,850	4,872	34	3,259	1,690
M23	(0.01,0.99)	56,937	73,385	40,488	16,448		
		93	167	44	79		
M23	(0.10,0.90)	61,344	77,808	44,880	16,464		
		153	185	152	72		
M23	(0.50,0.50)	80,984	97,492	64,476	16,508		
		235	267	221	72		
M23	(0.90,0.10)	100,591	117,157	84,025	16,566		
		156	184	144	56		
M23	(0.99,0.01)	104,984	121,571	88,397	16,587		
		85	144	50	66		
M23BB	(0.01,0.99)	113,596	142,573	84,618	28,978	25,947	18,173
		2,403	2,872	1,935	470	1,032	959
M23BB	(0.10,0.90)	122,944	151,952	93,936	29,008	29,351	19,703
		2,545	2,992	2,098	449	1,198	983
M23BB	(0.50,0.50)	161,367	189,947	132,788	28,579	44,141	24,202
		3,975	4,507	3,443	535	2,036	1,518
M23BB	(0.90,0.10)	199,920	228,137	171,703	28,217	60,836	26,859
		6,241	6,889	5,593	648	3,630	2,029
M23BB	(0.99,0.01)	207,501	235,528	179,474	28,027	64,126	26,942
		5,058	5,571	4,545	514	3,032	1,627

† The second value in each cell is the standard deviation of the metric

Figure 4.1: *The p-value estimated based on MCMC sample produced using M23BB algorithm (left), and the efficiency of the algorithm (right), for contingency table* T_S



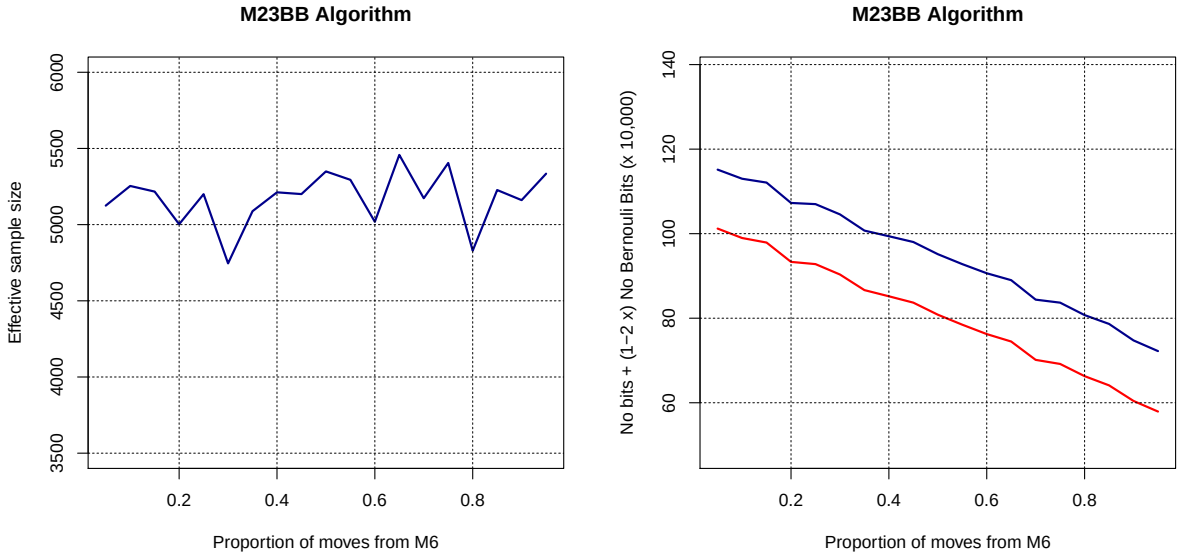
M23BB Algorithm

We extend the Bunea and Besag (2000) algorithm to use M_6 as well as M_4. The proportion of moves selected from M_4, p can assume values in $[0, 1]$, where $p = 0$ corresponds to M2BB algorithm, as it uses only basic moves. In the extreme case when $p = 1$ the algorithm chooses the moves only from M_6.

Figure 4.1 (left) shows the estimated p-value for the given table. This p-value has been computed for different proportions, p, of moves chosen from M_6. The figure shows that change of proportion does not affect the estimated p-value significantly. Figure 4.1 (right) shows the lower and upper values of efficiency of the algorithm for several values of p. It can be seen that choosing more moves from $M6$ gradually increases the efficiency of the algorithm.

Figure 4.2 (left) shows the effective sample size of the generated sample tables as we move from an algorithm which chooses more basic moves M_4 rather than moves of degree 6, M_6. The graph does not reveal any clear pattern for the effective sample size when higher proportion of moves is selected from M6. Figure 4.2 (right) shows the lower and upper band for the cost function and shows that cost decreases as we select more moves from M_6.

Figure 4.2: *The effective sample size of M23BB algorithm for several proportions of moves selected from the moves of degree 6, M_6 (left); and the cost of algorithm (right), for contingency table T_S*



M23 Algorithm

For a table of size $2 \times 3 \times 3$, an irreducible Markov chain can be achieved, without leaving $\mathcal{S}$, by choosing moves from M_4 and M_6. For this reason, we can generate the MCMC sample by randomly choosing a move from M_4 and M_6 as they provide the complete reference set. The proportion of moves selected from M_4, p can assume values in $(0, 1)$. Note that in this algorithm the proposed tables with negative cells are always rejected.

Figure 4.3 (left) shows the p-value computed for different scenarios in terms of the proportion of moves selected from M_6. This shows a negligible changes for different values of p. These are also very close to values computed by other algorithms. Figure 4.3 (right) computes the efficiency of the algorithm for different proportions of selected moves, p. It shows that by choosing more moves of higher degree the efficiency of algorithm increases.

Figure 4.4 (right) depicts the effective sample size for different proportion of moves allocated to moves of degree 6. similarly to the M23BB algorithm, changing the proportion of moves of degree 6 does not affect the effective sample size. Figure 4.4 (right) shows the number of bits used in the M23 algorithm. We can clearly see that increase of the moves selected from M_6 decreases the cost of the algorithm considerably.

Figure 4.3: *The p-value estimated based on MCMC sample produced using M23 algorithm (left), and the efficiency of the algorithm (right)*

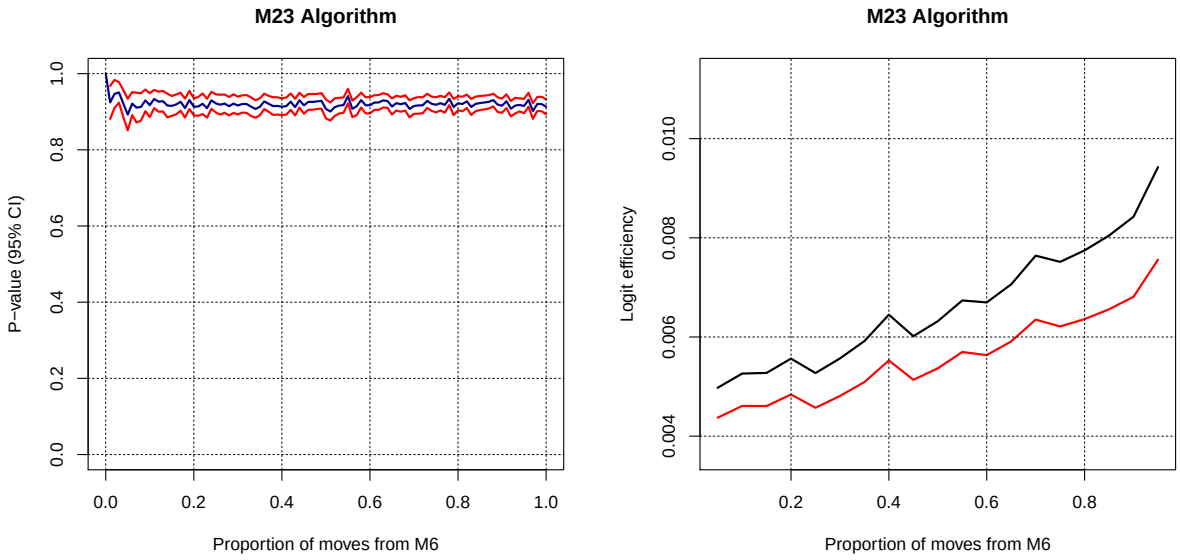
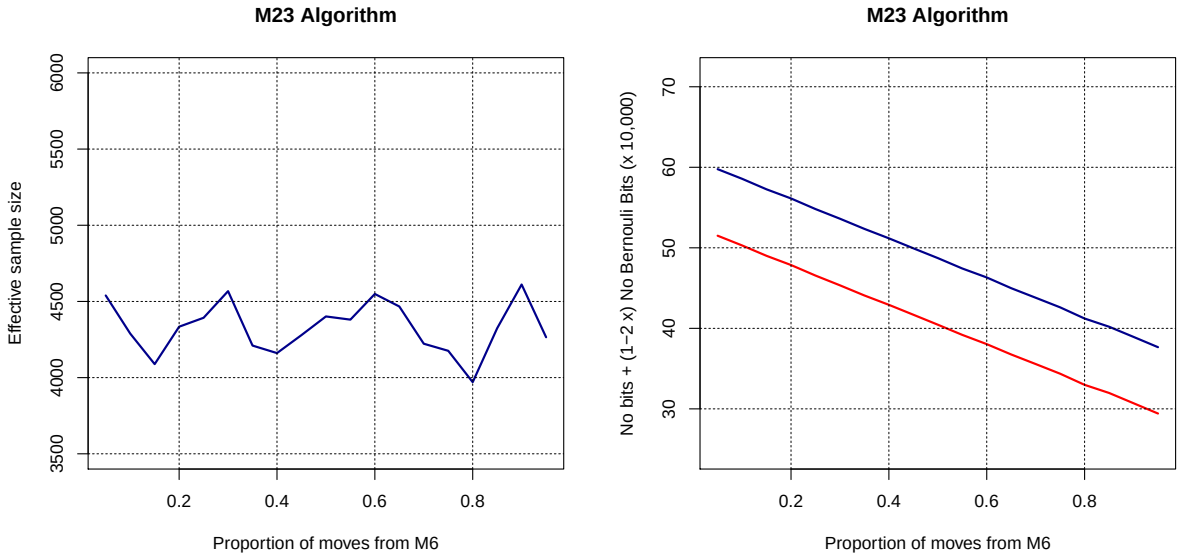


Figure 4.4: *The effective sample size of M23 algorithm for several proportions of moves selected from the moves of degree 6, M_6 (left); and the cost of algorithm (right), for contingency table T_S*



We defer comparison of the algorithms and detailed examination of their relative efficiency to the next section. This includes applying the algorithms on tables with different sizes, as well as tables of different dimensions.

4.3.1 Small/Medium/Large tables

In this section we aim to compare the performance of the three algorithms for tables of different sizes. So we consider three tables of the same dimensions, $2 \times 3 \times 3$, which vary in terms of the minimum cell entry and the total of the table frequencies.

Small size table

As a small size table we continue to use T_S. Table 4.3 summarises the results from all three algorithms for our small size table of dimension $2 \times 3 \times 3$. For M23 and M23BB algorithm we show the efficiency of the algorithm for five different scenarios, based on the probability of choosing a move of degree 6 (p=0.01,0.1, 0.5, 0.9, 0.99). The M2BB algorithm shows the higher effective sample size in comparison with the M23 algorithm. The highest effective sample size is produced by our extended algorithm M23BB.

Table 4.4 compare the cost information of the three algorithms. The cost of algorithm M23 in terms of number of bits used is lower than the other algorithms. This is due to the fact that M2BB and M23BB use some bits while they are in $\mathcal{S}'$. It can be seen that in our small size table, T_S, M2BB and M23BB use about 50% of the total number of bits making and accepting or rejecting proposals in $\mathcal{S}$; about 30% are used making moves in $\mathcal{S}'$ and about 20% in making proposals outside $\mathcal{S}'$.

The number of bits used for the Bernoulli random generation is smaller for the M2BB algorithms. This is because in the M23 and the M23BB algorithms a Bernoulli random bit is required to choose between a random move of M_4 or M_6. We add the Bernoulli random bit to the random bit required for moves to compute the cost of the algorithm. In total, the cost of M2BB and M23BB is more than the cost of the M23 algorithm.

Having focused on the efficiency measure based on bits, M23 is more efficient than the other methods. Comparing the M2BB and M23BB the efficiency depends on the proportion of times that moves are selected from M_4 and M_6. So we can find an interval on this proportion for which M2BB is more efficient than M23BB. In this example the efficiency of the latter algorithm increases as it becomes more likely to select moves from M6.

Table 4.5: The efficiency of the algorithms for a table of dimension $2\times3\times3$, with moderate values cell entries, T_M. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6)	Estimated p-value	N_{eff}	Eff Lower	Eff Upper	Time (Sec)	Eff Time
M2BB	(1.00,0.00)	0.00026	328	0.00323	0.00345	1.61	203.45
		0.00042	37	0.00036	0.00039	0.02	22.56
M23	(0.01,0.99)	0.00031	427	0.00580	0.00749	1.61	266.36
		0.00028	42	0.00058	0.00074	0.05	27.63
M23	(0.10,0.90)	0.00036	417	0.00535	0.00679	1.61	260.02
		0.00038	46	0.00059	0.00075	0.06	31.24
M23	(0.50,0.50)	0.00041	378	0.00388	0.00467	1.58	239.51
		0.00045	40	0.00041	0.00049	0.06	26.89
M23	(0.90,0.10)	0.00043	336	0.00288	0.00335	1.55	217.18
		0.00050	38	0.00032	0.00037	0.05	26.61
M23	(0.99,0.01)	0.00040	333	0.00275	0.00318	1.52	219.06
		0.00042	35	0.00029	0.00033	0.06	25.15
M23BB	(0.01,0.99)	0.00019	439	0.00597	0.00770	1.84	239.36
		0.00038	46	0.00062	0.00080	0.05	26.84
M23BB	(0.10,0.90)	0.00022	421	0.00540	0.00686	1.81	232.65
		0.00036	39	0.00050	0.00064	0.05	22.52
M23BB	(0.50,0.50)	0.00025	387	0.00398	0.00479	1.81	214.17
		0.00037	40	0.00041	0.00049	0.05	23.56
M23BB	(0.90,0.10)	0.00034	336	0.00288	0.00335	1.79	188.27
		0.00047	31	0.00027	0.00031	0.07	19.76
M23BB	(0.99,0.01)	0.00031	326	0.00270	0.00312	1.77	184.52
		0.00056	37	0.00031	0.00035	0.06	22.96

† The second value in each cell is the standard deviation of the metric

Table 4.6: The cost of the algorithms for a table of dimension $2 \times 3 \times 3$, with moderate values cell entries, T_M. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6)	Cost Lower	Cost Upper	N-Bits Total	N-Bern	N-Bits Sp	N-Bits So
M2BB	(1.00,0.00)	95,191	101,492	88,889	6,302	0	0
		31	63	1	31	1	0
M23	(0.01,0.99)	57,069	73,655	40,483	16,586		
		55	73	46	28		
M23	(0.10,0.90)	61,462	78,016	44,908	16,554		
		150	159	147	32		
M23	(0.50,0.50)	80,905	97,345	64,464	16,441		
		277	284	275	35		
M23	(0.90,0.10)	100,342	116,672	84,011	16,330		
		139	155	131	36		
M23	(0.99,0.01)	104,702	121,001	88,402	16,300		
		57	78	49	31		
M23BB	(0.01,0.99)	57,065	73,643	40,487	16,578	0	0
		64	87	53	33	0	0
M23BB	(0.10,0.90)	61,444	77,996	44,892	16,552	0	0
		152	164	148	35	0	0
M23BB	(0.50,0.50)	80,852	97,287	64,417	16,435	1	0
		247	254	244	32	5	2
M23BB	(0.90,0.10)	100,341	116,666	84,017	16,324	0	0
		147	154	148	33	2	0
M23BB	(0.99,0.01)	104,703	121,007	88,399	16,304	0	0
		56	73	54	32	0	0

† The second value in each cell is the standard deviation of the metric

Medium size table

Now we extend our study to see how the algorithms perform over a medium size table. We use a table of the same dimension as in the previous section. This time the minimum value for the cells is 10 and the total of the table cell entries is 385.

T_M :

20	15	10
10	15	30
10	25	20

25	30	10
15	30	40
50	15	25

Contrary to the small size table, the effective sample size for the M23 is larger than the M2BB equivalents. On the other hand, the effective sample sizes for the M23 algorithm are very close to the M23BB. The similarity between the M23 and the M23BB can be explained by the small number of times that M23BB visits $\mathcal{S}'$, as indicated by the small number of bits used in making moves in $\mathcal{S}'$ or making proposals outside $\mathcal{S}'$.

The M2BB imposes a lower Bernoulli cost in comparison with the other methods which gives it advantages when the large proportion of moves are selected from M_4. That is, the M2BB has higher efficiency than the M23 and M23BB for some range of ps. Hence, it would be beneficial to find a threshold $p_\circ$ so that for $p < p_\circ$ the M23 and M23BB outperform the M2BB. Table 4.7 shows that the system time slightly increases as the algorithms select more moves from M6.

Table 4.5 and Table 4.6 show the results based on $1,000,000$ MCMC samples.

Large size table

We repeat the analysis for a table with larger entries and total. The table T_L is selected with minimum value 30. The total over the table cells is 1940.

T_L :

100	50	40
55	40	110
70	125	130

200	50	120
400	50	30
120	150	100

Table 4.7 and Table 4.8 summarize the outputs of the three algorithms. It can be seen that the M2BB and M23BB are not likely to move into $\mathcal{S}'$ or to make proposals outside $\mathcal{S}'$. For this reason, the M23 and M23BB perform quite similarly. Effective sample sizes show the same pattern as for medium size table, i.e. the more moves from M6, the higher

Table 4.7: The efficiency of the algorithms for a table of dimension $2 \times 3 \times 3$, with large values for cell entries, T_L. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6)	Estimated p-value	N_{eff}	Eff Lower	Eff Upper	Time (Sec)	Eff Time
M2BB	(1.00,0.00)	0.00016	58	0.00058	0.00062	1.69	34.58
		0.00022	12	0.00012	0.00012	0.05	6.81
M23	(0.01,0.99)	0.00024	78	0.00109	0.00139	1.61	48.48
		0.00021	17	0.00024	0.00030	0.06	10.44
M23	(0.10,0.90)	0.00023	72	0.00094	0.00119	1.66	43.54
		0.00018	15	0.00020	0.00025	0.05	9.18
M23	(0.50,0.50)	0.00024	68	0.00071	0.00085	1.58	43.24
		0.00026	15	0.00016	0.00019	0.05	9.74
M23	(0.90,0.10)	0.00026	59	0.00051	0.00059	1.59	36.93
		0.00024	13	0.00011	0.00013	0.04	7.97
M23	(0.99,0.01)	0.00027	57	0.00048	0.00055	1.50	38.48
		0.00035	10	0.00009	0.00010	0.04	7.11
M23BB	(0.01,0.99)	0.00014	79	0.00110	0.00140	1.84	42.90
		0.00020	20	0.00027	0.00035	0.05	10.64
M23BB	(0.10,0.90)	0.00016	73	0.00096	0.00121	1.83	40.02
		0.00022	15	0.00019	0.00024	0.05	7.90
M23BB	(0.50,0.50)	0.00015	69	0.00072	0.00086	1.80	38.08
		0.00021	13	0.00014	0.00017	0.05	7.37
M23BB	(0.90,0.10)	0.00017	58	0.00051	0.00059	1.77	33.09
		0.00026	14	0.00012	0.00014	0.05	8.11
M23BB	(0.99,0.01)	0.00016	55	0.00046	0.00053	1.73	31.71
		0.00024	10	0.00008	0.00009	0.05	5.76

† The second value in each cell is the standard deviation of the metric

Table 4.8: The cost of the algorithms for a table of dimension $2 \times 3 \times 3$, with large values for cell entries, T_L. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6)	Cost Lower	Cost Upper	N-Bits Total	N-Bern	N-Bits Sp	N-Bits So
M2BB	(1.00,0.00)	94,503	100,118	88,889	5,614	0	0
		32	65	0	32	0	0
M23	(0.01,0.99)	56,251	72,012	40,489	15,761		
		62	81	55	31		
M23	(0.10,0.90)	60,651	76,401	44,900	15,750		
		138	148	134	31		
M23	(0.50,0.50)	80,144	95,832	64,456	15,688		
		217	224	216	34		
M23	(0.90,0.10)	99,628	115,261	83,994	15,634		
		143	151	140	28		
M23	(0.99,0.01)	104,020	119,637	88,403	15,617		
		60	84	50	34		
M23BB	(0.01,0.99)	56,252	72,018	40,486	15,766	0	0
		50	67	45	28	0	0
M23BB	(0.10,0.90)	60,628	76,375	44,880	15,747	0	0
		131	140	128	31	0	0
M23BB	(0.50,0.50)	80,117	95,802	64,432	15,685	0	0
		205	213	201	33	0	0
M23BB	(0.90,0.10)	99,645	115,279	84,011	15,634	0	0
		157	166	153	29	0	0
M23BB	(0.99,0.01)	104,020	119,643	88,397	15,623	0	0
		60	78	57	32	0	0

† The second value in each cell is the standard deviation of the metric

the effective sample size, but the cost of the algorithm decreases by choosing more moves from M6.

In general, there is no considerable difference between tables with moderate and large cell entries in terms of pattern of efficiency, cost and effective sample size of the algorithms. Finally, the M23 and M23BB algorithms have higher efficiency for some range of proportion of M6 but they do not fully dominate the M2BB in terms of efficiency.

Overall, comparing the three algorithms for these different tables, we conclude that while M2BB is sometimes less efficient than the other algorithms, the different in efficiency is not great and is offset by the fact that it is much simpler to write a single program for M2BB which works for all tables than for the other algorithms.

4.3.2 Non-irreducible tables

In this section, we check the performance of the algorithms for a specific class of $2 \times 3 \times 3$ tables for which using only basic moves of degree 4 does not provide an irreducible Markov Chain. Consider the following table which includes structural zero cells:

T_{nir} :

3	0	3
0	2	0
0	0	2

1	0	3
6	2	0
0	3	3

For this table, the elements of the reference set will not be connected without accepting a proposal in $\mathcal{S}'$. The zeros in the table are arranged so that any basic move must have a non-zero entry corresponding to one of the zeroes in the second layer. Since every zero in the second layer matches a zero in the first layer, every basic move would create at least one negative entry. On the other hand, there is more than one table in the reference set. We run the algorithms over T_{nir} to generate $100,000$ MCMC sample. Table 4.9 and Table 4.10 summarize the results.

We see that running M23 with $p_6 = 0$, the case when we use only basic moves and don't leave $\mathcal{S}$, provides a p-value equal to 1, because we can not make a step with M_4 and we only have the observed table, while M23BB and M23 provide p-value equal to 0.15.

Table 4.9: The efficiency of the algorithms for a table of dimension $2 \times 3 \times 3$, with small values for cell entries for which basic moves does not provide an irreducible Markov chain, T_{nir}. The number of MCMC sample is $N = 100 \times 1,000$

Algorithm	(p_4, p_6)	Estimated p-value	N_{eff}	Eff Lower	Eff Upper	Time (Sec)	Eff Time
M2BB	(1.00,0.00)	0.15619	344	0.00109	0.00109	4.32	79.11
		0.02931	89	0.00029	0.00030	0.25	21.18
M23	(0.10,0.90)	0.16312	12	0.00131	0.00167	0.17	54.57
		0.22368	14	0.00194	0.00232	0.01	81.48
M23	(0.50,0.50)	0.16151	35	0.00376	0.00438	0.17	205.17
		0.09501	28	0.00328	0.00379	0.01	177.26
M23	(0.90,0.10)	0.16183	53	0.00794	0.00953	0.25	210.18
		0.05759	21	0.00323	0.00388	0.01	84.65
M23	(1.00,0.00)	1.00000	0	0.00000	0.00000	0.21	0.00
		0.00000	0	0.00000	0.00000	0.02	0.00
M23BB	(0.10,0.90)	0.15705	242	0.00102	0.00118	5.43	43.59
		0.04213	66	0.00026	0.00031	0.24	11.77
M23BB	(0.50,0.50)	0.15892	337	0.00114	0.00128	5.44	62.88
		0.02905	71	0.00031	0.00034	0.23	13.77
M23BB	(0.90,0.10)	0.17352	327	0.00092	0.00103	5.23	63.12
		0.02817	69	0.00021	0.00027	0.27	14.18

† The second value in each cell is the standard deviation of the metric

Table 4.10: The cost of the algorithms for a table of dimension $2 \times 3 \times 3$, with small values for cell entries for which basic moves does not provide an irreducible Markov chain, T_{nir}. The number of MCMC sample is $N = 100 \times 1,000$

Algorithm	(p_4, p_6)	Cost Lower	Cost Upper	N-Bits Total	N-Bern	N-Bits Sp	N-Bits So
M2BB	(1.00,0.00)	311,049	311,644	310,520	562	166,449	135,277
		13,102	13,116	13,096	19	6,742	6,542
M23	(0.10,0.90)	5,484	6,531	4,488	1,008		
		45	43	41	3		
M23	(0.50,0.50)	7,495	8,544	6,455	1,045		
		75	78	73	6		
M23	(0.90,0.10)	95,201	105,421	92,099	10,933		
		172	1783	159	28		
M23BB	(0.10,0.90)	211,112	244,635	181,332	38,755	71,423	96,377
		8,282	9,815	6,754	1,601	2,788	4,132
M23BB	(0.50,0.50)	267,203	303,557	230,821	36,886	108,102	116,152
		11,033	12,629	9,736	1,472	4,273	5,422
M23BB	(0.90,0.10)	332,831	376,927	299,027	35,924	155,276	133,223
		14,798	17,002	13,351	1,524	6,623	6,754

† The second value in each cell is the standard deviation of the metric

4.4 Higher dimension 2×J×K tables

Now we consider the effect of the dimension of the tables on the performance of the proposed algorithms. We use a table of dimension $2 \times 5 \times 6$, T_B, to compute the p-value and the efficiency of the algorithms. We will apply Besag and Bunea's idea using M2BB algorithm. It should be note that M23 algorithm does not necessarily provide an irreducible Markov chain for T_B. To ensure that the Markov chain is irreducible, we must choose moves from a set of basic moves of degree 4, 6, 8, and 10. Hence, we extend the M23 algorithm to MMs algorithm, which chooses moves from M^*.

Algorithms of MMs and MMsBB

For a table of dimension $2 \times 5 \times 6$, an irreducible Markov chain can be achieved without leaving $\mathcal{S}$ by choosing moves from M_4 and M_6, M_8, M_{10}. The proportion of moves selected from M_{2k}, p_k, can assume values in $(0, 1)$ and $\sum_k p_k = 1$. In this algorithm, which we call MMs, the proposals having negative cells are automatically rejected. The algorithm M23 is a special case of MMs, where $\boldsymbol{p} = (p_1, p_2, 0, \ldots, 0)$.

In the same way, extend M23BB to create the MMsBB algorithm which is allowed to visit $\mathcal{S}'$. As before, for this algorithm we will compute the number of bits required when the algorithm makes moves in $\mathcal{S}'$ and when it makes proposals outside $\mathcal{S}'$. The algorithm M23BB is an special case of MMsBB, where $\boldsymbol{p} = (p_1, p_2, 0, \ldots, 0)$.

4.4.1 A case study for 2×5×6 tables

We have already applied the algorithms for tables of dimension $2 \times 3 \times 3$. We now work with the larger dimension $2 \times 5 \times 6$, chosen because it is clearly larger than $2 \times 3 \times 3$ but without being so large that presentation becomes difficult. We will consider two tables with differing magnitudes of cell entries and the performance of the algorithms will be evaluated.

An arbitrary table is chosen such that it has some cells with small frequencies, and also

with 14 cells equal to zero.

T_A :

2	3	4	2	0	0
4	4	6	2	1	0
1	1	3	0	4	0
5	4	4	3	2	2
2	2	1	2	0	1

2	3	2	1	0	1
1	0	1	2	1	0
5	0	8	0	1	0
3	1	2	0	1	0
5	0	2	0	1	0

When we apply the M2BB algorithm for T_A, the algorithms spend a considerable amount of time making moves in $\mathcal{S}'$ and proposals outside $\mathcal{S}'$. For this reason, it takes a long time until a sufficient sample is generated. It takes around 3 hours and 20 minutes to generate only 10 MCMC samples using M2BB. Assuming that time spent for the algorithm linearly increases, it would take more than a month to generate 100,000 sample. A similar situation takes place for the MMsBB algorithm. Table 4.11 and Table 4.12 show the result for the MMs algorithm.

Table 4.11: The efficiency of the algorithms for a table of dimension $2 \times 5 \times 6$, with small values for cell entries, T_A. The number of MCMC sample is $N = 100 \times 1,000$

Algorithm	(p_4, p_6)	Estimated p-value	N_{eff}	Eff Lower	Eff Upper	Time (Sec)	Eff Time
M2BB	(1.0,.00,.00,.00)						
MMs	(.40,.30,.20,.10)	0.30072	70	0.00039	0.00042	2.46	28.67
		0.05256	10	0.00006	0.00006	0.05	4.26
MMs	(.25,.25,.25,.25)	0.30492	54	0.00031	0.00033	2.48	21.62
		0.05693	11	0.00006	0.00007	0.06	4.30
MMs	(.10,.20,.30,.40)	0.28527	41	0.00024	0.00026	2.43	16.72
		0.06678	8	0.00005	0.00005	0.04	3.18
M23BB	(.40,.30,.20,.10)						
MMsBB	(.25,.25,.25,.25)						
MMsBB	(.10,.20,.30,.40)						

† The second value in each cell is the standard deviation of the metric

As MMs only generates moves over $\mathcal{S}$, in contrast to the other algorithms, it enables us to compute the p-value. We use three different sets of probability for choosing moves. The obvious possibility is assign the same chance for each different type of move, that is $\boldsymbol{p} = (0.25, 0.25, 0.25, 0.25)$. An alternative option, suggested by the outcome for 2x3x3 tables, is to use $\boldsymbol{p} = (0.1, 0.2, 0.3, 0.4)$ which assigns more chance to moves of higher degree. Finally, as a contrast, we use an allocation of probabilities which gives more chance to moves of small degrees, that is $\boldsymbol{p} = (0.4, 0.3, 0.2, 0.1)$. Both efficiency measures suggest $\boldsymbol{p} = (0.4, 0.3, 0.2, 0.1)$ as the arrangement that gives the highest efficiency.

Table 4.12: The cost of the algorithms for a table of dimension $2 \times 5 \times 6$, with small values for cell entries, T_A. The number of MCMC sample is $N = 100 \times 1,000$

Algorithm	(p_4, p_6, p_8, p_{10})	Cost Lower	Cost Upper	N-Bits Total	N-Bern	N-Bits Sp	N-Bits So
M2BB	(1.0,.00,.00,.00)						
MMs	(.40,.30,.20,.10)	168,318 238	179,603 250	157,033 232	11,285 39		
MMs	(.25,.25,.25,.25)	163,164 236	174,142 250	152,187 229	10,977 42		
MMs	(.10,.20,.30,.40)	158,051 232	168,726 249	147,376 218	10,675 34		
M23BB	(.40,.30,.20,.10)						
MMsBB	(.25,.25,.25,.25)						
MMsBB	(.10,.20,.30,.40)						

† The second value in each cell is the standard deviation of the metric

We repeat this analysis for another table of dimension $2 \times 5 \times 6$, this time with moderate values and only one cell with zero entry. The results of the MCMC algorithms are given in Table 4.13 and Table 4.14. This time the M2BB algorithm provides results in reasonable time. Although the efficiency of the algorithms using the Bunea and Besag approach is much lower than the MMs algorithm, because of the time spent in S, they provide the p-value.

T_B :

2	3	4	2	2	10
4	4	6	2	1	4
1	1	3	0	4	9
5	4	4	3	2	2
2	2	1	2	5	1

2	3	2	1	10	1
1	2	1	2	1	2
5	4	8	1	1	10
3	1	2	16	1	8
5	3	2	12	1	7

4.5 Tables of dimension 3×3×K

In this section we assess the efficiency of MCMC methods introduced in Chapter 3 for computing the exact p-value for a table of dimension $3 \times 3 \times K$, where $K \geq 5$. We use an example of a $3 \times 3 \times 5$ table to investigate the methodology introduced by Bunea and Besag (2000). We compare them to the results from the MCMC method based on the method introduced by Aoki and Takemura (2003), where moves will be selected from the set of indispensable basic moves, described earlier in section 3.2, which from the unique minimal Markov Basis. We have produced the `mcmc.at` function in `R` (see Appendix A.2) to implement this algorithm. The measures of efficiency are computed for both algorithms.

Table 4.13: The efficiency of the algorithms for a table of dimension $2 \times 5 \times 6$, with moderate values for cell entries, T_B. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6)	Estimated p-value	N_{eff}	Eff Lower	Eff Upper	Time (Sec)	Eff Time
M2BB	(1.0,.00,.00,.00)	0.01642	18	8.18E-06	8.18E-06	32.63	0.56
		0.01833	5	2.87E-06	2.88E-06	5.20	0.20
MMs	(.40,.30,.20,.10)	0.01764	15	7.81E-04	8.52E-04	0.27	53.88
		0.01680	5	2.47E-04	2.70E-04	0.01	17.57
MMs	(.25,.25,.25,.25)	0.01826	14	7.50E-04	8.19E-04	0.27	51.54
		0.02389	4	2.31E-04	2.53E-04	0.01	15.63
MMs	(.10,.20,.30,.40)	0.01771	14	8.01E-04	8.76E-04	0.27	53.40
		0.01939	5	2.62E-04	2.87E-04	0.01	17.66
M23BB	(.40,.30,.20,.10)	0.01689	16	3.34E-06	3.54E-06	60.94	0.27
		0.01544	5	1.12E-06	1.19E-06	7.76	0.09
MMsBB	(.25,.25,.25,.25)	0.01642	16	3.07E-06	3.26E-06	73.30	0.22
		0.01642	5	1.03E-06	1.09E-06	10.91	0.07
MMsBB	(.10,.20,.30,.40)	0.01660	15	3.02E-06	3.21E-06	98.57	0.15
		0.01603	4	9.16E-07	9.74E-07	13.00	0.05

† The second value in each cell is the standard deviation of the metric

Table 4.14: The cost of the algorithms for a table of dimension $2 \times 5 \times 6$, with moderate values for cell entries, T_B. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6, p_8, p_{10})	Cost Lower	Cost Upper	N-Bits Total	N-Bern	N-Bits Sp	N-Bits So
M2BB	(1.0,.00,.00,.00)	2,231,655	2,232,463	2,230,847	808	1,500,525	721,433
		344,424	344,425	344,422	11	231,106	113,432
MMs	(.40,.30,.20,.10)	17,268	18,838	15,698	1,570		
		75	89	70	29		
MMs	(.25,.25,.25,.25)	16,771	18,322	15,220	1,551		
		82	90	81	25		
MMs	(.10,.20,.30,.40)	16,278	17,807	14,749	1,529		
		75	92	66	29		
M23BB	(.40,.30,.20,.10)	4,630,247	4,908,295	4,352,198	278,048	1,652,587	874,256
		594,578	630,197	558,960	35,619	260,214	124,559
MMsBB	(.25,.25,.25,.25)	4,858,858	5,159,274	4,558,441	300,416	2,339,782	2,203,431
		651,205	691,360	611,051	40,154	312,682	298,535
MMsBB	(.10,.20,.30,.40)	4,698,271	4,997,696	4,398,846	299,425	2,037,754	2,346,350
		573,138	609,561	536,714	36,424	246,142	290,824

† The second value in each cell is the standard deviation of the metric

In the Aoki and Takemura algorithm, AT algorithm, we choose different proportion over the range of parameter values to determine whether there is a trend or optimum value for the proportion. Below both algorithm are applied over the following table, T_D:

T_D :

14	12	18	14	18
18	18	10	18	12
12	12	18	16	16

14	10	12	12	22
10	14	10	14	12
16	12	14	18	20

12	10	12	10	12
12	12	12	16	10
14	14	10	20	24

Table 4.15 and Table 4.16 show the results of the MCMC algorithms for the given table.

Estimated p-values using the M2BB and AT algorithms are close, considering the standard deviation of the p-value. The efficiency measures have been computed based on 100,000

Table 4.15: The efficiency of the algorithms for a table of dimension $3 \times 3 \times 5$, with moderate values for cell entries, T_D. The number of MCMC sample is $N = 100 \times 1,000$

Algorithm	(p_4, p_6)	Estimated p-value	N_{eff}	Eff Lower	Eff Upper	Time (Sec)	Eff Time
M2BB	(1.0,.00,.00,.00)	0.01535	84	0.00066	0.00070	2.18	38.48
		0.00847	11	0.00009	0.00009	0.11	5.41
AT	(.40,.30,.20,.10)	0.96317	127	0.00121	0.00145	2.75	46.29
		0.00871	15	0.00014	0.00017	0.06	5.53
AT	(.25,.25,.25,.25)	0.96492	139	0.00144	0.00174	2.86	48.77
		0.00930	16	0.00017	0.00020	0.06	5.72
AT	(.10,.20,.30,.40)	0.96380	147	0.00165	0.00204	2.95	49.88
		0.00874	16	0.00018	0.00022	0.09	5.50

† The second value in each cell is the standard deviation of the metric

Table 4.16: The cost of the algorithms for a table of dimension $3 \times 3 \times 5$, with moderate values for cell entries, T_D. The number of MCMC sample is $N = 100 \times 1,000$

Algorithm	(p_4, p_6, p_8, p_{10})	Cost Lower	Cost Upper	N-Bits Total	N-Bern	N-Bits Sp	N-Bits So
M2BB	(1.0,.00,.00,.00)	120,091	126,395	113,787	6,304	8	1
		97	114	90	35	82	8
AT	(.40,.30,.20,.10)	88,120	104,893	71,346	16,773		
		214	223	210	33		
AT	(.25,.25,.25,.25)	80,042	96,953	63,132	16,911		
		186	189	189	34		
AT	(.10,.20,.30,.40)	71,934	88,970	54,898	17,036		
		131	140	130	33		

† The second value in each cell is the standard deviation of the metric

(100 chain of the size 1,000) sample for M2BB and AT algorithm. From this table it is quite clear that the effective sample size, and consequently efficiency, increases when the algorithm accepts more moves with higher degrees.

4.6 Tables of dimension 3×4×4

In this section we assess the efficiency of MCMC methods introduced in Chapter 3 for computing the exact p-value for a table of dimension $3 \times 4 \times 4$. We use two examples of $3 \times 4 \times 4$ tables to investigate the methodology introduced by Bunea and Besag (2000). Similarly to the previous section, the Bunea and Besag algorithm will be compared to Aoki and Takemura approach in which moves are selected from the minimum set of indispensable basic moves of unique minimal Markov Basis. The measures of efficiency are computed for both algorithms. In the Aoki and Takemura algorithm we choose a range of parameter values to determine whether there is an trend or optimum value for the proportions of different kinds of moves chosen. The algorithms are applied to the

following tables: T_E with small entries and T_F with moderate cell entries.

T_E :

2	4	4	0
4	1	2	3
1	4	4	0
1	0	3	2

1	1	1	1
1	2	5	1
0	4	1	0
2	6	1	0

3	7	2	2
5	2	5	5
1	3	1	5
0	4	2	5

T_F :

12	12	12	14
18	12	14	12
12	18	18	16
12	10	16	14

12	12	12	14
12	14	20	20
10	18	12	12
14	22	12	14

16	24	14	16
20	14	20	24
12	16	12	18
10	18	14	12

Results for T_E are shown in Table 4.17 and Table 4.18. For this table with small entries and some zeros the M2BB fails to provide the p-value due to the long running time, at least 48 hours. However we can observe the behaviour of the AT algorithm with different probability settings. Efficiency is higher when a higher proportion of simpler moves is used.

Results for T_F are shown in Table 4.17 and Table 4.18. For this table M2BB provides the p-value. The efficiency measures show that the M2BB is not much less efficient than AT. Unlike T_E, AT becomes more efficient as the proportions of simple moves decreases.

Table 4.17: The efficiency of the algorithms for a table of dimension $3 \times 4 \times 4$, with small values for cell entries, T_E. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6, p_8)	Estimated p-value	N_{eff}	Eff Lower	Eff Upper	Time (Sec)	Eff Time
M2BB	(1.0,.00,.00)						
AT	(.50,.30,.20)	0.18283	131	0.001413	0.001645	3.53	37.35
		0.02788	15	0.000159	0.000186	0.10	4.38
AT	(.34,.33,.33)	0.18176	120	0.001423	0.001683	3.58	33.52
		0.02751	15	0.000176	0.000209	0.07	4.32
AT	(.20,.30,.50)	0.17811	107	0.001461	0.001767	3.63	29.50
		0.03399	14	0.000182	0.000221	0.06	3.78

† The second value in each cell is the standard deviation of the metric

4.7 Conclusion

In this chapter we compared the efficiency of various algorithms (M23, M23BB, MMs, MMsBB and AT) to the efficiency of the Bunea and Besag basic algorithm. We first computed the efficiency of these algorithm for tables of dimension $2 \times J \times K$. As a special

Table 4.18: The cost of the algorithms for a table of dimension $3 \times 4 \times 4$, with small values for cell entries, T_E. The number of MCMC sample is $N = 100 \times 1,000$

Algorithm	(p_4, p_6, p_8)	Cost Lower	Cost Upper	N-Bits Total	N-Bern	N-Bits Sp	N-Bits So
M2BB	(1.0,.00,.00)						
AT	(.50,.30,.20)	80,120	93,411	66,829	13,290		
		270	312	243	74		
AT	(.24,.33,.33)	71,257	82,264	58,251	13,006		
		238	274	219	70		
AT	(.20,.30,.50)	60,651	73,362	47,940	12,711		
		281	320	258	74		

† The second value in each cell is the standard deviation of the metric

Table 4.19: The efficiency of the algorithms for a table of dimension $3 \times 4 \times 4$, with moderate values for cell entries, T_F. The number of MCMC sample is $N = 100 \times 1,000$

Algorithm	(p_4, p_6, p_8)	Estimated p-value	N_{eff}	Eff Lower	Eff Upper	Time (Sec)	Eff Time
M2BB	(1.0,.00,.00)	0.91090	89	0.00082	0.00088	3.44	25.87
		0.01603	11	0.00010	0.00011	0.03	5.47
AT	(.50,.30,.20)	0.91593	104	0.00104	0.00125	3.71	28.24
		0.01701	14	0.00013	0.00016	0.08	3.82
AT	(.34,.33,.33)	0.91207	108	0.00118	0.00145	3.77	28.92
		0.01606	12	0.00013	0.00017	0.14	3.53
AT	(.20,.30,.50)	0.91518	116	0.00142	0.00179	3.85	30.12
		0.01626	13	0.00015	0.00020	0.07	3.35

† The second value in each cell is the standard deviation of the metric

Table 4.20: The cost of the algorithms for a table of dimension $3 \times 4 \times 4$, with moderate values for cell entries, T_F. The number of MCMC sample is $N = 100 \times 10,000$

Algorithm	(p_4, p_6, p_8)	Cost Lower	Cost Upper	N-Bits Total	N-Bern	N-Bits Sp	N-Bits So
M2BB	(1.0,.00,.00)	101,262	107,709	94,814	6,447	0	0
		35	69	222	35	0	0
AT	(.50,.30,.20)	83,478	100,140	66,816	16,661		
		237	245	235	35		
AT	(.34,.33,.33)	75,018	91,749	58,287	16,731		
		317	328	311	34		
AT	(.20,.30,.50)	64,730	81,554	47,906	16,823		
		291	287	299	32		

† The second value in each cell is the standard deviation of the metric

case we used three examples of $2 \times 3 \times 3$ tables with small, moderate and large sample size. In all $2 \times J \times K$ examples, results show that the Besag and Bunea method provides a somewhat less efficient way to generate the tables and compute the p-value. This lack of efficiency is mostly due to the time (number of bits) that the algorithm spends when it is out of $\mathcal{S}$. Ignoring the number of bits (or time) spent in computing the efficiency gives us larger efficiency measures. On the other hand, the Bunea and Besag algorithm enables us to compute the p-value for tables of different dimension without knowing the set of indispensable basic moves.

The running time for the Bunea and Besag algorithm is always higher for tables with very small values (including cells with zero entries). When the number of cells of the table with small entries increases the Bunea and Besag method would fail to provide a p-value, as the algorithm spends a considerable amount of time out of reference set $\mathcal{S}$. In practice we would never be able to reach an acceptable effective sample size.

The modified version of Bunea and Besag, which accepts moves both from M_4 and M_6 does increase the efficiency when one is able to choose an optimum proportion of each types of move. Using tables of different sample size and cell values shows that for tables with small cell entries choosing basic moves of degree 4 increases the efficiency of the algorithm. For tables with moderate and large cell entries the pattern is totally different, i.e. choosing moves of higher degree increases the efficiency of the M2BB algorithm.

Having tables with many zero cells causes the algorithm to spend a considerable time out of the reference set $\mathcal{S}$ so that it has very high cost and, sometimes fail to gives a required sample size in reasonable running time. This problem is more significant when the dimension of the table increases, as we observed for a table with dimension of $2 \times 5 \times 6$ with zero cell entries we could not use the M23BB algorithm.

We used two different approaches for computing the efficiency of the algorithms: one used the running time as a cost of the algorithm, and the second one counts the number of bits used as the cost. We observed that these approach do not necessarily lead us to the same answer when comparing the efficiency of the algorithms. In tables of dimension $2 \times J \times K$ the efficiency of the algorithm decreases when the proportion of moves chosen of higher degree increases. Using the running time efficiency shows a different pattern for tables with small size cell entries versus tables with moderate and large cell entries. That is, the running time efficiency increases by choosing more moves of higher degrees, whilst for tables with moderate and large cell entries the running time efficiency measure decreases when more moves of higher degree are chosen.

For tables of dimension $3 \times J \times K$ we observe a different pattern of efficiency for small cell entries versus tables with moderate values for cell entries. For small samples it is more advantageous to choose moves with smaller degrees, whilst for moderate tables choosing moves of higher degrees increase the efficiency of the algorithm.

Chapter 5

Conclusion

5.1 Inference for contingency tables

There are several different approaches to analysing categorical data when represented through contingency tables. The most common model to be investigated in two-way contingency tables is the independence model in which row and column variables are assumed to be independent. We reviewed different types of sampling distribution to represent the probabilistic behaviour of contingency tables: Poisson sampling, multinomial, and product of multinomial distributions. We also reviewed the role of conditional distributions in making inference about the models in contingency tables.

Many statistical tests on contingency tables rely on large sample distribution theory where for tables with small sample size the limiting distribution does not provide a suitable approximation for the test statistic. In this case, the Fisher exact test enables us to compute the exact p-value for the independence test. The number of possible tables increases as row and column or the total sample size increases so that using Fisher's methods becomes infeasible. Hence, we reviewed another class of methods for computing the exact p-value, known as enumeration methods. An early example of enumeration methods was introduced by Pagano and Halvorsen (1981), referred to as the G-algorithm. We also reviewed the so called Network algorithm introduced by Mehta and Patel (1983) and explained it with an example. This method helps to compute the exact p-value without enumerating all possible tables in the reference set. It should be noted that, by increasing the dimension or sample size of the table even the G-algorithm and the Network algorithm fail to compute the exact p-value in a reasonable time because there are too

many tables in the reference set. Also, although we have explained the enumeration methods as an approach to finding the exact p-value, it has not been the main interest in this thesis.

The Monte Carlo approach has also been used to compute exact p-value. This approach does not need complete enumeration, or asymptotic approximation, to compute the exact p-value. On the contrary, it takes a sample from an exact probability distribution of the test statistic under the null hypothesis to test the hypothesis of interest and can be applied in multi dimensional contingency tables. The exact distribution of the test statistic has received a great deal of attention in studies of the independence model, but there are many other models for which the distribution of the test statistic is needed to be studied. In many cases this is not simple. For this reason Monte Carlo Markov chain can be applied to generate samples from conditional distributions. So when simple Monte Carlo tests are not available, an MCMC procedure can help as an alternative. The MCMC method has been the core of the methods we have used throughout this thesis. Hence, this approach has been explained in more detail in the first chapter.

5.2 Developing MCMC exact methods for three-way tables

Chapter 2 mainly focused on the specific MCMC approach proposed by Bunea and Besag (2000) for tables of dimension $2 \times J \times K$. This method uses random basic moves to generate samples from a reference set. The key idea is to allow tables to be sampled from a set of all tables with the same two-way marginal totals as the observed table, having all non-negative entries except for at most a single -1. For the proof of the irreducibility of this set, they use the Rasch model and they also refer to Diaconis and Sturmfels (1998), where an irreducible set of moves is defined as $\mathcal{M}^*$. For this reason we have explained the Rasch model in detail. We also provided a detailed proof for the propositions given in Bunea and Besag (2000).

We also introduced a direct proof which shows that $\mathcal{M}^*$ is an irreducible Markov basis. We also provided detailed proof that irreducible Markov chains can be made over $\mathcal{S}$ using only basic moves, allowing at most two negative cells which are -1 and there is no need to use $\mathcal{M}^*$. These results were used later to prove that the Bunea and Besag approach can provide an irreducible Markov chain for $3 \times 3 \times K$ and $3 \times 4 \times 4$ tables. The Bunea and

Besag approach is also presented in the form of a flowchart to clarify their algorithm. The R codes for implementing the Besag and Bunea method were produced (see Appendix) and applied for a sample table of $2 \times 3 \times 3$ dimension.

Aoki and Takemura (2003) introduced a Minimal Markov basis which contains different types of move to make an irreducible Markov chain over the space of particular tables and each particular move has been defined and represented in a three dimensional view. It has been also clarified that not all types of move of a particular degree are needed in the Markov basis. We explained the Aoki and Takemura minimal Markov basis in Chapter 3.

The main innovation in Chapter 3 was to construct the irreducible Markov chain for tables of $3 \times 3 \times K$ and $3 \times 4 \times 4$ using only basic moves allowing intermediate tables to have at most two -1s. The proof is based on the property of the difference between two tables that can be constructed by a number of basic moves for any table of dimension $I \times J \times K$. This way, the Bunea and Besag algorithm has been generalised to higher dimension tables.

5.3 Efficiency study

In Chapter 4 we investigated the efficiency of the Bunea and Besag algorithm compared to other MCMC approaches. We used two different approaches to compute the efficiency of the algorithms. One was the classic way of measuring efficiency using running time of the algorithm. The second was to compute the cost of the algorithm by counting the number of bits used during the algorithm. We observed that these two efficiencies do not necessarily give the same result for some circumstances when comparing the efficiency of the algorithms.

An innovation in this chapter was the introduction of a modified version of Besag and Bunea algorithm in which we chose basic moves of all different degrees rather than only degree of 4. As the chance of choosing moves of different degrees can vary, we chose three scenarios for the chance allocated to the moves of different degree.

The efficiency was calculated for several settings with a study of the efficiency of the algorithm for tables of small, moderate, and large dimensions. This was also implemented for tables with small, moderate, and large sample size. The effect of special zero patterns on the efficiency of algorithms was also investigated.

Our study shows that the Bunea and Besag method provides a less efficient way of gen-

erating the tables and computing the exact p-value. The reason for this lack of efficiency is due to the time (number of bits) the algorithm spends when it is out of $\mathcal{S}$. It should be noted that having ignored the number of bits (or time) outside $\mathcal{S}$, the Bunea and Besag algorithm outperforms the other methods. On the other hand, one advantage of the Bunea and Besag algorithm over the Aoki and Takemura approach is that it enables us to compute the p-value for tables of different dimension without knowing the set of indispensable basic moves.

The running time for the Bunea and Besag algorithm is always higher for tables with very small values (including cells with zero entries). When the number of cells of the table with small entries increases the Bunea and Besag Method will fail to provide a p-value, as the algorithm spends a considerable amount of time out of reference set $\mathcal{S}$. Because of this, in practice we will never be able to reach an acceptable effective sample size. This problem is more significant when the dimension of the table increases.

The modified Bunea and Besag version accepts moves from $M_4, M_6, \ldots, M_{2\min J,K}$. This modified version can outperform the original one if we choose the optimum proportion for the moves of different degree.

For tables of dimension $3 \times 3 \times K$ and $3 \times 4 \times 4$ we observe a different pattern of efficiency for small cell entries versus tables with moderate value for cell entries. For small samples it is more advantageous to choose moves with smaller degrees, whilst for moderate and large tables choosing moves of higher degree increases the efficiency of the algorithm.

5.4 Further work

The Bunea and Besag (2000) introduced a method for $2 \times J \times K$. We could prove that this method also provides an irreducible Markov Chain for tables of $3 \times 3 \times K$ and $3 \times 4 \times 4$. Further investigation is necessary to determine whether this method can provide an irreducible Markov chain for tables of higher dimension. On the other hand, the main focus throughout this thesis has been on the independence model of contingency tables of dimension $I \times J \times K$. It would be quite useful to expand our study and discussion to more general types of model of contingency tables. The efficiency study could also be extended to different types of model.

The efficiency study showed us that the Bunea and Besag algorithm fails to compute exact p-values in a practical running time for tables with many zero cells. So a further

development could be obtained by finding an adjustment for the Bunea and Besag algorithm which considers these settings. This suggests a need to create an algorithm which would reduce the chance of chain to choose samples from outside $\mathcal{S}$.

Aoki and Takemura (2003) found the set of minimal Markov basis for some specific tables. The efficiency studies showed that the Minimal Markov basis provides a very efficient method for finding exact p-value. They did not introduce a methodology to find the minimal Markov basis. It would be highly advantageous if a methodology enabled us to find the minimal Markov basis for a general contingency table of dimension $I \times J \times K$.

Bibliography

A. Agresti. *Categorical Data Analysis*. Wiley Inter. Science, 1990.

A. Agresti. *An Introduction to Categorical Data Analysis*. Wiley, 1996.

A. H. Anderson. Multidimensional contingency tables. *Scandinavian Journal of Statistics*, 1:115–127, 1974.

S. Aoki and A. Takemura. Minimal basis for connected Markov chain over $3 \times 3 \times K$ contingency tables with fixed two-way marginals. *Australian and New Zealand Journal of Statistics*, 45:229–249, 2003.

G. A. Bernard. Discussion on paper by M.S. Bartlett. *Journal of Royal Statistical Society B*, 25:294, 1963.

J. Besag and P. Clifford. Generalized Monte Carlo significance tests. *Biometrika*, 76: 633–642, 1989.

F. Bunea and J. Besag. MCMC in $I \times J \times K$ contingency tables. In N.H. Madras, editor, *Monte Carlo Methods*, pages 25–36. Fields Institute Communications, 2000.

Luc Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag, New York, 1986.

P. Diaconis and B. Sturmfels. Algebraic algorithms for sampling from conditional distribution. *The Annals of Statistics*, 26:363–397, 1998.

E. A. Erosheva, S.E. Fienberg, and B.W. Junker. Alternative statistical models and representations for large sparse multi-dimensional contingency tables. *Annales de la Faculté des Sciences de Toulouse*, XI:485–505, 2002.

R.A. Fisher. *Statistical Methods for Research Workers*. Oliver & Boyd, Edinburgh, 1934.

D. Gamerman and H.F. Lopes. *Markov chain Monte Carlo, Stochastic Simulation for Bayesian Inference.* Chapman and Hall/CRC, 2006.

C.R. Mehta and N.R. Patel. A network algorithm for performing Fisher's exact test in $r \times c$ contingency tables. *Journal of the American Statistical Association*, 76:931–934, 1983.

M. Pagano and K.T. Halvorsen. An algorithm for finding the exact significance levels of $r \times c$ contingency tables. *Journal of the American Statistical Association*, 78:427–434, 1981.

G. Rasch. *Probabilistic Models for Some Intelligence and Attainment Tests (2nd ed.).* University of Chicago Press, 1980.

H. J. Ryser. Combinatorial mathematics. *The Mathematical Association of America*, 14, 1963.

P.W.F. Smith, J.J. Forster, and J.W. McDonald. Monte Carlo exact tests for square contingency tables. *Journal of the Royal Statistical Society A*, 159:309–321, 1996.

F. Yates. Contingency tables involving small numbers and the χ^2 test. *Journal of Royal Statistical Society, Supplementary*, 1:217–235, 1934.

Appendix A

R Codes

A.1 Algorithms for 2 $\times$J$\times$K tables

```
#------------------ A function to produce random number and its counter
myrunif = function(){
  assign("N.runif", N.runif+1, envir=.GlobalEnv)
  return(runif(1))
}

##------------------ A function to make a list name to apply in Condor
make.list = function(names){
  l = list()
  for(n in names) l[[n]] = get(n, envir=sys.parent())
  l
}

##------------------ A function to Compute test statistics for the produced table

rand = function(xold){
      r = length(xold)
      J = ceiling(r*runif(1))
      u = runif(1)
      delta = (1)*(u<0.5)+ (-1)*(u>=0.5)
      xs = xold
      xs[J] =  xs[J] + delta
      return(xs)
      }


##------------------ A function to produce random number and its counter -------

rbern = function(){
  assign("N.rbern", N.rbern+1, envir=.GlobalEnv)
  return(rbinom(1,1,0.5))
```

```
}

##------------------ A function to compute c(n); number of bits ----------------

bits = function(n){
       m  = ceiling(log(n,2))
       cn = 2^m*(m/n)
       return(cn)
      }


##------------- A simpler function to check acceptance of random sample --------

accept = function(pi){
  if(pi >= 1) return(TRUE)
  assign('N.rbern', N.rbern+1, envir=.GlobalEnv)
  return(runif(1) <= pi)
  }

##------------------ A function to produce random move from basic move sets (M2)

rand.m =  function(x0){
      n = length(x0)
      dim=dim(x0)
      m = array(0, dim=dim)
      i =  sample(1:dim[1],2)
      j =  sample(1:dim[2],2)
      k =  sample(1:dim[3],2)

      m[i[1],j[1],k[1]] =  +1
      m[i[2],j[2],k[1]] =  +1
      m[i[2],j[1],k[1]] =  -1
      m[i[1],j[2],k[1]] =  -1
      m[,,k[2]] = -m[,,k[1]]

      n.moves = choose(dim[1],2)*choose(dim[2],2)*2
      assign("CN",   CN+bits(n.moves),  envir=.GlobalEnv)
      assign("CN.iter",CN.iter+1,    envir=.GlobalEnv)
      return(m)
   }

##------------------ A function to produce random move from M6 -----------------

rand.m3 <- function(x0){
   dims=dim(x0)
   m = array(0, dim=dims)
   i =  sample(1:dims[1],3)
   j =  sample(1:dims[2],3)
   k =  sample(1:dims[3],2)

   m[i[1],j[1],k[1]] =  +1
   m[i[2],j[2],k[1]] =  +1
   m[i[3],j[3],k[1]] =  +1
   m[i[1],j[2],k[1]] =  -1
```

```
   m[i[2],j[3],k[1]] =  -1
   m[i[3],j[1],k[1]] =  -1

   m[,,k[2]] = -m[,,k[1]]
   n.moves = choose(dims[1],3)*choose(dims[2],3)*factorial(3)
   assign("CN",    CN+bits(n.moves),  envir=.GlobalEnv)
   assign("CN.iter",CN.iter+1,    envir=.GlobalEnv)
   return(m)
}

##---- A function to produce random move from M4 and M6 with rpoportion p ------

rand.m23 <- function(x0, p2){
               if (accept(p2))     {m <- rand.m(x0);  dg <- 2}
               else                {m <- rand.m3(x0); dg <- 3}
               return(list(m=m,dg=dg))
            }

##-------------------- Function to find the move of degree 2M, wher M = min(J,K)

rand.Ms <- function(x0){
    dims = dim(x0)
    M = min(dims[1:2])
    Mi=floor(runif(1, min=2, max=M+1))

    m =  array(0, dim=dims)
    Ri =  sample(1:dims[1],Mi)
    Ci =  sample(1:dims[2],Mi)
    Li =  sample(1:dims[3],2)

    for (i in 1:M)      m[Ri[i],Ci[i],  Li[1]] =  +1
    for (i in 1:(M-1))  m[Ri[i],Ci[1+i],Li[1]] =  -1
    m[Ri[Mi],Ci[1], Li[1]] = -1

    m[,,Li[2]] = -m[,,Li[1]]
    n.moves = choose(dims[1],Mi)*choose(dims[2],Mi)*factorial(Mi)
    assign("CN",    CN+bits(n.moves),  envir=.GlobalEnv)
    assign("N.rbern", N.rbern+1, envir=.GlobalEnv)
    return(m)
}

##--------------Function for MCMC Bunea and Besag algorithm
##------------------------------
mcmc.m2bb = function(x0,n){
    stepv   = NULL
    stepspv = NULL
    chisq   = numeric(n)
    xold    = x0
    step    = 0
    stepsp  = 0
    chisq0  = chisq3ind(x0)
    xl = matrix(0, ncol=n, nrow=length(x0))
    for(t in 1:n)
        {
```

```
        if (t%%10000==0) cat(t, "\n")
        chisq[t] =  chisq3ind(xold)
        x =  xold
        m =  rand.m2(x0) #--------------------produce the m as a basic move in M2
        xs =  x+m       #------------ producing subsequent tables by random move
        if(any(xs < (-1)))  #------------------ have proposal outside S': reject
            {
            xl[,t] = xold
            next
            }
        stepsp = 0
        while(any(xs<0)) { # In S'\S: need to wait until return to S
          stepsp = stepsp+1
          m =  rand.m2(x0)
          xs2 = xs+m
          while(any(xs2 < (-1))) { # Outside S': need to keep trying for move staying in S'
            m = rand.m(x0)
            xs2 = xs+m
          }
          xs = xs2
        }
        stepspv = c(stepspv, stepsp)
        ##------------------------ Have proposal in S: do Met-Hast accept reject
        pi =  min(1, exp(sum(lfactorial(xold)-lfactorial(xs))))
        if (accept(pi)) xold =  xs
        xl[,t] = xold
        }
    return(list(xl=xl, chisq=chisq, stepv=stepv, stepspv=stepspv) )
    }

##------------------------------------ Function for MCMC Besag and Bunea method

mcmc.m23bb = function(x0,n,p){
    xold    = x0
    xl      = matrix(0, ncol=n, nrow=length(x0))
    for(t in 1:n){
        if (t%%10000==0) cat(t, "\n")
        x   =  xold
        if(all(x>= 0))  where = 1
        if(any(x<  0))  where = 2
        if(any(x< -1))  where = 3

        m   =  rand.m23(x0,p,where) ## produce the m as a basic move in M2
        xs  =  x+m          ## producing subsequent tables by random move
        if(any(xs < (-1))){
          ## have proposal outside S': reject
          xl[,t] = xold
          next
        }
        stepsp = 0
        while(any(xs<0)){
             # In S'\S: need to wait until return to S
             if(any(xs<  0))  where = 2
             if(any(xs< -1))  where = 3
```

```
            m   = rand.m23(x0,p,where)
            xs2 = xs+m
           while(any(xs2 < (-1))){
               # Outside S': need to keep trying for move staying in S'
                if(any(xs2<  0))  where = 2
                if(any(xs2< -1))  where = 3
                m     = rand.m23(x0,p,where)
                xs2   = xs+m
           }
         xs = xs2
       }
       # Have proposal in S: do Met-Hast accept reject
       pi =  min(1, exp(sum(lfactorial(xold)-lfactorial(xs))))
       if (accept(pi)) xold =  xs
       xl[,t] = xold
       }
   return(xl)
   }


##-------------- Function for MCMC from move of size 2 and 3 disable to go to S'

mcmc.m23 = function(x0,n,p){
    xold   <-  x0
    xl     <- matrix(0,ncol=n,nrow=length(x0))
    for(t in 1:n){
        if (t%%10000==0) cat(t, "\n")
        m     <- rand.m23(x0,p) ## produce the m as a basic move in M2
        x     <- xold
        xl[,t]<- x
        xs <- x+m   ## producing subsequent tables by random move
        if (any(xs<0)){
            x   <- xold
        }
        else{
             pi <- min(1, exp(sum(lfactorial(xold)-lfactorial(xs))))
             if (accept(pi)) xold <- xs
        }
    }
    return(xl=xl)
  }

##--------- compute and plot efficency for camparison of  mcmc.M2bb and mcmc.m23

ess = function(x0,n=10000){
    assign("N.rbern" , 0 , envir=.GlobalEnv)
    library(lattice)
    library(coda)
    T = mcmc.m2bb(x0,n)$xl
    likl = function(x){1/exp(sum(lfactorial(x)))}
    P   = apply(T, MARGIN=2, FUN= likl)
    p0  = likl(as.vector(x0))
    logit =  log(P)
    I   = as.numeric(P <= p0)
```

```
    Ibar = mean(I)
    ne.I = effectiveSize(I)
    n.rbern = N.rbern
    names(ne.I) = NULL
    ne.logit = effectiveSize(logit)
    names(ne.logit) = NULL
    eff.logit = ne.logit/N.rbern
    se.Ibar = sqrt(Ibar*(1-Ibar)/ne.I)
    return(list(Ibar=Ibar, ne.logit=ne.logit, se.Ibar=se.Ibar,
                eff.logit=eff.logit, n.rbern=N.rbern))
    }

##----------- Find the optimum value for the proportion of in algorithm mcmc.m23

OptProp23 = function(x0, n=10000){
    prop = seq(0.00,1,0.01)
    k = length(prop)
    library(splines)
    library(lattice)
    library(coda)
    Ibar      = numeric(k)
    ne.logit  = numeric(k)
    ne.I      = numeric(k)
    se.Ibar   = numeric(k)
    cost.low  = numeric(k)
    cost.upp  = numeric(k)
    ne.I      = numeric(k)
    n.cn      = numeric(k)

    for (i in 1:k)
       {
        assign("N.rbern" , 0 , envir=.GlobalEnv)
        assign("CN" ,      0 , envir=.GlobalEnv)
       output = mcmc.m23(x0, n, prop[i])$xl
       T = matrix(output, ncol=n)
       likl     = function(x){1/exp(sum(lfactorial(x)))}
       P   = apply(T, MARGIN=2, FUN= likl)
       p0  = likl(as.vector(x0))
       logit =  log(P)
       I   = as.numeric(P <= p0)
       Ibar[i] = mean(I)
       ne.I[i] = effectiveSize(I)
       ne.logit[i] = effectiveSize(logit)

       cost.low[i] = CN + 1* N.rbern
       cost.upp[i] = CN + 2* N.rbern
        se.Ibar[i]  = sqrt(Ibar[i]*(1-Ibar[i])/ne.I[i])
        n.cn[i]     = CN
       }
        return(list(prop=prop,          Ibar=Ibar,
                    se.Ibar=se.Ibar,    ne.logit=ne.logit,    n.cn=n.cn,
                    cost.low=cost.low,  cost.upp=cost.upp
                    )
               )
```

```
    }

##--------- Find the optimum value for the proportion of in algorithm mcmc.m23bb

OptProp23bb = function(x0, n=10000){
    prop = seq(0.00,1,0.01)
    k = length(prop)
    library(splines)
    library(lattice)
    library(coda)
    Ibar      = numeric(k)
    ne.logit  = numeric(k)
    ne.I      = numeric(k)
    se.Ibar   = numeric(k)
    cost.low  = numeric(k)
    cost.upp  = numeric(k)
    n.cn      = numeric(k)
    n.accept  = numeric(k)


    for (i in 1:k)
       {
        assign("N.accept", 0 , envir=.GlobalEnv)
        assign("CN" ,      0 , envir=.GlobalEnv)
       output = mcmc.m23bb(x0, n, prop[i])$xl
       T = matrix(output, ncol=n)
       likl     = function(x){1/exp(sum(lfactorial(x)))}
       P   = apply(T, MARGIN=2, FUN= likl)
       p0  = likl(as.vector(x0))
       Logit =  log(P)
       I      = as.numeric(P <= p0)
       Ibar[i] = mean(I)
       ne.I[i] = effectiveSize(I)
       ne.logit[i] = effectiveSize(Logit)

       cost.low[i] = CN + 1* N.rbern
       cost.upp[i] = CN + 2* N.rbern
        se.Ibar[i]  = sqrt(Ibar[i]*(1-Ibar[i])/ne.I[i])
        n.cn[i]     = CN
       }
        return(list(prop=prop,          Ibar=Ibar,    ne.logit=ne.logit,
                    se.Ibar=se.Ibar,    cost.low=cost.low,
                    cost.upp=cost.upp,  n.cn=n.cn))
}

#------------------------------------ Diaconis and Sturmlfles Method for 2xJxK

eff.dc = function(x0, n=10000){
    library(splines)
    library(lattice)
    library(coda)

    assign("N.bern",   0 , envir=.GlobalEnv)
```

```
    assign("CN" ,      0 , envir=.GlobalEnv)
   output = mcmc.ds(x0, n)$xl
   T      = matrix(output, ncol=n)
   likl   = function(x){1/exp(sum(lfactorial(x)))}
   P      = apply(T, MARGIN=2, FUN= likl)
   p0     = likl(as.vector(x0))
   logit  = log(P)
   I      = as.numeric(P <= p0)
   Ibar   = mean(I)
   ne.I   = effectiveSize(I)
   ne.logit = effectiveSize(logit)

   cost.low = CN + 1* N.rbern
   cost.upp = CN + 2* N.rbern
    se.Ibar  = sqrt(Ibar*(1-Ibar)/ne.I)
    n.cn     = CN
    return(list(Ibar=Ibar,    se.Ibar=se.Ibar,    ne.logit=ne.logit,    cost.low=cost.low,
                cost.upp=cost.upp,  n.cn=n.cn))
}
```

A.2 R codes for complete reference set

```
 #---------------------- Calculation for the exact p-value for tables Ts
 #-----------------------------------

i <- 0
x.all <- NULL
for (x5 in 0:10)
for (x6 in 0:10)
for (x8 in 0:10)
for (x9 in 0:10)
    {

    x1  = -5 + x5 + x8 + x6 + x9
    x10 =  9 - x5 - x8 - x6 - x9
    x11 =  2 + x5 + x8
    x12 = -3 + x6 + x9
    x13 =  3 + x5 + x6
    x14 = -x5 + 4
    x15 = -x6 + 5
    x16 = -2 + x8 + x9
    x17 = -x8 + 3
    x18 = -x9 + 7
    x2  = 4 - x5 - x8
    x3  = 9 - x6 - x9
    x4  = 6 - x5 - x6
    x7  = 8 - x8 - x9
    x5  = x5
    x6  = x6
```

```
    x8  = x8
    x9  = x9

    foo <- c(x1, x4, x7 ,   x2, x5, x8,   x3, x6, x9,
             x10,x13,x16,   x11,x14,x17,  x12,x15,x18)
    x.all <- rbind(x.all, foo)
}
ok  <- apply(x.all, MARGIN=1, FUN=function(x){all(x>=0)})
ref <- x.all[ok,]
rownames(ref) <-NULL
dim(ref)
x0 <- array(c(  2,  4,  3,    2,  1,  1,    4,  1,  4,
                2,  5,  3,    4,  3,  2,    2,  4,  3), dim=c(3,3,2))

fun.foo <- function(x){1/prod(factorial(x))}
ptab <- apply(ref, MARGIN=1, FUN=myprob)
tot.lik <- sum(ptab)
ptab <- ptab/tot.lik
rownames(ptab) <- NULL
p0 <- (1/prod(factorial(x0)))/tot.lik
loc <- which(apply(ref, MARGIN=1,  FUN=function(x){all(x==as.vector(x0))}))
ref
sum(ptab[ptab<=ptab[loc]])
#array(ref[1,],dim=c(3,3,2))

#-------------------------------------------------------------------------------------------------
> ref
       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18]
  [1,]    0    6    3    4    0    0    4    0    5     4     3     3     2     4     3     2     5     2
  [2,]    1    6    2    4    0    0    3    0    6     3     3     4     2     4     3     3     5     1
  [3,]    2    6    1    4    0    0    2    0    7     2     3     5     2     4     3     4     5     0
  [4,]    0    6    3    3    0    1    5    0    4     4     3     3     3     4     2     1     5     3
  [5,]    1    6    2    3    0    1    4    0    5     3     3     4     3     4     2     2     5     2
  [6,]    2    6    1    3    0    1    3    0    6     2     3     5     3     4     2     3     5     1
  [7,]    3    6    0    3    0    1    2    0    7     1     3     6     3     4     2     4     5     0
  [8,]    0    6    3    2    0    2    6    0    3     4     3     3     4     4     1     0     5     4
  [9,]    1    6    2    2    0    2    5    0    4     3     3     4     4     4     1     1     5     3
 [10,]    2    6    1    2    0    2    4    0    5     2     3     5     4     4     1     2     5     2
 [11,]    3    6    0    2    0    2    3    0    6     1     3     6     4     4     1     3     5     1
 [12,]    1    6    2    1    0    3    6    0    3     3     3     4     5     4     0     0     5     4
 [13,]    2    6    1    1    0    3    5    0    4     2     3     5     5     4     0     1     5     3
 [14,]    3    6    0    1    0    3    4    0    5     1     3     6     5     4     0     2     5     2
 [15,]    0    5    4    4    0    0    4    1    4     4     4     2     2     4     3     2     4     3
 [16,]    1    5    3    4    0    0    3    1    5     3     4     3     2     4     3     3     4     2
 [17,]    2    5    2    4    0    0    2    1    6     2     4     4     2     4     3     4     4     1
 [18,]    3    5    1    4    0    0    1    1    7     1     4     5     2     4     3     5     4     0
 [19,]    0    5    4    3    0    1    5    1    3     4     4     2     3     4     2     1     4     4
 [20,]    1    5    3    3    0    1    4    1    4     3     4     3     3     4     2     2     4     3
 [21,]    2    5    2    3    0    1    3    1    5     2     4     4     3     4     2     3     4     2
 [22,]    3    5    1    3    0    1    2    1    6     1     4     5     3     4     2     4     4     1
 [23,]    4    5    0    3    0    1    1    1    7     0     4     6     3     4     2     5     4     0
 [24,]    0    5    4    2    0    2    6    1    2     4     4     2     4     4     1     0     4     5
 [25,]    1    5    3    2    0    2    5    1    3     3     4     3     4     4     1     1     4     4
 [26,]    2    5    2    2    0    2    4    1    4     2     4     4     4     4     1     2     4     3
```

```
[27,]  3  5  1  2  0  2  3  1  5  1  4  5  4  4  1  3  4  2
[28,]  4  5  0  2  0  2  2  1  6  0  4  6  4  4  1  4  4  1
[29,]  1  5  3  1  0  3  6  1  2  3  4  3  5  4  0  0  4  5
[30,]  2  5  2  1  0  3  5  1  3  2  4  4  5  4  0  1  4  4
[31,]  3  5  1  1  0  3  4  1  4  1  4  5  5  4  0  2  4  3
[32,]  4  5  0  1  0  3  3  1  5  0  4  6  5  4  0  3  4  2
[33,]  0  4  5  4  0  0  4  2  3  4  5  1  2  4  3  2  3  4
[34,]  1  4  4  4  0  0  3  2  4  3  5  2  2  4  3  3  3  3
[35,]  2  4  3  4  0  0  2  2  5  2  5  3  2  4  3  4  3  2
[36,]  3  4  2  4  0  0  1  2  6  1  5  4  2  4  3  5  3  1
[37,]  4  4  1  4  0  0  0  2  7  0  5  5  2  4  3  6  3  0
[38,]  0  4  5  3  0  1  5  2  2  4  5  1  3  4  2  1  3  5
[39,]  1  4  4  3  0  1  4  2  3  3  5  2  3  4  2  2  3  4
[40,]  2  4  3  3  0  1  3  2  4  2  5  3  3  4  2  3  3  3
[41,]  3  4  2  3  0  1  2  2  5  1  5  4  3  4  2  4  3  2
[42,]  4  4  1  3  0  1  1  2  6  0  5  5  3  4  2  5  3  1
[43,]  0  4  5  2  0  2  6  2  1  4  5  1  4  4  1  0  3  6
[44,]  1  4  4  2  0  2  5  2  2  3  5  2  4  4  1  1  3  5
[45,]  2  4  3  2  0  2  4  2  3  2  5  3  4  4  1  2  3  4
[46,]  3  4  2  2  0  2  3  2  4  1  5  4  4  4  1  3  3  3
[47,]  4  4  1  2  0  2  2  2  5  0  5  5  4  4  1  4  3  2
[48,]  1  4  4  1  0  3  6  2  1  3  5  2  5  4  0  0  3  6
[49,]  2  4  3  1  0  3  5  2  2  2  5  3  5  4  0  1  3  5
[50,]  3  4  2  1  0  3  4  2  3  1  5  4  5  4  0  2  3  4
[51,]  4  4  1  1  0  3  3  2  4  0  5  5  5  4  0  3  3  3
[52,]  0  3  6  4  0  0  4  3  2  4  6  0  2  4  3  2  2  5
[53,]  1  3  5  4  0  0  3  3  3  3  6  1  2  4  3  3  2  4
[54,]  2  3  4  4  0  0  2  3  4  2  6  2  2  4  3  4  2  3
[55,]  3  3  3  4  0  0  1  3  5  1  6  3  2  4  3  5  2  2
[56,]  4  3  2  4  0  0  0  3  6  0  6  4  2  4  3  6  2  1
[57,]  0  3  6  3  0  1  5  3  1  4  6  0  3  4  2  1  2  6
[58,]  1  3  5  3  0  1  4  3  2  3  6  1  3  4  2  2  2  5
[59,]  2  3  4  3  0  1  3  3  3  2  6  2  3  4  2  3  2  4
[60,]  3  3  3  3  0  1  2  3  4  1  6  3  3  4  2  4  2  3
[61,]  4  3  2  3  0  1  1  3  5  0  6  4  3  4  2  5  2  2
[62,]  0  3  6  2  0  2  6  3  0  4  6  0  4  4  1  0  2  7
[63,]  1  3  5  2  0  2  5  3  1  3  6  1  4  4  1  1  2  6
[64,]  2  3  4  2  0  2  4  3  2  2  6  2  4  4  1  2  2  5
[65,]  3  3  3  2  0  2  3  3  3  1  6  3  4  4  1  3  2  4
[66,]  4  3  2  2  0  2  2  3  4  0  6  4  4  4  1  4  2  3
[67,]  1  3  5  1  0  3  6  3  0  3  6  1  5  4  0  0  2  7
[68,]  2  3  4  1  0  3  5  3  1  2  6  2  5  4  0  1  2  6
[69,]  3  3  3  1  0  3  4  3  2  1  6  3  5  4  0  2  2  5
[70,]  4  3  2  1  0  3  3  3  3  0  6  4  5  4  0  3  2  4
[71,]  1  2  6  4  0  0  3  4  2  3  7  0  2  4  3  3  1  5
[72,]  2  2  5  4  0  0  2  4  3  2  7  1  2  4  3  4  1  4
[73,]  3  2  4  4  0  0  1  4  4  1  7  2  2  4  3  5  1  3
[74,]  4  2  3  4  0  0  0  4  5  0  7  3  2  4  3  6  1  2
[75,]  1  2  6  3  0  1  4  4  1  3  7  0  3  4  2  2  1  6
[76,]  2  2  5  3  0  1  3  4  2  2  7  1  3  4  2  3  1  5
[77,]  3  2  4  3  0  1  2  4  3  1  7  2  3  4  2  4  1  4
[78,]  4  2  3  3  0  1  1  4  4  0  7  3  3  4  2  5  1  3
[79,]  1  2  6  2  0  2  5  4  0  3  7  0  4  4  1  1  1  7
[80,]  2  2  5  2  0  2  4  4  1  2  7  1  4  4  1  2  1  6
```

```
 [81,]    3    2    4    2    0    2    3    4    2    1    7    2    4    4    1    3    1    5
 [82,]    4    2    3    2    0    2    2    4    3    0    7    3    4    4    1    4    1    4
 [83,]    2    2    5    1    0    3    5    4    0    2    7    1    5    4    0    1    1    7
 [84,]    3    2    4    1    0    3    4    4    1    1    7    2    5    4    0    2    1    6
 [85,]    4    2    3    1    0    3    3    4    2    0    7    3    5    4    0    3    1    5
 [86,]    2    1    6    4    0    0    2    5    2    2    8    0    2    4    3    4    0    5
 [87,]    3    1    5    4    0    0    1    5    3    1    8    1    2    4    3    5    0    4
 [88,]    4    1    4    4    0    0    0    5    4    0    8    2    2    4    3    6    0    3
 [89,]    2    1    6    3    0    1    3    5    1    2    8    0    3    4    2    3    0    6
 [90,]    3    1    5    3    0    1    2    5    2    1    8    1    3    4    2    4    0    5
 [91,]    4    1    4    3    0    1    1    5    3    0    8    2    3    4    2    5    0    4
 [92,]    2    1    6    2    0    2    4    5    0    2    8    0    4    4    1    2    0    7
 [93,]    3    1    5    2    0    2    3    5    1    1    8    1    4    4    1    3    0    6
 [94,]    4    1    4    2    0    2    2    5    2    0    8    2    4    4    1    4    0    5
 [95,]    3    1    5    1    0    3    4    5    0    1    8    1    5    4    0    2    0    7
 [96,]    4    1    4    1    0    3    3    5    1    0    8    2    5    4    0    3    0    6
 [97,]    0    5    4    3    1    0    5    0    4    4    4    2    3    3    3    1    5    3
 [98,]    1    5    3    3    1    0    4    0    5    3    4    3    3    3    3    2    5    2
 [99,]    2    5    2    3    1    0    3    0    6    2    4    4    3    3    3    3    5    1
[100,]    3    5    1    3    1    0    2    0    7    1    4    5    3    3    3    4    5    0
[101,]    0    5    4    2    1    1    6    0    3    4    4    2    4    3    2    0    5    4
[102,]    1    5    3    2    1    1    5    0    4    3    4    3    4    3    2    1    5    3
[103,]    2    5    2    2    1    1    4    0    5    2    4    4    4    3    2    2    5    2
[104,]    3    5    1    2    1    1    3    0    6    1    4    5    4    3    2    3    5    1
[105,]    4    5    0    2    1    1    2    0    7    0    4    6    4    3    2    4    5    0
[106,]    1    5    3    1    1    2    6    0    3    3    4    3    5    3    1    0    5    4
[107,]    2    5    2    1    1    2    5    0    4    2    4    4    5    3    1    1    5    3
[108,]    3    5    1    1    1    2    4    0    5    1    4    5    5    3    1    2    5    2
[109,]    4    5    0    1    1    2    3    0    6    0    4    6    5    3    1    3    5    1
[110,]    2    5    2    0    1    3    6    0    3    2    4    4    6    3    0    0    5    4
[111,]    3    5    1    0    1    3    5    0    4    1    4    5    6    3    0    1    5    3
[112,]    4    5    0    0    1    3    4    0    5    0    4    6    6    3    0    2    5    2
[113,]    0    4    5    3    1    0    5    1    3    4    5    1    3    3    3    1    4    4
[114,]    1    4    4    3    1    0    4    1    4    3    5    2    3    3    3    2    4    3
[115,]    2    4    3    3    1    0    3    1    5    2    5    3    3    3    3    3    4    2
[116,]    3    4    2    3    1    0    2    1    6    1    5    4    3    3    3    4    4    1
[117,]    4    4    1    3    1    0    1    1    7    0    5    5    3    3    3    5    4    0
[118,]    0    4    5    2    1    1    6    1    2    4    5    1    4    3    2    0    4    5
[119,]    1    4    4    2    1    1    5    1    3    3    5    2    4    3    2    1    4    4
[120,]    2    4    3    2    1    1    4    1    4    2    5    3    4    3    2    2    4    3
[121,]    3    4    2    2    1    1    3    1    5    1    5    4    4    3    2    3    4    2
[122,]    4    4    1    2    1    1    2    1    6    0    5    5    4    3    2    4    4    1
[123,]    1    4    4    1    1    2    6    1    2    3    5    2    5    3    1    0    4    5
[124,]    2    4    3    1    1    2    5    1    3    2    5    3    5    3    1    1    4    4
[125,]    3    4    2    1    1    2    4    1    4    1    5    4    5    3    1    2    4    3
[126,]    4    4    1    1    1    2    3    1    5    0    5    5    5    3    1    3    4    2
[127,]    2    4    3    0    1    3    6    1    2    2    5    3    6    3    0    0    4    5
[128,]    3    4    2    0    1    3    5    1    3    1    5    4    6    3    0    1    4    4
[129,]    4    4    1    0    1    3    4    1    4    0    5    5    6    3    0    2    4    3
[130,]    0    3    6    3    1    0    5    2    2    4    6    0    3    3    3    1    3    5
[131,]    1    3    5    3    1    0    4    2    3    3    6    1    3    3    3    2    3    4
[132,]    2    3    4    3    1    0    3    2    4    2    6    2    3    3    3    3    3    3
[133,]    3    3    3    3    1    0    2    2    5    1    6    3    3    3    3    4    3    2
[134,]    4    3    2    3    1    0    1    2    6    0    6    4    3    3    3    5    3    1
```

```
[135,]  0  3  6  2  1  1  6  2  1  4  6  0  4  3  2  0  3  6
[136,]  1  3  5  2  1  1  5  2  2  3  6  1  4  3  2  1  3  5
[137,]  2  3  4  2  1  1  4  2  3  2  6  2  4  3  2  2  3  4
[138,]  3  3  3  2  1  1  3  2  4  1  6  3  4  3  2  3  3  3
[139,]  4  3  2  2  1  1  2  2  5  0  6  4  4  3  2  4  3  2
[140,]  1  3  5  1  1  2  6  2  1  3  6  1  5  3  1  0  3  6
[141,]  2  3  4  1  1  2  5  2  2  2  6  2  5  3  1  1  3  5
[142,]  3  3  3  1  1  2  4  2  3  1  6  3  5  3  1  2  3  4
[143,]  4  3  2  1  1  2  3  2  4  0  6  4  5  3  1  3  3  3
[144,]  2  3  4  0  1  3  6  2  1  2  6  2  6  3  0  0  3  6
[145,]  3  3  3  0  1  3  5  2  2  1  6  3  6  3  0  1  3  5
[146,]  4  3  2  0  1  3  4  2  3  0  6  4  6  3  0  2  3  4
[147,]  1  2  6  3  1  0  4  3  2  3  7  0  3  3  3  2  2  5
[148,]  2  2  5  3  1  0  3  3  3  2  7  1  3  3  3  3  2  4
[149,]  3  2  4  3  1  0  2  3  4  1  7  2  3  3  3  4  2  3
[150,]  4  2  3  3  1  0  1  3  5  0  7  3  3  3  3  5  2  2
[151,]  1  2  6  2  1  1  5  3  1  3  7  0  4  3  2  1  2  6
[152,]  2  2  5  2  1  1  4  3  2  2  7  1  4  3  2  2  2  5
[153,]  3  2  4  2  1  1  3  3  3  1  7  2  4  3  2  3  2  4
[154,]  4  2  3  2  1  1  2  3  4  0  7  3  4  3  2  4  2  3
[155,]  1  2  6  1  1  2  6  3  0  3  7  0  5  3  1  0  2  7
[156,]  2  2  5  1  1  2  5  3  1  2  7  1  5  3  1  1  2  6
[157,]  3  2  4  1  1  2  4  3  2  1  7  2  5  3  1  2  2  5
[158,]  4  2  3  1  1  2  3  3  3  0  7  3  5  3  1  3  2  4
[159,]  2  2  5  0  1  3  6  3  0  2  7  1  6  3  0  0  2  7
[160,]  3  2  4  0  1  3  5  3  1  1  7  2  6  3  0  1  2  6
[161,]  4  2  3  0  1  3  4  3  2  0  7  3  6  3  0  2  2  5
[162,]  2  1  6  3  1  0  3  4  2  2  8  0  3  3  3  3  1  5
[163,]  3  1  5  3  1  0  2  4  3  1  8  1  3  3  3  4  1  4
[164,]  4  1  4  3  1  0  1  4  4  0  8  2  3  3  3  5  1  3
[165,]  2  1  6  2  1  1  4  4  1  2  8  0  4  3  2  2  1  6
[166,]  3  1  5  2  1  1  3  4  2  1  8  1  4  3  2  3  1  5
[167,]  4  1  4  2  1  1  2  4  3  0  8  2  4  3  2  4  1  4
[168,]  2  1  6  1  1  2  5  4  0  2  8  0  5  3  1  1  1  7
[169,]  3  1  5  1  1  2  4  4  1  1  8  1  5  3  1  2  1  6
[170,]  4  1  4  1  1  2  3  4  2  0  8  2  5  3  1  3  1  5
[171,]  3  1  5  0  1  3  5  4  0  1  8  1  6  3  0  1  1  7
[172,]  4  1  4  0  1  3  4  4  1  0  8  2  6  3  0  2  1  6
[173,]  3  0  6  3  1  0  2  5  2  1  9  0  3  3  3  4  0  5
[174,]  4  0  5  3  1  0  1  5  3  0  9  1  3  3  3  5  0  4
[175,]  3  0  6  2  1  1  3  5  1  1  9  0  4  3  2  3  0  6
[176,]  4  0  5  2  1  1  2  5  2  0  9  1  4  3  2  4  0  5
[177,]  3  0  6  1  1  2  4  5  0  1  9  0  5  3  1  2  0  7
[178,]  4  0  5  1  1  2  3  5  1  0  9  1  5  3  1  3  0  6
[179,]  4  0  5  0  1  3  4  5  0  0  9  1  6  3  0  2  0  7
[180,]  0  4  5  2  2  0  6  0  3  4  5  1  4  2  3  0  5  4
[181,]  1  4  4  2  2  0  5  0  4  3  5  2  4  2  3  1  5  3
[182,]  2  4  3  2  2  0  4  0  5  2  5  3  4  2  3  2  5  2
[183,]  3  4  2  2  2  0  3  0  6  1  5  4  4  2  3  3  5  1
[184,]  4  4  1  2  2  0  2  0  7  0  5  5  4  2  3  4  5  0
[185,]  1  4  4  1  2  1  6  0  3  3  5  2  5  2  2  0  5  4
[186,]  2  4  3  1  2  1  5  0  4  2  5  3  5  2  2  1  5  3
[187,]  3  4  2  1  2  1  4  0  5  1  5  4  5  2  2  2  5  2
[188,]  4  4  1  1  2  1  3  0  6  0  5  5  5  2  2  3  5  1
```

```
[189,]  2  4  3  0  2  2  6  0  3  2  5  3  6  2  1  0  5  4
[190,]  3  4  2  0  2  2  5  0  4  1  5  4  6  2  1  1  5  3
[191,]  4  4  1  0  2  2  4  0  5  0  5  5  6  2  1  2  5  2
[192,]  0  3  6  2  2  0  6  1  2  4  6  0  4  2  3  0  4  5
[193,]  1  3  5  2  2  0  5  1  3  3  6  1  4  2  3  1  4  4
[194,]  2  3  4  2  2  0  4  1  4  2  6  2  4  2  3  2  4  3
[195,]  3  3  3  2  2  0  3  1  5  1  6  3  4  2  3  3  4  2
[196,]  4  3  2  2  2  0  2  1  6  0  6  4  4  2  3  4  4  1
[197,]  1  3  5  1  2  1  6  1  2  3  6  1  5  2  2  0  4  5
[198,]  2  3  4  1  2  1  5  1  3  2  6  2  5  2  2  1  4  4
[199,]  3  3  3  1  2  1  4  1  4  1  6  3  5  2  2  2  4  3
[200,]  4  3  2  1  2  1  3  1  5  0  6  4  5  2  2  3  4  2
[201,]  2  3  4  0  2  2  6  1  2  2  6  2  6  2  1  0  4  5
[202,]  3  3  3  0  2  2  5  1  3  1  6  3  6  2  1  1  4  4
[203,]  4  3  2  0  2  2  4  1  4  0  6  4  6  2  1  2  4  3
[204,]  1  2  6  2  2  0  5  2  2  3  7  0  4  2  3  1  3  5
[205,]  2  2  5  2  2  0  4  2  3  2  7  1  4  2  3  2  3  4
[206,]  3  2  4  2  2  0  3  2  4  1  7  2  4  2  3  3  3  3
[207,]  4  2  3  2  2  0  2  2  5  0  7  3  4  2  3  4  3  2
[208,]  1  2  6  1  2  1  6  2  1  3  7  0  5  2  2  0  3  6
[209,]  2  2  5  1  2  1  5  2  2  2  7  1  5  2  2  1  3  5
[210,]  3  2  4  1  2  1  4  2  3  1  7  2  5  2  2  2  3  4
[211,]  4  2  3  1  2  1  3  2  4  0  7  3  5  2  2  3  3  3
[212,]  2  2  5  0  2  2  6  2  1  2  7  1  6  2  1  0  3  6
[213,]  3  2  4  0  2  2  5  2  2  1  7  2  6  2  1  1  3  5
[214,]  4  2  3  0  2  2  4  2  3  0  7  3  6  2  1  2  3  4
[215,]  2  1  6  2  2  0  4  3  2  2  8  0  4  2  3  2  2  5
[216,]  3  1  5  2  2  0  3  3  3  1  8  1  4  2  3  3  2  4
[217,]  4  1  4  2  2  0  2  3  4  0  8  2  4  2  3  4  2  3
[218,]  2  1  6  1  2  1  5  3  1  2  8  0  5  2  2  1  2  6
[219,]  3  1  5  1  2  1  4  3  2  1  8  1  5  2  2  2  2  5
[220,]  4  1  4  1  2  1  3  3  3  0  8  2  5  2  2  3  2  4
[221,]  2  1  6  0  2  2  6  3  0  2  8  0  6  2  1  0  2  7
[222,]  3  1  5  0  2  2  5  3  1  1  8  1  6  2  1  1  2  6
[223,]  4  1  4  0  2  2  4  3  2  0  8  2  6  2  1  2  2  5
[224,]  3  0  6  2  2  0  3  4  2  1  9  0  4  2  3  3  1  5
[225,]  4  0  5  2  2  0  2  4  3  0  9  1  4  2  3  4  1  4
[226,]  3  0  6  1  2  1  4  4  1  1  9  0  5  2  2  2  1  6
[227,]  4  0  5  1  2  1  3  4  2  0  9  1  5  2  2  3  1  5
[228,]  3  0  6  0  2  2  5  4  0  1  9  0  6  2  1  1  1  7
[229,]  4  0  5  0  2  2  4  4  1  0  9  1  6  2  1  2  1  6
[230,]  1  3  5  1  3  0  6  0  3  3  6  1  5  1  3  0  5  4
[231,]  2  3  4  1  3  0  5  0  4  2  6  2  5  1  3  1  5  3
[232,]  3  3  3  1  3  0  4  0  5  1  6  3  5  1  3  2  5  2
[233,]  4  3  2  1  3  0  3  0  6  0  6  4  5  1  3  3  5  1
[234,]  2  3  4  0  3  1  6  0  3  2  6  2  6  1  2  0  5  4
[235,]  3  3  3  0  3  1  5  0  4  1  6  3  6  1  2  1  5  3
[236,]  4  3  2  0  3  1  4  0  5  0  6  4  6  1  2  2  5  2
[237,]  1  2  6  1  3  0  6  1  2  3  7  0  5  1  3  0  4  5
[238,]  2  2  5  1  3  0  5  1  3  2  7  1  5  1  3  1  4  4
[239,]  3  2  4  1  3  0  4  1  4  1  7  2  5  1  3  2  4  3
[240,]  4  2  3  1  3  0  3  1  5  0  7  3  5  1  3  3  4  2
[241,]  2  2  5  0  3  1  6  1  2  2  7  1  6  1  2  0  4  5
[242,]  3  2  4  0  3  1  5  1  3  1  7  2  6  1  2  1  4  4
```

```
[243,]    4    2    3    0    3    1    4    1    4    0    7    3    6    1    2    2    4    3
[244,]    2    1    6    1    3    0    5    2    2    2    8    0    5    1    3    1    3    5
[245,]    3    1    5    1    3    0    4    2    3    1    8    1    5    1    3    2    3    4
[246,]    4    1    4    1    3    0    3    2    4    0    8    2    5    1    3    3    3    3
[247,]    2    1    6    0    3    1    6    2    1    2    8    0    6    1    2    0    3    6
[248,]    3    1    5    0    3    1    5    2    2    1    8    1    6    1    2    1    3    5
[249,]    4    1    4    0    3    1    4    2    3    0    8    2    6    1    2    2    3    4
[250,]    3    0    6    1    3    0    4    3    2    1    9    0    5    1    3    2    2    5
[251,]    4    0    5    1    3    0    3    3    3    0    9    1    5    1    3    3    2    4
[252,]    3    0    6    0    3    1    5    3    1    1    9    0    6    1    2    1    2    6
[253,]    4    0    5    0    3    1    4    3    2    0    9    1    6    1    2    2    2    5
[254,]    2    2    5    0    4    0    6    0    3    2    7    1    6    0    3    0    5    4
[255,]    3    2    4    0    4    0    5    0    4    1    7    2    6    0    3    1    5    3
[256,]    4    2    3    0    4    0    4    0    5    0    7    3    6    0    3    2    5    2
[257,]    2    1    6    0    4    0    6    1    2    2    8    0    6    0    3    0    4    5
[258,]    3    1    5    0    4    0    5    1    3    1    8    1    6    0    3    1    4    4
[259,]    4    1    4    0    4    0    4    1    4    0    8    2    6    0    3    2    4    3
[260,]    3    0    6    0    4    0    5    2    2    1    9    0    6    0    3    1    3    5
[261,]    4    0    5    0    4    0    4    2    3    0    9    1    6    0    3    2    3    4
> sum(ptab[ptab<=ptab[loc]])
[1] 0.9190594
```

A.3 Algorithms for 3 × 3 × K tables

```
#-------------------------------------------------------------------------------
#------------------------------- A function to generate general discrete values
#-------------------------------

rdisc =  function(n=1, prop=NULL, levels=NULL){
    if (is.null(prop)){
        print("probability vector should be specified")
        stop
    }
    K =  length(prop)
    if (is.null(levels)) levels=1:K
    cum.prob=  cumsum(prop)
    foo =  rep(0, n)
    for (i in 1:n){
        u =  runif(1) ;as.numeric(u <= cum.prob)
        test = levels[match(1,as.numeric(u <= cum.prob))]
        foo[i] =  ifelse(is.na(test),K,test)
    }
    return(foo)
  }

#------------------ A function to compute c(n); number of bits ----------------

bits = function(n){
       m  = ceiling(log(n,2))
       cn = 2^m*(m/n)
```

```
       return(cn)
      }

#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#----------------------- Function to generate move of degree 4 for 3x3x4 tables

rand.m4 <- function(x0){
    dims = dim(x0)
    m = array(0, dim=dims)
    i =  sample(1:dims[1],2)
    j =  sample(1:dims[2],2)
    k =  sample(1:dims[3],2)

    m[i[1],j[1],k[1]] =  m[i[2],j[2],k[1]] = +1
    m[i[2],j[1],k[1]] =  m[i[1],j[2],k[1]] = -1
    m[,,k[2]] = -m[,,k[1]]

    n.moves = choose(dims[1],2)*choose(dims[2],2)*choose(dims[3],2)
    assign("CN",    CN+bits(n.moves),  envir=.GlobalEnv)
    return(m)
    }

#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#----------------------- Function to generate move of degree 6 for 3x3x4 tables
rand.m6 <- function(x0) {
    dims=dim(x0)
    m = array(0, dim=dims)
    Type=floor(runif(1, min=1, max=4))

    if (Type==1){
        i =  sample(1:dims[1],3)
        j =  sample(1:dims[2],3)
        k =  sample(1:dims[3],2)

        m[i[1],j[1],k[1]] =  m[i[2],j[2],k[1]] =  m[i[3],j[3],k[1]] = +1
        m[i[1],j[2],k[1]] =  m[i[2],j[3],k[1]] =  m[i[3],j[1],k[1]] = -1
        m[,,k[2]] = -m[,,k[1]]
        n.moves = choose(dims[1],3)*choose(dims[2],3)*choose(dims[3],2)
    }

    if (Type==2){
        i =  sample(1:dims[1],2)
        j =  sample(1:dims[2],3)
        k =  sample(1:dims[3],3)

        m[i[1],j[1],k[1]] =  m[i[1],j[2],k[2]] =  m[i[1],j[3],k[3]] = +1
        m[i[1],j[1],k[2]] =  m[i[1],j[2],k[3]] =  m[i[1],j[3],k[1]] = -1
        m[i[2],,] = -m[i[1],,]
        n.moves = choose(dims[1],2)*choose(dims[2],3)*choose(dims[3],3)
    }

    if (Type==3){
```

```
        i =  sample(1:dims[1],3)
        j =  sample(1:dims[2],2)
        k =  sample(1:dims[3],3)

        m[i[1],j[1],k[1]] =  m[i[2],j[1],k[2]] =  m[i[3],j[1],k[3]] = +1
        m[i[1],j[1],k[2]] =  m[i[2],j[1],k[3]] =  m[i[3],j[1],k[1]] = -1
        m[,j[2],] = -m[,j[1],]
        n.moves = choose(dims[1],3)*choose(dims[2],2)*choose(dims[3],3)
    }

    assign("CN",    CN+bits(n.moves),  envir=.GlobalEnv)
    return(m)
}

#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#------------------------ Function to generate move 0f degree 8 for 3x3x4 tables
rand.m8 <- function(x0){
    dims=dim(x0)
    m = array(0, dim=dims)
    i =  sample(1:dims[3],3)
    j =  sample(1:dims[1],3)
    k =  sample(1:dims[2],4)

    m[j[1],k[1],i[1]] = m[j[2],k[2],i[1]] = m[j[1],k[3],i[2]] = m[j[2],k[1],i[2]] = +1
    m[j[3],k[4],i[2]] = m[j[1],k[2],i[3]] = m[j[2],k[4],i[3]] = m[j[3],k[3],i[3]] = +1

    m[j[1],k[2],i[1]] = m[j[2],k[1],i[1]] = m[j[1],k[1],i[2]] = m[j[2],k[4],i[2]] = -1
    m[j[3],k[3],i[2]] = m[j[1],k[3],i[3]] = m[j[2],k[2],i[3]] = m[j[3],k[4],i[3]] = -1

    n.moves = choose(dims[3],3)*choose(dims[1],3)*choose(dims[2],4)
    assign("CN",    CN+bits(n.moves),  envir=.GlobalEnv)
    return(m)
    }

#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#------------------------ Function to generate move 0f degree 10 for 3x3x4 tables
rand.m10 <- function(x0){
    dims=dim(x0)
    m = array(0, dim=dims)
    i =  sample(1:dims[3],3)
    j =  sample(1:dims[1],3)
    k =  sample(1:dims[2],5)

    m[j[1],k[1],i[1]] = m[j[2],k[2],i[1]] = m[j[2],k[5],i[1]] = m[j[3],k[4],i[1]] = +1
    m[j[1],k[3],i[2]] = m[j[2],k[1],i[2]] = m[j[3],k[5],i[2]] = m[j[1],k[2],i[3]] = +1
    m[j[2],k[4],i[3]] = m[j[3],k[3],i[3]] = +1

    m[j[1],k[2],i[1]] = m[j[2],k[1],i[1]] = m[j[2],k[4],i[1]] = m[j[3],k[5],i[1]] = -1
    m[j[1],k[1],i[2]] = m[j[2],k[5],i[2]] = m[j[3],k[3],i[2]] = m[j[1],k[3],i[3]] = -1
    m[j[2],k[2],i[3]] = m[j[3],k[4],i[3]] = -1

    n.moves = choose(dims[3],3)*choose(dims[1],3)*choose(dims[2],4)
```

```
    assign("CN",    CN+bits(n.moves),  envir=.GlobalEnv)
    return(m)
}

#-------------------------------------------------------------------------
#-------------------------------------------------------------------------
#---------- Function to generate move from M_star for a table of 3x3x4 dimension
rand.ms = function(x0, prop){
    dims = dim(x0)
    M = max(dims)
    Mi=2*rdisc(1, prop, levels=2:M)

    if (Mi==4)  m = rand.m4(x0)
    if (Mi==6)  m = rand.m6(x0)
    if (Mi==8)  m = rand.m8(x0)
    if (Mi==10) m = rand.m10(x0)
    assign("CN.rbern",CN.rbern+1,    envir=.GlobalEnv)
    return(m)
}

##------------- A simpler function to check acceptance of random sample --------

accept = function(pi){
  if(pi >= 1) return(TRUE)
  assign('CN.rbern', CN.rbern+1, envir=.GlobalEnv)
  return(runif(1) <= pi)
}

##-------------- Function for MCMC from move from M* based on Aoki-Takemura

mcmc.at = function(x0,n, prop){
    xold   <-  x0
    xl     <- matrix(0,ncol=n,  nrow=length(x0))
    for(t in 1:n){
        if (t%%10000==0) cat(t, "\n")
        m     <- rand.ms(x0, prop) ## produce the m from M*
        x     <- xold
        xl[,t]<- x
        xs <- x+m   ## producing subsequent tables by random move
        if (any(xs<0)){
            x   <- xold
        }
        else{
             pi <- min(1, exp(sum(lfactorial(xold)-lfactorial(xs))))
             if (accept(pi)) xold <- xs
        }
    }
    return(xl)
}

#------------------------------------------- Aoki- takemura method for 3x3xK

eff.at = function(x0, n=10000, prop){
    library(splines)
```

```
    library(lattice)
    library(coda)

    assign("CN.rbern",   0 , envir=.GlobalEnv)
    assign("CN" ,      0 , envir=.GlobalEnv)
   output = mcmc.at(x0, n, prop)
   T      = matrix(output, ncol=n)
   likl   = function(x){1/exp(sum(lfactorial(x))/sum(x0))}
   P      = apply(T, MARGIN=2, FUN= likl)
   p0     = likl(as.vector(x0))
   logit  = log(P)
   I      = as.numeric(P <= p0)
   Ibar   = mean(I)
   ne.I   = effectiveSize(I)
   ne.logit = effectiveSize(logit)

   cost.low = CN + 1* CN.rbern
   cost.upp = CN + 2* CN.rbern
    se.Ibar  = sqrt(Ibar*(1-Ibar)/ne.I)
    n.cn     = CN
    eff.logit.upp = ne.logit/cost.low
    eff.logit.low = ne.logit/cost.upp

    return(list(Ibar          = Ibar,
                se.Ibar       = se.Ibar,
                ne.logit      = ne.logit,
                cost.low      = cost.low,
                cost.upp      = cost.upp,
                n.cn          = n.cn,
                n.bern        = CN.rbern,
                eff.logit.low = eff.logit.low,
                eff.logit.upp = eff.logit.upp))
}
```