NTNU – Trondheim
Norwegian University of
Science and Technology

A Bayesian model for the dependence structure in binary Markov random fields.

Cecilie Drabløs Frøysa

Master of Science in Physics and Mathematics
Submission date: June 2014
Supervisor: Håkon Tjelmeland, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Description

The candidate should formulate the problem of making inference for binary Markov random fields in a Bayesian setting. In particular she should consider how to make inference on the neighbourhood system and dependence structure of the Markov random field. She should develop and implement necessary simulation algorithms to explore the defined model empirically.

Preface

This thesis with course code TMA4905 Statistics completes my Master's degree in Physics and Mathematics at the Norwegian University of Science and Technology (NTNU), Department of Mathematical Sciences. The work was carried out in the spring of 2014.

I would like to thank my supervisor Håkon Tjelmeland for excellent guidance and assistance throughout this semester. His feedback has been extremely helpful.

Trondheim, June 2014

Cecilie Drabløs Frøysa

Abstract

In this thesis a reversible jump Markov chain Monte Carlo (MCMC) method for simulation of the graph structure of a binary Markov random field (MRF) is presented. The reversible jump MCMC method allows for simulation of both the graph structure and the parameter values of the MRF. First a Bayesian model for the problem is described. The prior model used is a slightly altered version of the spike and slab prior used by Chen and Welling (2012). Next the algorithm for simulation is presented and the method is then tested for simulated datasets of different sized based on two example graphs. The algorithm is able to find models that give good fits to most of the datasets, but we see signs of the algorithm not converging properly.

Sammendrag

I denne masteroppgaven presenteres en metode for Markovkjedesimulering (MCMC) for grafstrukturen til et binært Markovfelt. Metoden som presenteres er en metode som kan simulere modeller med varierende dimensjoner som lar oss simulere både grafstrukturen og parameterverdiene i Markovfeltet. Vi beskriver først en Bayesiansk modell for problemet. Apriorimodellen som brukes er en litt endret versjon av "the spike and slab prior" som presenteres av Chen og Welling (2012). Deretter presenteres simuleringsalgoritmen for modellen og til slutt testes algoritmen for simulerte datasett basert på to forskjellige eksempelgrafer. Algoritmen er i stand til å finne modeller som passer godt til de fleste datasettene, men vi ser tegn på at algoritmen ikke konvergerer som den skal.

Contents

1 Introduction

Markov random fields (Kindermann and Snell, 1980), also known as undirected probabilistic graphical models or Markov networks, consist of a set of variables with a distribution having a Markov property. The Markov property of the distribution is given by an undirected graph. Undirected graphical models are useful in modelling phenomena where the direction of the interaction between variables cannot be naturally ascribed, but direct computation of these models can be time consuming or impossible. Exact computation for Markov random fields is often limited by the normalizing constant that in many cases will become more and more time consuming to compute as the number of variables increase. Different Bayesian models and simulation methods for Markov random fields are presented by Murray and Ghahramani (2004), Parise and Welling (2006), Jones et al. (2005), Chen and Welling (2012) and several others.

We look at the Markov random field in a Bayesian setting using and the spike and slab prior distribution used by Chen and Welling (2012), and develop a Markov chain Monte Carlo method for structure learning and parameter estimation in a binary Markov random field. The method developed is a reversible jump Markov chain Monte Carlo method (Green, 1995). Reversible jump Markov chain Monte Carlo methods can be seen as a generalization of Metropolis-Hastings algorithms (Gamerman and Lopes, 2006) and a tutorial derivation of a reversible jump Markov chain Monte Carlo method is given by Waagepetersen and Sorensen (2012). The algorithm is tested on simulated datasets from two different underlying distributions. Testing the algorithm for simulated datasets where the underlying distribution makes it possible to see if the results of simulations give a good fit to the underlying model and the different datasets.

In Section 2 theory about graphs and MRFs is briefly presented. Section 3 contains a description of the spike and slab prior used by Chen and Welling (2012). Section 4 gives a brief introduction to Markov chain Monte Carlo methods. In Section 5 the Bayesian model for our particular problem is presented while Section 6 presents a Markov chain Monte Carlo method for simulation of the posterior distribution presented in Section 5. In Section 7 results from simulations are presented, and the last section gives closing remarks.

2 Markov Random Fields

This section contains a brief introduction to the most important concepts concerning graphs and Markov random fields. A Markov random field can be viewed as a generalization of a Markov chain process and is generally used to represent dependencies between a set of stochastic variables. More information about Markov random fields can be found in Kindermann and Snell (1980) and Koller and Friedman (2009).

2.1 Graphs, Neighbourhoods and Cliques

An undirected graph is a data structure consisting of a set of vertices and a set of edges. We now define such an undirected graph $G = \{V, E\}$ consisting of the vertex set $V = \{1, ..., n\}$ and edge set $E = \{e_1, ...e_m\}$. Here n and m are the number of vertices and edges, respectively. In the graph G we can have vertices with edges connecting it to one or more other vertices, or vertices with no edges connected to it. Each edge $e_i \in E$ connects two distinct vertices, which means that $e_i \in \{(p, q)|p, q \in V, p \neq q\}$. The undirected graph can be visualized as in Figure 1. For this particular example graph we see that $n = 14$ and $m = 10$. If two different vertices $p, q \in V$, are connected by an edge $(p, q) \in E$, we say that p and q are neighbours and the set of all vertices that are neighbours of a particular vertex p is called the neighbourhood of p. The neighbourhood of p is denoted by ∂p and for a vertex with no edges connected to it, this is the empty set $\emptyset$. Because each element of the edge set must consist of two distinct vertices we see that $p \notin \partial p \ \forall p \in V$. This also means that no vertex can be a member of its own neighbourhood. It follows that $p \in \partial q \Leftrightarrow q \in \partial p$, which means that if p is in the neighbourhood of q, then q is also in the neighbourhood of p. The maximal possible number of edges in an undirected graph is $M = n(n-1)/2$ and we let the set of all possible edges in a graph be denoted by $E_{full} = \{(p, q)|p \neq q, \forall p, q \in V\}$. Some examples of neighbourhoods in the example graph in Figure 1 are $\partial 1 = \emptyset$, $\partial 2 = \{3, 6\}$ and $\partial 5 = \{9\}$. In the example graph no vertices have more than three neighbours and the maximal number of edges that could be members of the edge set is 91.

A clique, C, is a subset of the vertex set, V, where every two vertices in C are connected by an edge. This means that every vertex in the clique is in the neighbourhoods of all other vertices in the clique, or in other words that any two vertices $p, q \in C, p \neq q$, are neighbours. We see that a clique is a subset containing either no vertices, a single vertex, or a bigger group of vertices. The number of elements

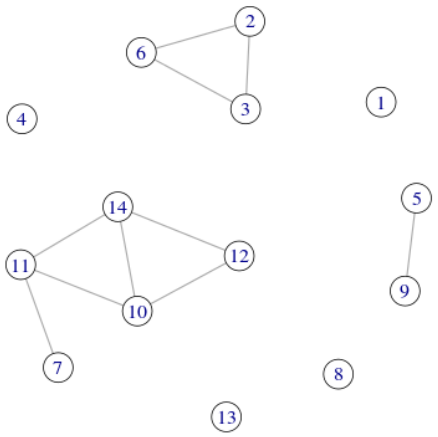

Figure 1: An undirected graph $G = \{V, E\}$ with $V = \{1, ..., 14\}$ and $E = \{(2,3), (2,6), (3,6), (5,9), (7,11), (10,11), (10,12), (10,14), (11,14), (12,14)\}$

in the clique is called the level of the clique. This means that a clique with i members is called a level i clique. The example graph in Figure 1 have 28 cliques by this definition; the empty set, all the level one cliques containing each of the 14 vertices, 10 level two cliques determined by the edges and the three level three cliques $\{2, 3, 6\}$, $\{10, 12, 14\}$ and $\{10, 11, 14\}$. We denote the set of all cliques in a graph G by $\mathcal{C}$.

2.2 Markov Random Fields Defined on Graphs

A random field is a collection of random variables where each variable takes a value in a given set. If we now let x_p be a discrete random variable assigned to the vertex $p \in V$ and $\Omega = \{1, ..., L-1\}$ be the sample space for all $x_p, p \in V$, we get a discrete random field. In order for this field to be a Markov random field we need the variables to have a Markov property defined on a graph G. That the variables have a Markov property defined on a graph means that the full conditional of the random variable x_p is only dependent of the variables in its neighbourhood, $x_{\partial p}$. Thus we have

$$P(x_p|x_{-p}) = P(x_p|x_{\partial p}) \ \forall p \in V, \tag{1}$$

where $x_{-p} = (x_1, ..., x_{p-1}, x_{p+1}, ..., x_n)$. We see that the edges in the graph G now represent dependencies between the variables $x_p, p \in V$. If $L = 2$ we get a binary

Markov random field and the sample space is now $\Omega = \{0, 1\}$.

The Hammersley-Clifford theorem (Hurn et al, 2003) gives a general form of the distribution of a Markov random field. It states that a distribution satisfying $\pi(\mathbf{x}) > 0$, where $\mathbf{x} = (x_p, p \in V)$, for all possible assignments of values to all of a network's random variables is a Markov random field if and only if it has a joint density of the form

$$\pi(\mathbf{x}) = Z \exp\left(\sum_{\mathrm{C}\in\mathcal{C}} f_{\mathrm{C}}(\mathbf{x}_{\mathrm{C}})\right) \tag{2}$$

for some feature functions $\{f_{\mathrm{C}}, C \in \mathcal{C}\}$, where $\mathbf{x}_C = (x_p, p \in C)$. Here Z is a normalizing constant and $\mathcal{C}$ is the set of all cliques in the graph of the Markov random field. The normalizing constant Z is generally given as

$$Z = \left(\sum_{\mathbf{x}\in\Omega^n} \exp\left(\sum_{\mathrm{C}\in\mathcal{C}} f_{\mathrm{C}}(\mathbf{x}_{\mathrm{C}})\right)\right)^{-1}. \tag{3}$$

If we let the feature function $f_{\mathrm{C}}(\mathbf{x}_{\mathrm{C}})$ be

$$f_C(\mathbf{x}_C) = \theta_C I(x_p = 1 \forall p \in C), \tag{4}$$

where θ_C is an associated parameter to the clique $C, \in \mathcal{C}$, we get the density

$$\pi(\mathbf{x}) = Z(\theta) \exp\left(\sum_{C\in\mathcal{C}} \theta_C I(x_p = 1 \forall p \in C)\right), \tag{5}$$

which satisfies the Hemmersley-Clifford theorem. The normalizing constant is now a function of the associated parameters $\theta = (\theta_C, C \in \mathcal{C})$. As the number of vertices increase the normalizing constant, $Z(\theta)$, will often get more and more time consuming to compute. In some cases, however, the Markov structure simplifies the computation of the normalizing constant, see for example the discussion in Friel and Rue (2007).

3 The Spike and Slab Prior

In the Bayesian approach to statistics the parameters are considered to be quantities whose variation can be described by a probability distribution called the prior distribution. After a data sample is obtained the prior distribution is updated with the sample information. This updated prior is called the posterior distribution. This means that the Bayesian approach considers uncertainties associated with all

unknown quantities; observed or unobserved. To construct the posterior distribution Bayes' law, $\pi(\theta|x) \propto \pi(x|\theta)\pi(\theta)$, is used. The choice of a good prior model is essential because different prior models will give different posterior distributions. If the prior model is too strong or too weak relative to the likelihood, the posterior will be too similar to the prior, or the effect of the prior will be negligible. By using the Bayesian approach on Markov random fields we can infer the posterior distribution of the parameters and the graph structure of the fields.

The spike and slab prior is a mixture distribution where the spike is a point mass at zero and the slab is a widely spread distribution. The spike part of the distribution controls the sparsity of the structure in the posterior distribution and the slab part have a mild shrinkage effect on the parameters of the existing edges even in a highly sparse model. The spike and slab prior model used by Chen and Welling (2012) is a hierarchical model and is defined as follows

$$\theta_C = Y_C A_C, \tag{6}$$

where

$$Y_C \sim Bern(p_0) \tag{7}$$

and

$$A_C \sim N(0, \sigma_0^2). \tag{8}$$

$Y_C \in \{0, 1\}$ is a binary random variable representing the existence of the edges in the clique C and A_C is the actual value of the parameter associated with each clique. This means that

$$p(a_C|\sigma_0^2) = \frac{1}{\sqrt{\sigma_0^2}} \exp\left(-\frac{a_C^2}{2\sigma_0^2}\right), \tag{9}$$

$P(Y_C = 1) = p_0$ and $P(Y_C = 0) = 1 - p_0$. When $Y_C = 1$ we have

$$p(\theta_C|Y_C = 1, \sigma_0^2) = p(a_c|\sigma_0^2) = \frac{1}{\sqrt{\sigma_0^2}} \exp\left(-\frac{\theta_C^2}{2\sigma_0^2}\right). \tag{10}$$

Because the model is hierarchical, prior distributions for p_0 and σ_0^2 are defined. The distributions used are $p_0 \sim Beta(a, b)$ and $\sigma_0^{-2} \sim \Gamma(c, d)$. The distribution functions for p_0 and σ_0^2 are therefore given by

$$p(p_0) = \frac{1}{B(a, b)} p_0^{a-1}(1 - p_0)^{b-1} \tag{11}$$

and

$$p(\sigma_0^2) = \frac{1}{d^c \Gamma(c)} (\sigma_0^2)^{-(c+1)} \exp\left(-\frac{1}{\sigma_0^2 d}\right). \tag{12}$$

In their experiments Chen and Welling consider only pairwise features, which means that only cliques of level two or lower is included and that the values of θ_C for cliques with more than two members is zero.

4 Markov Chain Monte Carlo Methods

Markov chain Monte Carlo methods can be used when direct sampling from the posterior is time consuming or impossible. The technique is to use a Markov chain having limiting distribution equal to the desired distribution $\pi(\theta)$. In a Bayesian setting the desired distribution is the posterior distribution, or in other words the prior distribution updated with information from a dataset. The simulation of this chain is carried out until equilibrium is essentially reached. Different schemes are used to ensure that the limiting distribution is the desired one, examples of such schemes are Gibbs sampling, Metropolis-Hastings algorithms and reversible jump Markov chain Monte Carlo methods (RJMCMC). We will now describe Metropolis-Hastings algorithms, reversible jump Markov chain Monte Carlo methods and different proposal distributions for these schemes. For more information about Markov Chain Monte Carlo methods the reader is referred to Gamerman and Lopes (2006) and for a tutorial derivation of the reversible jump MCMC the reader is referred to Waagepetersen and Sorensen (2001).

4.1 Metropolis-Hastings Algorithms

A Metropolis-Hastings algorithm produce a sequence of random samples from the target distribution $\pi(\theta)$ and this sequence can be used to approximate the distribution. The algorithm can produce samples from any distribution provided it is possible to compute the value of some function that is proportional to that distribution. Because the mechanism is to use a Markov chain, the next sample value of the sequence is only dependent on the current sample value. If we let θ_{old} denote the current state of a Markov chain, one iteration of a Metropolis-Hastings algorithm is started by generating a potential new state θ_{new} from a proposal distribution $q(\theta_{new}|\theta_{old})$. The new state, θ_{new}, is then accepted with a probability

$$\alpha(\theta_{new}|\theta_{old}) = \min\left\{1, \frac{\pi(\theta_{new})q(\theta_{old}|\theta_{new})}{\pi(\theta_{old})q(\theta_{new}|\theta_{old})}\right\}, \tag{13}$$

and is otherwise rejected. This acceptance probability is defined such that we will accept moves to more probable states more often than moves to less probable states. Because of this we will tend to stay in hight-density regions of the desired

distribution, while low-density regions will be visited less often. The acceptance probability stated above results in a transition kernel given by

$$p(A|\theta_{old}) = \int_A q(\theta_{old}|\theta_{new})\alpha(\theta_{new}|\theta_{old})d\theta_{new} + I(\theta_{old} \in A)\left[1 - \int q(\theta_{new}|\theta_{old})\alpha(\theta_{new}|\theta_{old})d\theta_{new}\right], \quad (14)$$

for any subset A of the parameter space. The transition kernel of a Markov chain is a mixed distribution for the new state of the chain and it defines a distribution for every possible state of the chain given the current state. For more information about Markov chains and transition probabilities, the reader is referred to Ross (2007). We will now look at reversible jump Markov chain Monte Carlo Methods, which is a generalization of the Metropolis-Hastings algorithm, and then present some examples of proposal distributions for both reversible jump MCMC and Metropolis-Hastings.

4.2 Reversible Jump Markov Chain Monte Carlo Methods

The reversible jump Markov chain Monte Carlo method (Green,1996) makes simulations of the the target distribution possible even if the number of parameters is stochastic. Metropolis-Hastings algorithms can be seen as special cases of reversible jump MCMC methods. The model proposed in a reversible jump Markov chain Monte Carlo algorithm is a model with a different number of parameters than the current model. Generally we first propose a new model with a different number of parameters than the current one, then parameter values are proposed and the acceptance probability of the move is evaluated. As in the Metropolis-Hastings method we stay in the current state if the proposed state is not accepted.

In the acceptance probability of the method we now need to consider both the probability of choosing a specific model, and the distribution of the proposed parameter values. When proposing a move from the model with parameters θ_{old} to a model with parameters θ_{new} we need to generate a vector u_{old} of random variables from a proposal distribution $q(u_{old}|\theta_{old})$. We denote the probability of choosing this move by p_{new}. We then let the new parameters θ_{new} be a function of the current parameters and the random variables u_{old}; $\theta_{new} = g_\theta(\theta_{old}, u_{old})$. If we now propose a move in the opposite direction; a move from the model with parameters θ_{new} to the model with parameters θ_{old}, we would generate a vector u_{new} of random variables and let θ_{old} be a function of θ_{new} and u_{new}; $\theta_{old} = h_\theta(\theta_{new}, u_{new})$. We denote the probability of the move from the model with parameters θ_{new} to θ_{old} by p_{old}. The numbers of random variables in u_{new} and u_{old} needs to satisfy the following condition; $k_{new} + n_{C,new} = k_{old} + n_{C,old}$, where k_{new} is the length of u_{new}, k_{old} is

the length of u_{old}, $n_{C,old}$ is the number of parameters in the old model, and $n_{C,new}$ is the number of parameters in the new model. This condition is referred to by Waagepetersen and Sorensen (2001) as the crucial dimension matching condition. The condition ensures that the vectors of Markov chain states and proposal random variables are of equal dimensions. For the moves between models to be reversible, the mapping from (θ_{old}, u_{old}) to (θ_{new}, u_{new}) needs to be one to one. This means that we need functions $u_{new} = g_u(\theta_{old}, u_{old})$ and $u_{old} = h_u(\theta_{new}, u_{new})$ such that

$$\left.\begin{array}{l} \theta_{new} = g_\theta(\theta_{old}, u_{old}) \\ u_{new} = g_u(\theta_{old}, u_{old}) \end{array}\right\} \Leftrightarrow \left\{\begin{array}{l} \theta_{old} = h_\theta(\theta_{new}, u_{new}) \\ u_{old} = h_u(\theta_{new}, u_{new}). \end{array}\right. \tag{15}$$

The acceptance probability of a move from the model with parameters θ_{old} to a proposed model with parameters θ_{new} where the lengths of θ_{old} and θ_{new} is not necessarily the same is

$$\alpha(\theta_{new}|\theta_{old}) = \min\left\{1, \frac{\pi(\theta_{new})}{\pi(\theta_{old})} \frac{p_{new} q(u_{old}|\theta_{old})}{p_{old} q(u_{new}|\theta_{new})} |J|\right\}, \tag{16}$$

where

$$\begin{aligned} J &= \frac{\partial(\theta_{new}, u_{new})}{\partial(\theta_{old}, u_{old})} \\ &= \begin{bmatrix} \frac{\partial g_\theta(\theta_{old}, u_{old})}{\partial \theta_{old}} & \frac{\partial g_u(\theta_{old}, u_{old})}{\partial \theta_{old}} \\ \frac{\partial g_\theta(\theta_{old}, u_{old})}{\partial u_{old}} & \frac{\partial g_u(\theta_{old}, u_{old})}{\partial u_{old}} \end{bmatrix}. \end{aligned} \tag{17}$$

We will later see that the Jacobian in some cases will be one which simplifies the computation of the acceptance probability.

4.3 Proposal Distributions

When proposing a new state for our chain we need to propose a value for one or more of the parameters in our model. The proposal distribution from which these values are drawn, $q(\theta_{old}|\theta_{new})$, should be a known distribution that it is simple to generate realizations from. If the generation of a proposed next step is too complicated the point of using a Markov chain Monte Carlo method is gone. If the model contains several parameters we can propose to update one single parameter in each step or propose new values for several or all parameters before computing the acceptance probability.

To make the computation of the acceptance probability, $\alpha(\theta_{new}|\theta_{old})$, easier we can choose a distribution which is symmetric around the current parameter values,

this results in $q(\theta_{old}|\theta_{new}) = q(\theta_{new}|\theta_{old})$. When this is the case the proposal distributions will cancel out in the acceptance probability given in (13) and we get

$$\alpha(\theta_{new}|\theta_{old}) = \min\left\{1, \frac{\pi(\theta_{new})}{\pi(\theta_{old})}\right\}. \tag{18}$$

One example of a proposal distribution that is symmetric around the current parameter values is the normal distribution with expected value equal to the value in the current state of the chain; $\theta_{new} \sim N(\theta_{old}, I\sigma^2_{tune})$, where I is the identity matrix of the appropriate size.

The Gibbs sampler is a special case of the Metropolis-Hastings algorithm where only some of the parameters of our model are updated in each step. New values for these parameters are proposed from the distribution of these parameters conditioned on the remaining parameters in our model. This means that when updating the parameter θ_C we propose a new value $\theta_{C,new}$ from its full conditional given all other parameters; $\theta_{C,new} \sim \pi(\theta_C|\theta_{-C})$. When updating the parameter θ_C in a Gibbs step, the target distribution is the full conditional of this parameter; $\pi(\theta_C|\theta_{-C})$ and we get $\alpha(\theta_{new}|\theta_{old}) = 1$ in every step. This means that all proposed states are accepted. The Gibbs step is an effective method for sampling the variable when the full conditional is known. We will later see that this is the case for some of the variables in the Markov random field model when using the spike and slab prior.

For the reversible jump Markov chain Monte Carlo method we first need to propose a new model. There are several ways to do this. One way is to select one of all possible models that are not the current one and propose a move to this model. Another possibility is to select a parameter that could be a member of the model and then propose to add it if it is not a member of the current model or remove it if it is already a part of the current model. A third option is to first decide if we want to propose a model with more or less parameters than the current one, and then choose which of the parameters to add or remove. When a new model is chosen we need to propose values for each of the new or removed parameters, this is what is done when generating the random vectors u_{old} and u_{new} in Section 4.2. The random variables are generated from proposal distributions $u_{old} \sim q(u_{old}|\theta_{old})$ and $u_{new} \sim q(u_{new}|\theta_{new})$.

When considering a Markov random field it is natural to propose a model with one more or one less edge than the current one. When proposing a new model where the graph structure differs by one edge only we can start by deciding if we want to add or remove an edge. The simplest way to do this is to let the probability of adding an edge be p_{add} and the probability of removing an edge be $p_{remove} = 1 - p_{add}$. If the outcome is to add an edge we can choose one of the edges

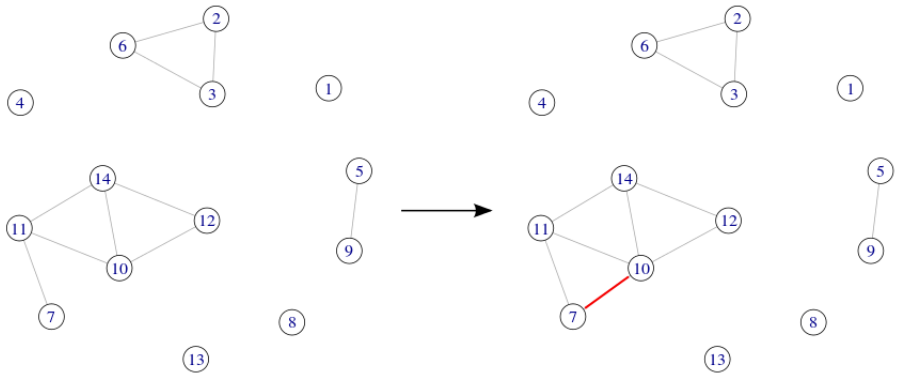

Figure 2: A move from the example graph in Section 2 to a graph with one more edge.

that is not present in the current model at random. The probability of adding a particular edge is then $1/(M - m_{old})$, where M is the number of possible edges in the graph in general and m_{old} is the number of edges in the current graph. An example move from one model to a model with more parameters is shown in Figure 2. The proposed move is a move from the graph presented in Section 2 to a graph with one more edge; the edge $(7, 10)$. This will result in two new cliques and thereby a model with two more parameters than the original one. The new cliques are $\{7, 10\}$ and $\{7, 10, 11\}$. The probability of choosing this move is $p_{add}/82$. When moving in the opposite direction, from the model with parameters θ_{new} to the model with parameters θ_{old}, the proposed graph is a graph with one less edge. This time we can simply select one of the edges in the current graph at random and remove it, or in other words; the probability of selecting the move is now $p_{remover}/m_{new}$. After proposing a new model we need to propose values of the vectors of random variables u_{new} and u_{old}. A simple way to do this is to propose values from a normal distribution with expected value zero; $N(0, I\sigma^2_{tune})$. When considering dimension matching for the move to a graph which contains one more or one less edge than the current graph, we see that the lengths of u_{new}, u_{old}, θ_{new} and θ_{old} satisfies the crucial matching condition. The function $g_\theta(\theta_{old}, u_{old})$ is now simply a function including the new parameters, and the function $h_\theta(\theta_{new}, u_{new})$ simply excludes the parameters that are removed. The dimensions of the vectors of Markov chain states and proposal random variables are in this case of equal dimension and the Jacobian of the move is one.

4.4 Acceptance Rate

The proportion of the steps in the algorithm that are accepted is called the acceptance rate. This rate should not be too low or too high. The acceptance rate is determined by how much the parameters are changed in each iteration. If the proposal distribution suggests small changes in each parameter the acceptance probability $\alpha(\theta_{new}|\theta_{old})$ will be close to one and many steps will be accepted. The result of this will be that the convergence of the chain is very slow as it will take many steps for the chain to traverse the whole parameter space. On the other hand, if we propose larger moves the acceptance probability will be small. A low acceptance probability increase the probability of staying in the same state for several steps. This means that if the acceptance rate is too low we will rarely move from one state to another. It is therefore important to keep the acceptance rate in an acceptable range. An acceptance rate between 0.2 and 0.5 for Metropolis-Hastings algorithms is suggested by Gamerman and Lopes (2006).

5 Bayesian Model

This section contains a presentation of the Bayesian model for our Markov random field. The prior model used is slightly different from the one presented in Section 3 and cliques of all levels are included in the model. We will first present the full prior model, then the likelihood function used to update the prior and then the posterior model is derived from the prior model and the likelihood.

5.1 Prior Model

We will now present a slightly different version of the spike and slab prior presented in Section 3. The main difference will be in the definition of the binary random variable Y_C. The prior model presented in Section 3 assigns a binary random variable Y_C and a normal random variable A_C to each of the cliques in the clique set $\mathcal{C}$. We will change this part slightly by only assigning the random variable Y_C to the cliques of level two. The cliques of level one will always be members of $\mathcal{C}$ and because of this no Y_C is assigned to these parameters. This gives us the

following definition of the parameter θ_C;

$$\theta_C = \begin{cases} A_C & \text{when } C = \{p\} \text{ for a } p \in V \\ A_C Y_C & \text{when } C = \{p, q\} \text{ for } p, q \in V, p \neq q \\ A_C & \text{when } C \in \mathcal{C}, |C| \geq 3 \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

The joint prior distribution for the parameters θ, σ_0^2 and p_0 is

$$\begin{aligned} \pi(\theta, \sigma_0^2, p_0) &= p(p_0)p(\sigma_0^2)p(\theta|\sigma_0^2, p_0) \\ &= p(p_0)p(\sigma_0^2)\left(\prod_{C\in\mathcal{C}} p(\theta_C|p_0, \sigma_0^2)\right), \end{aligned} \tag{20}$$

where $p(p_0)$, $p(\sigma_0^2)$ is as given in (11) and (12). The form of $p(\theta_C|p_0, \sigma_0^2)$ is given by the cases in (19) and the prior distributions of A_C and Y_C is as given in Section 3.

5.2 Likelihood

As mentioned in Section 3 the prior model needs to be updated with information from a dataset. We assume that the value of each of the random variables associated to the vertices in the graph G is observed multiple times to form a dataset consisting of N data points; $\mathbf{x} = (x^{(1)}, ..., x^{(N)})$. Each data point consists of observed values for each vertex such that $x^{(i)} = \{x_p^{(i)} \forall p \in V\}$. Two different realizations of a binary Markov random field defined on the example graph presented in Section 2 is shown in Figure 3, the black vertices represent the value one and the white vertices represent the value zero. The graph to the left in Figure 3 results in the data point $x = (0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0)$ and the one to the right results in the data point $x = (0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0)$. By using the example feature function in (4) and the resulting distribution function in (5), and assuming each of the N datapoints in $\mathbf{x}$ are independent, we get the following likelihood function for our model;

$$\begin{aligned} f(\mathbf{x}|\sigma_0^2, p_0, \theta) &= \prod_{i=1}^{N} f(x^{(i)}|\sigma_0^2, p_0, \theta) \\ &= Z(\theta)^N \prod_{i=1}^{N} \exp\left\{\sum_{C\in\mathcal{C}} \theta_C I(x_p^{(i)} = 1, \forall p \in C)\right\}. \end{aligned} \tag{21}$$

The more data points we have in our dataset, the more information about the underlying distribution we get.

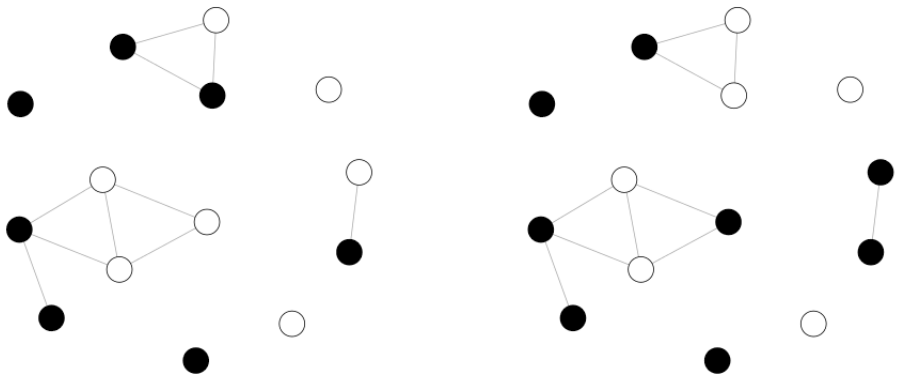

Figure 3: Two different realizations of a binary MRF defined on the graph in Figure 1

5.3 Posterior Model

When using Bayes' law for updating the prior model in (20) with information from the dataset we get the posterior model

$$\begin{aligned}\pi(\sigma_0^2, p_0, \theta|\mathbf{x}) &= f(\mathbf{x}|\sigma_0^2, p_0, \theta)\pi(\sigma_0^2, p_0, \theta) \\ &= \left(\prod_{i=1}^{N} f(x^{(i)}|\sigma_0^2, p_0, \theta)\right) p(p_0)p(\sigma_0^2)\left(\prod_{C\in\mathcal{C}} p(\theta_C|p_0, \sigma_0^2)\right), \quad (22)\end{aligned}$$

where the first factor is as given in (21), and the remaining factors are as given in Section 5.1. Depending on which of the parameters we want to update in our Markov chain Monte Carlo method different factors of the posterior model will remain unchanged and allow for simpler computation of the acceptance probability. If we for example keep the value of p_0 and σ_0^2 while updating the values of θ both $p(\sigma_0^2)$ and $p(p_0)$ will cancel out and we get

$$\pi(\sigma_0^2, p_0, \theta|\mathbf{x}) \propto \left(\prod_{i=1}^{N} f(x^{(i)}|\sigma_0^2, p_0, \theta)\right)\left(\prod_{C\in\mathcal{C}} p(\theta_C|p_0, \sigma_0^2)\right) \quad (23)$$

as a function of θ.

6 Algorithm and Proposal Mechanisms

In this section a Markov chain Monte Carlo method for the model in Section 5 is described. In each iteration we want to update p_0, σ_0^2, θ and the graph structure. First the values of p_0 and σ_0^2 are updated, then a change in all θ_C values are proposed. After this we propose to add an edge with probability p_{add}, or to remove an edge with probability $p_{remove} = 1 - p_{add}$. Finally we propose to change a single θ_C value before starting a new iteration. Which θ_C to update is selected at random. In the following the proposal mechanisms we use are described in more detail.

6.1 Updating p_0 and σ_0^2

In each iteration of the method we want the parameters p_0 and σ_0^2 to be updated while keeping the other parameters unchanged as proposed in Section 5.3. By removing every factor in (22) that is not a function of p_0 we get the full conditional of p_0;

$$\pi(p_0|\theta, \sigma_0^2, y_c, a_c, \mathbf{x}) \propto p(p_0) \left(\Pi_{c\in\mathcal{C}} p(y_c|p_0)\right). \tag{24}$$

Then by inserting the distribution functions for the prior models for the different parameters given in Section 3 we get

$$\begin{aligned} \pi(p_0|\theta, \sigma_0^2, y_C, a_C, \mathbf{x}) &\propto p_0^{a-1}(1-p_0)^{b-1}\Pi_{C\in\mathcal{C}} p_0^{y_c}(1-p_0)^{1-y_c} \\ &\propto p_0^{(a+\sum_{c\in\mathcal{C}} y_c)-1}(1-p_0)^{(b+\sum_{c\in\mathcal{C}}(1-y_c))-1}. \end{aligned} \tag{25}$$

This means that the full conditional of p_0 follows a Beta-distribution with parameters $\tilde{a} = a + n_C$ and $\tilde{b} = b + \sum_{C\in\mathcal{C}}(1 - y_C)$. As mentioned in Section 5; when it is possible to produce samples from the full conditional we can use a Gibbs step for updating p_0. The Beta-distribution is simple to draw values from, and because of this, a Gibbs step is a good option for updating the value of p_0.

We also want to update the value of σ_0^2 in each iteration. Because $\sigma_0^{-2} \sim \Gamma(c, d)$ the parameter σ_0^2 follows an inverse gamma distribution. By using (22) in the same way as we did for p_0 we get that the full conditional of σ_0^2 is

$$\begin{aligned} \pi(\sigma_0^2|\theta, p_0, y_c, a_c, \mathbf{x}) &\propto p(\sigma_0^2)\left(\Pi_{c\in\mathcal{C}} p(a_c|\sigma_0^2)\right) \\ &\propto (\sigma_0^2)^{-(c+1)} \exp\left\{-1\Big/\sigma_0^2 d\right\}\left(\Pi_{c\in\mathcal{C}} p(\theta_C|\sigma_0^2, y_C)\right) \\ &\propto (\sigma_0^2)^{-(c+(\frac{n_c}{2})+1)} \exp\left\{-\frac{1}{\sigma_0^2}\left(\frac{1}{d} + \sum_{C\in\mathcal{C}} \frac{\theta_C^2}{2}\right)\right\} \end{aligned} \tag{26}$$

and we see that the posterior distribution of σ_0^2 is an inverse gamma distribution with parameters $\tilde{c} = c + \frac{n_C}{2}$ and $\tilde{d} = (d^{-1} + \sum_{C\in\mathcal{C}} \theta_C/2)^{-1}$. Because of this we choose to use a Gibbs step for updating σ_0^2. This means that both p_0 and σ_0^2 can be updated separately from the rest of the parameters in the model and can be kept fixed when updating the other parameters or changing the graph structure. Keeping the values of p_0 and σ_0^2 fixed when updating the other parameters will simplify the acceptance probabilities of those steps.

6.2 Updating θ Values

When updating the values of the parameters in the model we can choose to update the value of a single parameter, θ_C, or the values of all the parameters, θ, at once. In our method both updating all parameters and a single parameter will be proposed in each iteration. In both cases a normal proposal distribution is used for the parameter values, this will gives us the benefits of a symmetric proposal distribution mentioned in Section 4 and we get the form of the acceptance probability given in (18). The values of p_0 and σ_0 is fixed both when updating a single parameter and when updating all of them.

If we want to update the values of all $\theta_{old} = (\theta_{1,old}, ..., \theta_{n_C,old})$ we first propose new values for all n_C parameters by drawing random samples from the normal distribution $N(\theta_{C,old}, \sigma^2_{all})$, independently. We denote the proposed parameter values as $\theta_{new} = (\theta_{1,new}, ..., \theta_{n_C,new})$ and get the following expression for the acceptance probability;

$$\alpha_\theta(\theta_{new}|\theta_{old}) = \min\left\{1, \frac{f(\mathbf{x}|\theta_{new}, \sigma_0^2, p_0)}{f(\mathbf{x}|\theta_{old}, \sigma_0^2, p_0)} \frac{p(\theta_{new}|\sigma_0^2, p_0)}{p(\theta_{old}|\sigma_0^2, p_0)}\right\}. \tag{27}$$

By using the posterior model given in (23) we get

$$\alpha_\theta(\theta_{new}|\theta_{old}) = \min\left\{1, \frac{\left(\prod_{i=1}^N f(x^{(i)}|\sigma_0^2, p_0, \theta_{new})\right)\left(\prod_{C\in\mathcal{C}} p(\theta_C|p_0, \sigma_0^2)\right)}{\left(\prod_{i=1}^N f(x^{(i)}|\sigma_0^2, p_0, \theta_{old})\right)\left(\prod_{C\in\mathcal{C}} p(\theta_C|p_0, \sigma_0^2)\right)}\right\}, \tag{28}$$

where $f(x^{(i)}|\sigma_0^2, p_0, \theta_{old/new})$ is as given in (21). This expression is evaluated in each iteration. The normalizing constant in the likelihood of the proposed state needs to be evaluated in each iteration, and as we mentioned in Section 2 this can become increasingly time consuming when the graph or the sample space is large.

When updating a single parameter we propose a new value for one parameter $\theta_{C,new}$ by drawing a value from the normal distribution $N(\theta_{C,old}, \sigma^2_{one})$, in the same way as we did when updating all parameters at once. Now the rest of the parameters remain unchanged and the expression in (27) can be further simplified. By

removing every part of the posterior that is not a function of the changed variable, θ_C, we get

$$\frac{f(\mathbf{x}|\theta_{new}, \sigma_0^2, p_0)}{f(\mathbf{x}|\theta_{old}, \sigma_0^2, p_0)} = \frac{Z(\theta_{new})^N \Pi_N \exp\{\theta_{C,new} I(x_p^{(i)} = 1, \forall p \in C)\}}{Z(\theta_{old})^N \Pi_N \exp\{\theta_{C,old} I(x_p^{(i)} = 1, \forall p \in C)\}} \tag{29}$$

and

$$\frac{p(\theta_{new}|\sigma_0^2, p_0)}{p(\theta_{old}|\sigma_0^2, p_0)} = \frac{\exp\{\frac{\theta_{C,new}^2}{2\sigma_0^2}\}}{\exp\{\frac{\theta_{C,old}^2}{2\sigma_0^2}\}} \tag{30}$$

for the factors of the acceptance probability. Updating one variable while keeping all the others fixed makes it easier to get acceptance of a proposed larger move than would be possible when updating all the parameters at once. This will give faster convergence of the chain. Proposing larger moves when updating a single parameter than when updating all at once means that $\sigma_{one}^2 > \sigma_{all}^2$.

6.3 Updating the Graph Structure

When updating the graph structure we propose to add or remove an edge with probabilities p_{add} and $p_{remove} = 1 - p_{add}$ by the method presented in Section 4. The move will be as described in Section 4.3; first determine if we are to propose adding an edge or removing an edge, then select an edge to add or remove. When moving from one model to another where the two models do not have the same number of edges in this way, the acceptance probability given in Section 4.2 becomes

$$\alpha_{add/remove}(\theta_{new}|\theta_{old}) = min\left\{1, \frac{f(\mathbf{x}|\theta_{new})}{f(\mathbf{x}|\theta_{old})} \frac{\pi(\theta_{new}, p_0, \sigma_0^2)}{\pi(\theta_{old}, p_0, \sigma_0^2)} \frac{q(\theta_{old}|\theta_{new})}{q(\theta_{new}|\theta_{old})}\right\}, \tag{31}$$

where the proposal distributions will be as presented in Section 4.3. Based on the value of $\alpha_{add/remove}(\theta_{new}|\theta_{old})$ the move from the current graph to the proposed graph is accepted or rejected. If the step is not accepted the current graph is kept. We will now first present the acceptance probability when adding an edge, and then when removing an edge.

6.3.1 Adding a New Edge

When proposing to add an edge we select an edge that is not already in the model at random and propose to add this edge. The move proposed is a move from the current graph to a graph with one more edge. This will as mentioned in Section 4 result in at least one new clique. We let n_{new} denote the number of new cliques in

the proposed graph and n_{old} the number of parameters in the current graph. The proposed graph will then have the associated parameters $\theta_{new} = (\theta_1, ..., \theta_{n_{old}+n_{new}})$ and the parameters in he current graph will be $\theta_{old} = (\theta_1, ..., \theta_{n_{old}})$. If we now let θ_{added} be the parameters that are new in the proposed graph, we can write $\theta_{new} = (\theta_{old}, \theta_{added})$. We will now look at the acceptance probability in (31) each factor at a time. We start by looking at the first factor in the expression and get

$$\begin{aligned}
\frac{f(\mathbf{x}|\theta_{new})}{f(\mathbf{x}|\theta_{old})} &= \frac{f(\mathbf{x}|\theta_{old}, \theta_{added})}{f(\mathbf{x}|\theta_{old})} \\
&= \left(\frac{Z(\theta_{new})}{Z(\theta_{old})}\right)^N \frac{\Pi_{i=1}^N \exp\{\Sigma_{\theta_{old},\theta_{added}} \theta_C I(x_p^{(i)} = 1, \forall p \in C)\}}{\Pi_{i=1}^N \exp\{\Sigma_{\theta_{old}} \theta_C I(x_p^{(i)} = 1, \forall p \in C)\}} \\
&= \left(\frac{Z(\theta_{new})}{Z(\theta_{old})}\right)^N \prod_{i=1}^N \exp\{\Sigma_{\theta_{added}} \theta_C I(x_p^{(i)} = 1, \forall p \in C)\} \qquad (32)
\end{aligned}$$

by using that $\theta_{new} = (\theta_{old}, \theta_{added})$ and the likelihood function given in (21). For the second factor in $\alpha_{add}(\theta_{new}|\theta_{old})$ we get

$$\begin{aligned}
\frac{\pi(\theta_{new}, p_0, \sigma_0^2)}{\pi(\theta_{old}, p_0, \sigma_0^2)} &= \frac{p(p_0)p(\sigma_0^2)p(\theta_{new}|p_0, \sigma_0^2)}{p(p_0)p(\sigma_0^2)p(\theta_{old}|p_0, \sigma_0^2)} \\
&= \frac{p(y_{new}|p_0)}{p(y_{old}|p_0)} \frac{p(a_{new}|\sigma_0^2)}{p(a_{old}|\sigma_0^2)} \\
&= \frac{p_0^{m_{new}}(1-p_0)^{M-m_{new}}}{p_0^{m_{old}}(1-p_0)^{M-m_{old}}} \prod_{\theta_{added}} \exp\left\{-\frac{\theta_C^2}{2\sigma_0^2}\right\}, \qquad (33)
\end{aligned}$$

were we have used the distribution functions from Section 3, the posterior given in (22) and the fact that all factors that are the same for both models cancel out. The last part of the acceptance probability is given by the proposal distributions. By using the proposal mechanism for adding or removing edges presented in Section 4.3 we get the following expression for the third and last factor of the acceptance probability;

$$\begin{aligned}
\frac{q(\theta_{old}|\theta_{new})}{q(\theta_{new}|\theta_{old})} &= \frac{(1/m_{new})p_{remove}}{(1/(M-m_{old}))p_{add}(2\pi)^{-n_{new}/2}(\sigma_{AR}^{-1})\exp\{\Sigma_{C=1}^{n_{new}} \theta_C^2/2\sigma_{AR}^2)\}} \\
&= \frac{p_{remove}(M-m_{old})(2\pi)^{n_{new}/2}\sigma_{AR}}{p_{add}m_{new}\exp\{\Sigma_{C=1}^{n_{new}} \theta_C^2/2\sigma_{AR}^2\}}, \qquad (34)
\end{aligned}$$

when the proposed values of the new parameters in the proposed state is drawn from a normal distribution $N(0, I\sigma_{AR}^2)$. We see that the dimension of the normal proposal match the dimension of the normal prior model which makes the acceptance probability dimensionless as desired.

6.3.2 Removing an Existing Edge

If we are not proposing to add an edge, the alternative is to remove an edge. The proposed graph will be a graph with one less edge than the current one and this will result in one or more cliques being removed from the model. If we let θ_{new} be the parameters in the proposed graph and θ_{old} be the parameters in the current graph we now get $\theta_{old} = (\theta_{new}, \theta_{removed})$. We will now look at each factor of the acceptance probability in (31) for the case when an edge is removed. For the first factor we now get

$$\begin{aligned} \frac{f(\mathbf{x}|\theta_{new})}{f(\mathbf{x}|\theta_{old})} &= \frac{f(\mathbf{x}|\theta_{new})}{f(\mathbf{x}|\theta_{new}, \theta_{removed})} \\ &= \left(\frac{Z(\theta_{new})}{Z(\theta_{old})}\right)^N \left(\Pi_{i=1}^N \exp\{\Sigma_{\theta_{removed}} \theta_C I(x_p^{(i)} = 1, \forall p \in C)\}\right)^{-1} \quad (35) \end{aligned}$$

in the same way as we did when adding an edge. For the second factor we now get

$$\begin{aligned} \frac{\pi(\theta_{new}, p_0, \sigma_0^2)}{\pi(\theta_{old}, p_0, \sigma_0^2)} &= \frac{p(p_0)p(\sigma_0^2)p(\theta_{new}|p_0, \sigma_0^2)}{p(p_0)p(\sigma_0^2)p(\theta_{old}|p_0, \sigma_0^2)} \\ &= \frac{p_0^{m_{new}}(1-p_0)^{M-m_{new}}}{p_0^{m_{old}}(1-p_0)^{M-m_{old}}} \left(\prod_{\theta_{removed}} \exp\{-\frac{\theta_C^2}{2\sigma_0^2}\}\right)^{-1}. \quad (36) \end{aligned}$$

The ratio of the proposal distributions for the move from the new to the old model and from the old to the new model is now

$$\frac{q(\theta_{old}|\theta_{new})}{q(\theta_{new}|\theta_{old})} = \frac{p_{add} m_{old} \exp\{\Sigma_{C=1}^{n_{removed}} \theta_C^2 / 2\sigma_{AR}^2\}}{p_{remove}(M - m_{new})(2\pi)^{n_{new}/2}\sigma_{AR}}, \quad (37)$$

again by using the proposal mechanism for adding or removing edges presented in Section 4.3. When adding or removing an edge the normalizing constant needs to be computed for the proposed model, this can be time consuming when the number of vertices is large. In this case we also see that the dimension of the normal proposal match the dimension of the normal prior model which makes the acceptance probability dimensionless.

7 Simulations and Results

In this section results from simulations of binary Markov random fields defined on different graphs are presented. A natural first step before testing the method on a real dataset where the underlying distribution is unknown is to test the method on a model where we know the underlying distribution. The goal is to test the algorithm on different graphs and datasets to see if it is working properly.

7.1 Simulated Data

When testing the algorithm we need a dataset to base the simulations on. If this dataset is sampled from a known distribution it will be possible to compare the results of the simulations with the underlying distribution. To make example graphs for testing our algorithm we first specify a graph structure and parameter values for all associated parameters. Datasets of different sizes are then sampled from the resulting Markov random field. Instead of listing each data point as a list of zeroes and ones we can assign a number to each of the possible outcomes. One way to do this is to read the data points as binary numbers and assign indexes based on this. This means that the data point with index two would for example represents $x^{(i)} = \{0, 0, 0, 1, 0\}$ for a graph with five vertices and the data point with index three for a graph with six vertices represents $x^{(i)} = \{0, 0, 0, 0, 1, 1\}$.

In both of the two following examples the corresponding parameters to each of the ten cliques of level one or higher in this graph are drawn from a normal distribution with expected value zero and variance $\left(\frac{1}{2}\right)^l \sigma_\theta^2$, where l is the number of elements in the clique. This means that $\theta_C \sim \mathcal{N}(0, \left(\frac{1}{2}\right)^l \sigma_\theta^2)$ and that most of the parameter values associated with cliques of a high level will be small compared to the ones associated to cliques with fewer members.

7.2 Results

We will now present results of simulations on two different example graphs; G_1 and G_2. The first graph is a small graph with six vertices and four edges. This graph is small enough so that simulations can be carried out for many iterations in a reasonable amount of time. The second graph is the example graph from Section 2. This graph is a bit larger and it is more time consuming to run simulations for the second graph for large numbers of iterations.

Example Graph $G_1 = \{V_1, E_1\}$

The first example graph that we want to test the algorithm for is the graph shown in Figure 4. We see that $n_1 = 6$ and $m_1 = 4$ for this example graph. The vertex set of this graph is $V_1 = \{1, 2, 3, 4, 5, 6\}$ and the edge set is $E_1 = \{(1, 2), (1, 4), (2, 4), (3, 6)\}$. The clique set, $\mathcal{C}_1$, of this graph contains 12 cliques; the empty set, one for each vertex, one for each edge and the level three clique $\{1, 2, 4\}$. Associated parameter values for each clique generated by the method described in Section 7.1 where $\sigma_\theta^2 = 1$ are listed in Table 1. The associated parameter

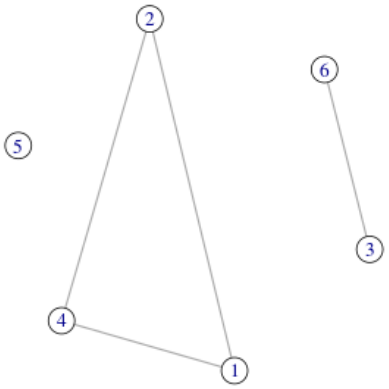

Figure 4: The undirected graph $G_1 = \{V_1, E_1\}$.

to the empty clique is assumed to be zero and not included in the simulations. The number of possible outcomes for a binary MRF defined on this graph is $2^{n_1} = 64$ and the number of possible edges is $M_1 = 15$. The datasets used for simulations on this example graph is sampled from the true distribution function given the parameter values in Table 1. Three different datasets are generated; $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ all based on the same underlying distribution. The number of data points in each dataset is $N_1 = 5000$, $N_2 = 100$ and $N_3 = 25$. Barplots for $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ can be seen in Figure 5. We see that more data points result in more information about the underlying distribution, which is as expected. The hyper parameters of the prior distributions of σ_0^2 and p_0 are set to $a = 1$, $b = 1$, $c = 3$ and $d = 3$ for all runs of the algorithm.

We now want to simulate the Markov random field defined on example graph G_1 by using the algorithm in Section 6. The chain is initialized with a graph with no edges and all parameter values set to zero. At the beginning of each iteration the values of p_0 and σ_0^2 are updated as described in Section 6.1. Then updating all θ values at once as presented in Section 6.2 is the proposed move, then adding or removing an edge as described in Section 6.4, and after that updating a random parameter value as described in Section 6.2. The steps are repeated in this order for the desired amount of iterations. The values of the tuning parameters for the two Metropolis-Hastings steps are found by trial and error and adjusted until the acceptance rates for the two moves are in the desired interval given in Section 4.4. The tuning parameter for the parameters added or removed in the reversible jump MCMC step is also set by trial and error. The task of finding an appropriate value

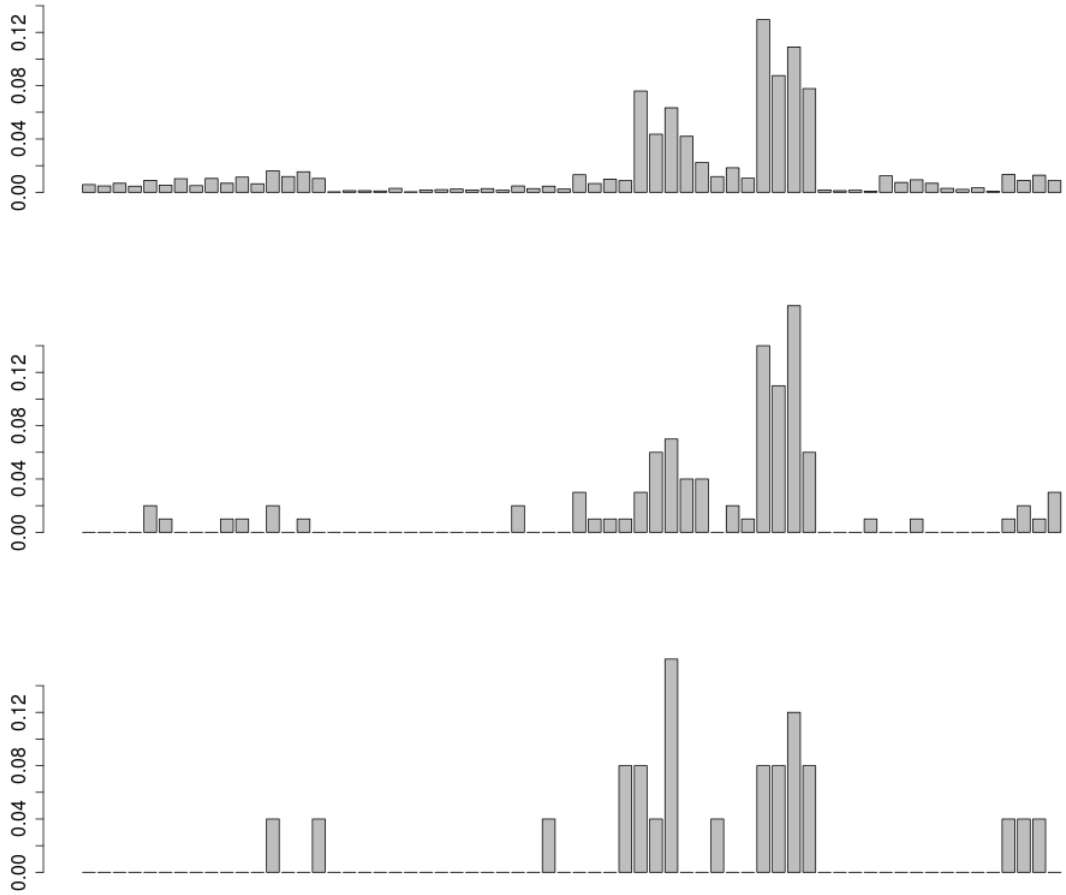

Figure 5: Barplots showing the frequencies of each outcome in the three simulated datasets $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$, sorted by index.

Clique	Parameter	Value
$\{1\}$	θ_1	0.593298
$\{2\}$	θ_2	-1.37835
$\{3\}$	θ_3	0.548659
$\{4\}$	θ_4	0.469639
$\{5\}$	θ_5	-1.10666
$\{6\}$	θ_6	-0.45754
$\{1,2\}$	$\theta_{1,2}$	-0.482739
$\{1,4\}$	$\theta_{1,4}$	1.37293
$\{2,4\}$	$\theta_{2,4}$	-0.0170734
$\{3,6\}$	$\theta_{3,6}$	0.0660981
$\{1,2,4\}$	$\theta_{1,2,4}$	-0.166371

Table 1: Values of $\theta_C, C \in \mathcal{C}_1$ for the example graph G_1

for this tuning parameter have been particularly hard. The main problem have been that both large and very small values have resulted in no edges being added or removed from the graph. The probability of adding an edge is $p_{add} = 0.5$ for all runs.

We start our testing by running simulations based on the dataset $\mathbf{x}_1$. The tuning parameters are adjusted to reach good acceptance rates and the resulting values are $\sigma^2_{one} = 0.105$, $\sigma^2_{all} = 0.0165$ and $\sigma^2_{AR} = 0.00025$. We run the algorithm with 1000000 iterations four times with the same values of hyper parameters and tuning parameters. We can compare the average probabilities of each realization obtained from the RJMCMC method to the frequencies of each outcome in the different datasets. In Figure 6 the vertical lines represent the empirical probabilities of each outcome in the dataset $\mathbf{x_1}$ and the stars represent the average probability of each outcome based on parameter values from the RJMCMC method. We see that the average probabilities of each outcome from the RJMCMC match the data frequencies quite good and that the resulting average probabilities are similar for all four runs of the algorithm. In Figure 7 we see QQ-plots of the frequencies of the average probabilities from the different runs compared to data frequencies. We see that all points are close to the straight line, which is yet another indicator that the models found by the algorithm fits the dataset quite well. If we now look at which edge are present for how many iterations in the four runs we get the graphs in Figure 8, a thicker edge represents an edge that is present in more of the iterations after some burn-in period than a thinner edge. We see that the resulting graph structures are similar but not exactly the same. Some of the most probable edges are present in all four graphs, while some edges are strong in only a few of them. One example is the edge $(2,4)$ which is only present in the bottom left graph. The

edges that are strong in all four graphs are $(1,2)$, $(1,4)$, $(1,5)$ and $(2,3)$, of these edges only two are present in the true graph; $(1,2)$ and $(1,4)$. The other edges in the true graph structure; $(2,4)$ and $(3,6)$, are not present in all four runs. The edge $(3,6)$ is not strong in either of the runs and the edge $(2,4)$ is only strong in one of the runs. That the resulting edge probabilities are not the same means that the method does not converge properly. The four runs are based on the same underlying distribution, they are performed for the same amount of iterations, the values of the tuning parameters are the same and the dataset used is the same. After convergence the four different chains should have the same quantitative and qualitative behaviour. When the resulting graph structure is not the same it means that the behaviour of the four chains is not the same. Even though the different runs give different models, all runs give good fits to the dataset. This means that we find different high probability regions of the posterior distribution, but that the chain is not able to move between these regions and find the best one.

If we now do a similar test for the dataset $\mathbf{x}_2$ we can look at the same kinds of figures. The tuning parameters for updating one or all parameters are now set to $\sigma^2_{one} = 0.3$ and $\sigma^2_{all} = 0.1$. The tuning parameter for the proposed values of the new or removed parameters is set to $\sigma^2_{AR} = 0.025$. The average probabilities of each outcome is shown together with the data frequencies in Figure 9 in the same way as for the previous dataset. We see that the fit is not as good as for the larger dataset, but that the fit is still quite good. QQ-plots for the average probabilities of each of the four RJMCMC runs compared to dataset $\mathbf{x}_2$ are presented in Figure 10. We see that all points are relatively close to the straight line which indicates that the fit is quite good for all four runs. The edge probabilities are now shown in Figure 11 for the four different runs. We see that there are only two edges that are strong inn all four runs; $(1,2)$ and $(1,5)$, and that only one of these; $(1,2)$, is present in the true graph in Figure 4. Also this time we see that the resulting edge structures are not the same and that the algorithm is not reaching convergence. Again each model is a good fit for the dataset, but which is the best one can not be identified.

We now want to test the algorithm for an even smaller dataset; $\mathbf{x}_3$. This dataset contains only 25 samples. This time the tuning parameters for updating one or all parameters are set to $\sigma^2_{one} = 0.8$ and $\sigma^2_{all} = 0.025$. The tuning parameter when adding or removing edges is now $\sigma^2_{AR} = 0.025$. In the same way as for the previous dataset the average probability of each outcome is shown together with the data frequencies in Figure 12. We see that the fit is not as good as for the two previous datasets. We also see that the resulting average probabilities of each run differ more than for the two previous datasets. The QQ-plots for the four different runs of average RJMCMC probabilities versus data frequencies can be seen in Figure 13. We see that these are not as close to the straight line as the ones in the two

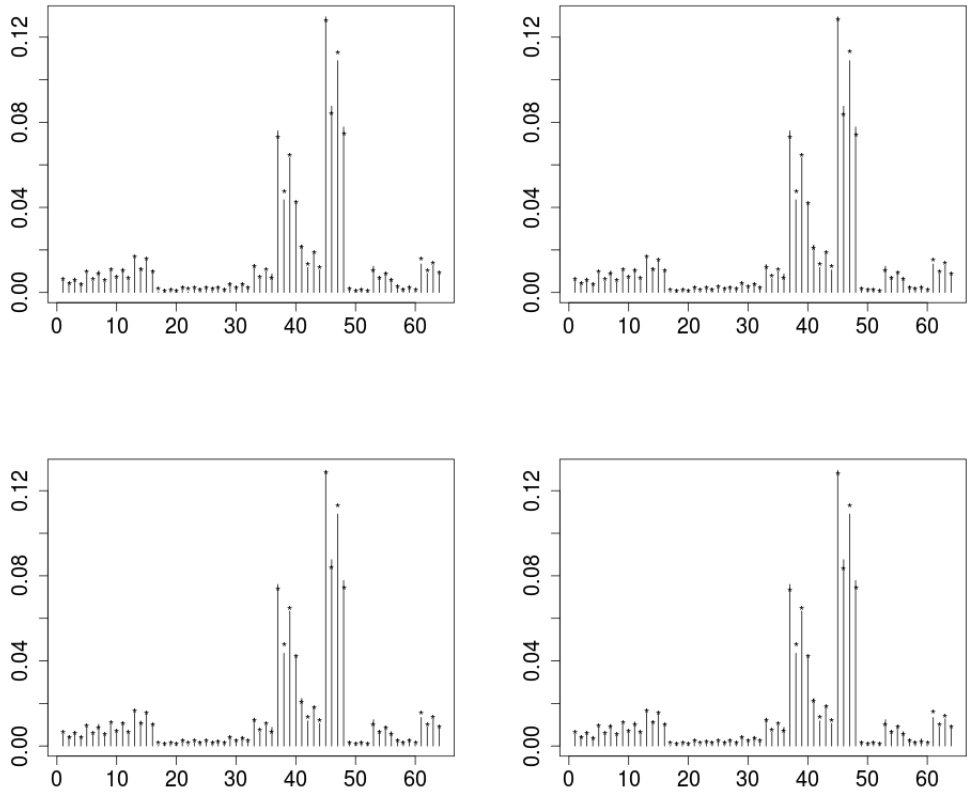

Figure 6: Vertical lines represent frequencies of each outcome in the datasets and the stars give the average probabilities based on simulated parameter values for four different RJMCMC runs for dataset $\mathbf{x}_1$.

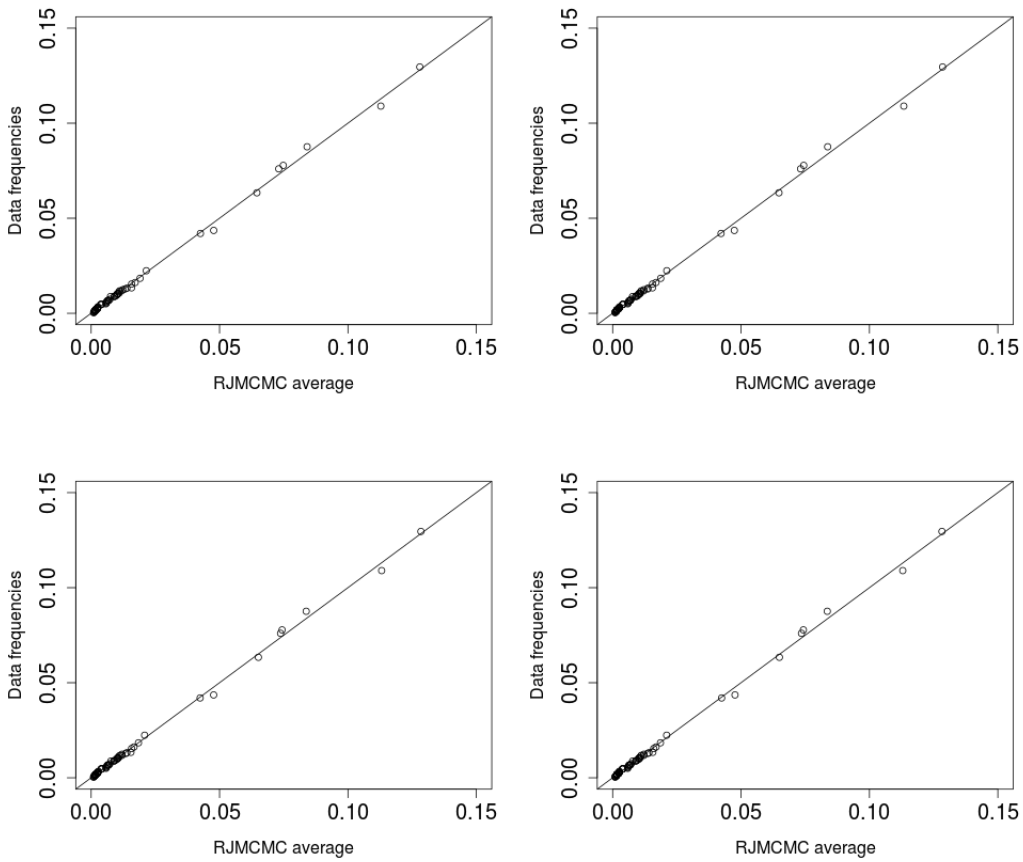

Figure 7: QQ-plot of average probabilities from four different RJMCMC runs compared with the frequencies of values in dataset $\mathbf{x}_1$

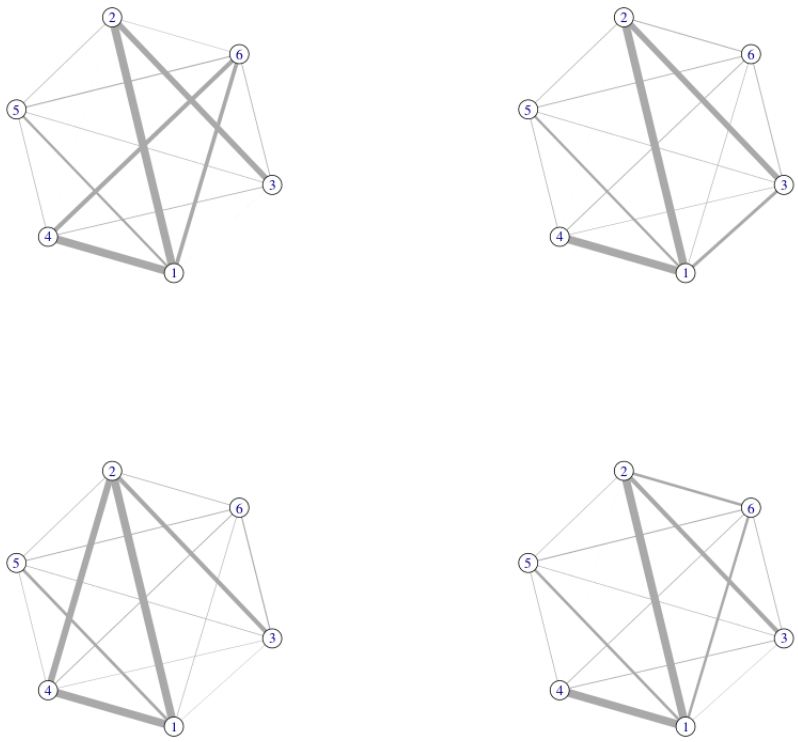

Figure 8: Probabilities of the different edges in each of the four RJMCMC runs for dataset $\mathbf{x}_1$, a thick edge represent an edge that is present for most of the steps during the run while a thin edge represents an edge that is rarely present in during the run.

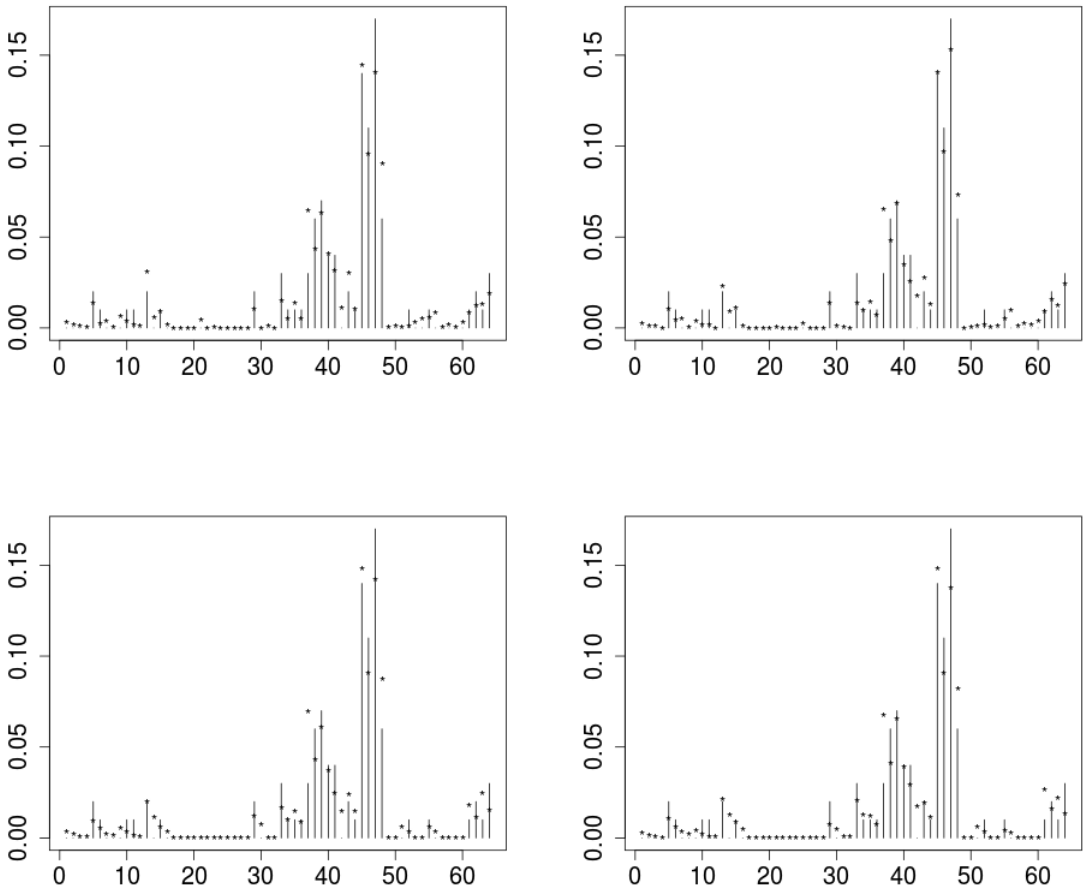

Figure 9: Vertical lines represent frequencies of each outcome in the datasets and the stars give the average probabilities based on simulated parameter values for four different RJMCMC runs for dataset $\mathbf{x}_2$.

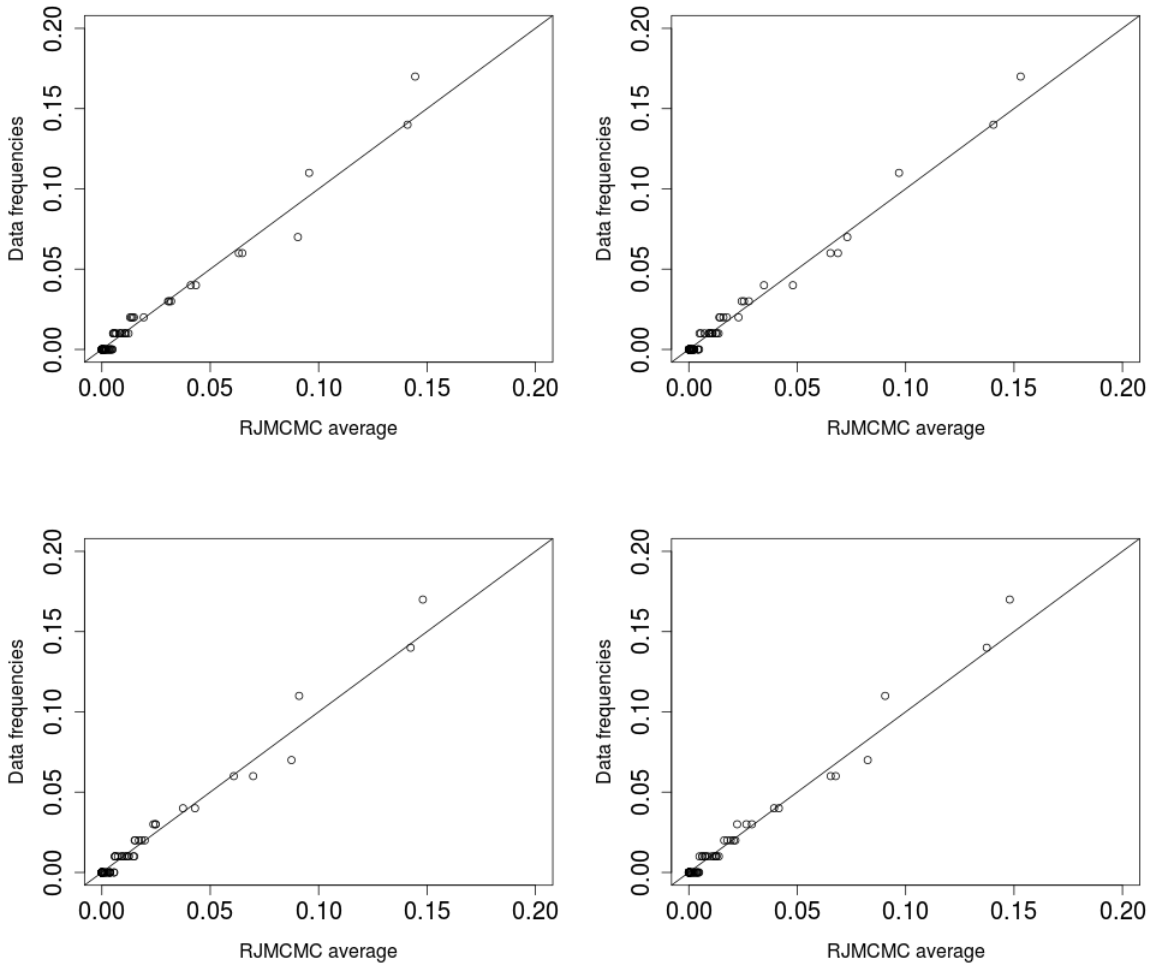

Figure 10: QQ-plot of average probabilities from four different RJMCMC runs compared with the frequencies of values in dataset $\mathbf{x}_2$

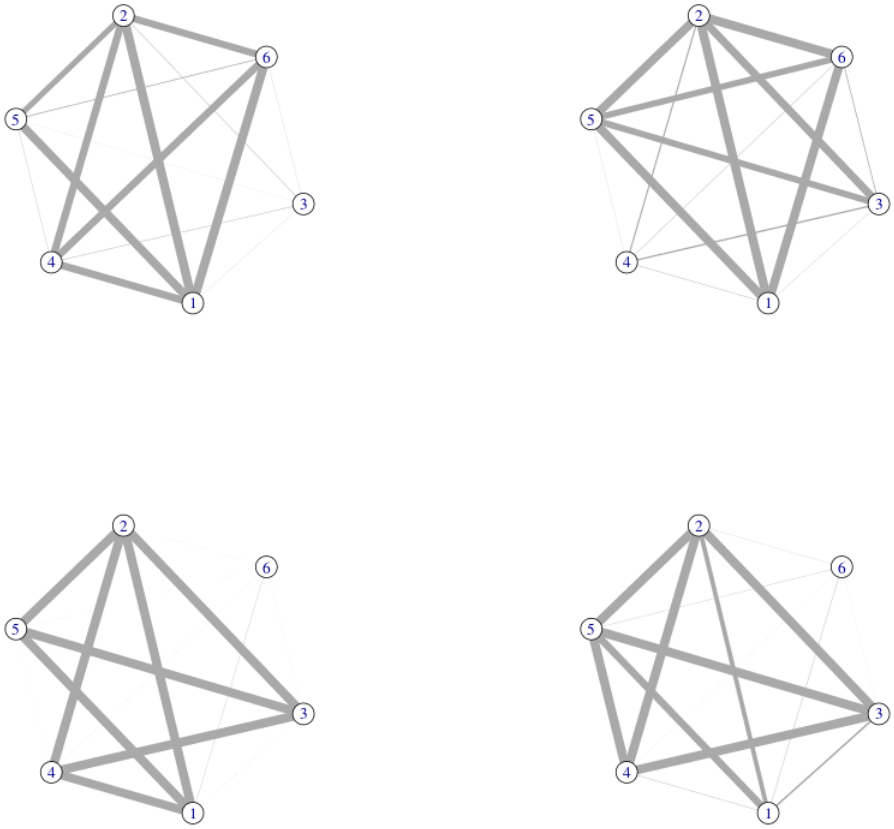

Figure 11: Probabilities of the different edges in each of the four RJMCMC runs for dataset $\mathbf{x}_2$, a thick edge represent an edge that is present for most of the steps during the run while a thin edge represents an edge that is rarely present in during the run.

previous experiments. This is as expected because the prior model will me more dominant compared to the likelihood of the data when the number of data points is low. The edge probabilities are now shown in Figure 14 for the four different runs. We see that these four runs result in different edge probabilities as well and that we also for this dataset get more strong edges than we did for the dataset with 5000 samples. This time there are four edges that are strong in all four runs; $(1, 2)$, $(1, 4)$, $(1, 6)$, and $(2, 3)$.

For all three datasets simulated from this underlying distribution we see that the resulting edge probabilities are not the same. This means that our chain is not converging properly. For all runs we find reasonably good fits, which means that the sample space might have several high probability regions and that our chains get stuck in such regions and do not move around the whole sample space. The chain seems to move into a high probability region and stay there. When the dataset is smaller the problem is larger than when the dataset is a bit larger. Also the algorithm seems to discover more strong edges when the dataset is small, this might mean that the prior probability of some edges being present in the model is too high. Generally very large real world datasets might be hard to find and a method that gives good results also for small datasets is desirable. This algorithm being able to identify high probability regions is a good start, but we also need to be able to move between these regions for the algorithm to reach convergence.

Example Graph $G_2 = \{V_2, E_2\}$

The second example graph is the example graph presented in Section 2. We now call this graph $G_2 = \{V_2, E_2\}$ and the graph structure can be seen in Figure 1. The parameter values generated as described in Section 7.1 where $\sigma_\theta^2 = 1$ can be seen in Table 2. Based on an MRF with these parameter values datasets $\mathbf{x}_4$, $\mathbf{x}_5$ and $\mathbf{x}_6$, containing $N_4 = 5000$, $N_5 = 100$ and $N_6 = 25$ data points are generated in the same way as for the first example graph. This time the barplots and the plots with the vertical lines and stars from the previous example will not be informative, as the number of possible outcomes is now 16384. Because of this we will now only look at the QQ-plots to see if the average probabilities from each RJMCMC chain fit the data, and look at the edge probabilities. The number of possible edges in this graph is 91. As for the previous example the hyper parameters of the prior distributions of σ_0^2 and p_0 are set to $a = 1$, $b = 1$, $c = 3$ and $d = 3$ for all runs of the algorithm.

As for the last example graph we now run four different chains based on each of the four datasets. All simulations are initialized with a graph with no edges

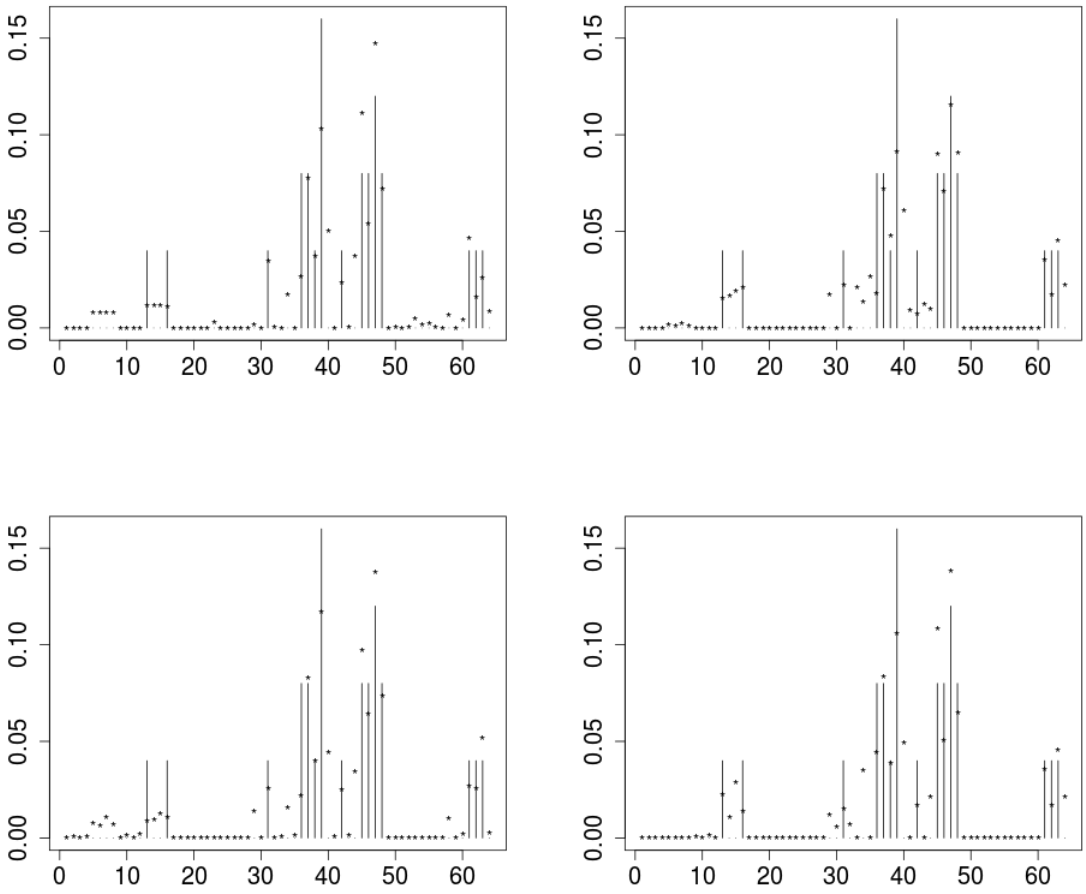

Figure 12: Vertical lines represent frequencies of each outcome in the datasets and the stars give the average probabilities based on simulated parameter values for four different RJMCMC runs for dataset $\mathbf{x}_3$.

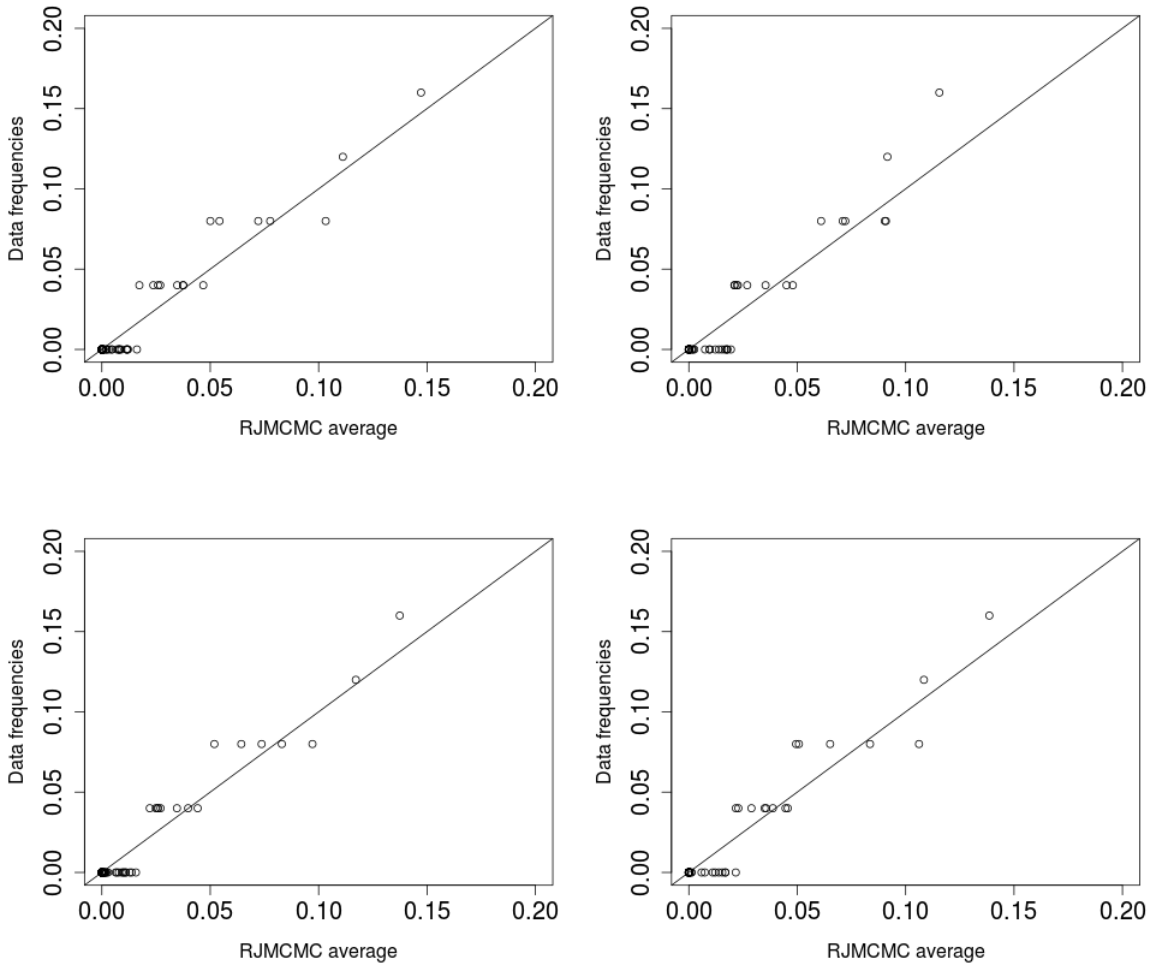

Figure 13: QQ-plot of average probabilities from four different RJMCMC runs compared with the frequencies of values in dataset $\mathbf{x}_3$

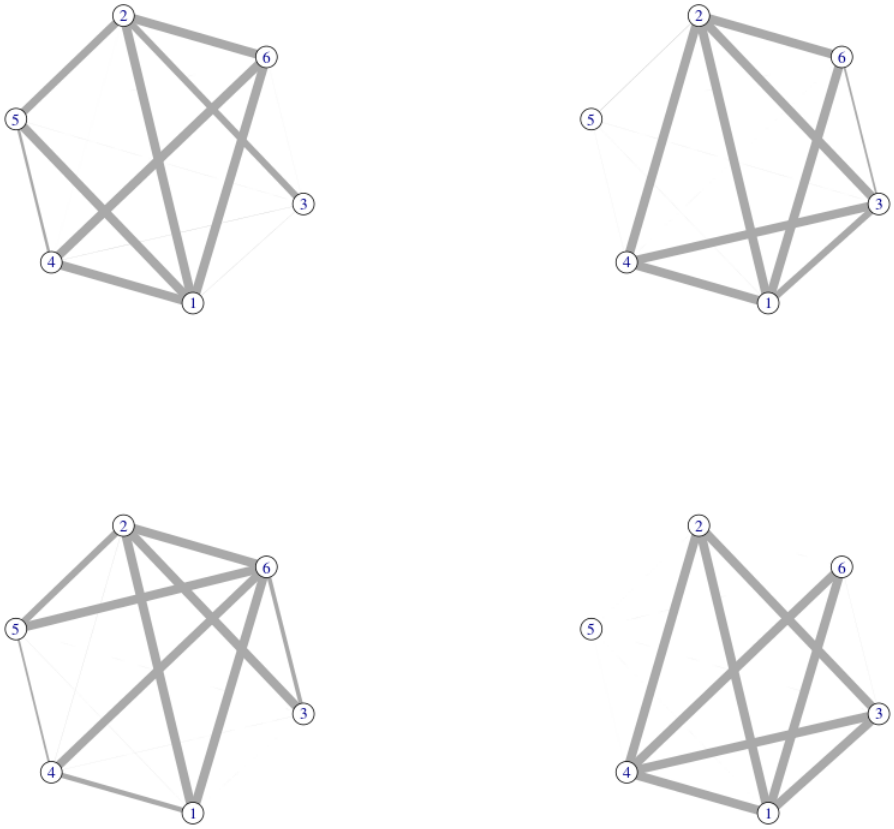

Figure 14: Probabilities of the different edges in each of the four RJMCMC runs for dataset $\mathbf{x}_3$, a thick edge represent an edge that is present for most of the steps during the run while a thin edge represents an edge that is rarely present in during the run.

Clique	Parameter	Value
$\{1\}$	θ_1	-0.573857
$\{2\}$	θ_2	-1.0375
$\{3\}$	θ_3	0.92405
$\{4\}$	θ_4	1.16424
$\{5\}$	θ_5	0.59804
$\{6\}$	θ_6	-0.315004
$\{7\}$	θ_7	1.16657
$\{8\}$	θ_8	0.076322
$\{9\}$	θ_9	2.094
$\{10\}$	θ_{10}	0.706495
$\{11\}$	θ_{11}	-1.53659
$\{12\}$	θ_{12}	0.583443
$\{13\}$	θ_{13}	-0.432034
$\{14\}$	θ_{14}	-0.467165
$\{2,3\}$	$\theta_{2,3}$	-0.248712
$\{2,6\}$	$\theta_{2,6}$	0.0952619
$\{3,6\}$	$\theta_{3,6}$	-0.0263643
$\{5,9\}$	$\theta_{5,9}$	0.430178
$\{7,11\}$	$\theta_{7,11}$	0.847827
$\{10,11\}$	$\theta_{10,11}$	0.530592
$\{10,12\}$	$\theta_{10,12}$	0.138162
$\{10,14\}$	$\theta_{10,14}$	-0.847798
$\{11,14\}$	$\theta_{11,14}$	-0.038785
$\{12,14\}$	$\theta_{12,14}$	-0.431356
$\{2,3,6\}$	$\theta_{1,2,4}$	-0.227529
$\{10,11,14\}$	$\theta_{10,11,14}$	0.352491
$\{10,12,14\}$	$\theta_{10,12,14}$	-0.179331

Table 2: Values of $\theta_C, C \in \mathcal{C}_2$ for the example graph G_2

and all parameter values set to zero. The values of the tuning parameters for the two Metropolis-Hastings steps and the tuning parameter for the reversible jump MCMC step are found by trial and error as for the previous example. The task of finding an appropriate value for the tuning parameter of the RJMCMC step have been hard also for this graph, and the problems described for the previous graph is also the case for this example graph. Every chain is run for 250000 iterations. As before we run four chains based on each dataset with the same settings of hyper parameters and tuning parameters. The probability of adding an edge is $p_{add} = 0.5$ for all runs.

The first of the three datasets for this example graph that is used for testing is the dataset $\mathbf{x}_4$. The tuning parameters are set by trial and error and the resulting values are $\sigma^2_{one} = 0.1$, $\sigma^2_{all} = 0.015$ and $\sigma^2_{AR} = 0.00015$. The QQ-plot of data frequencies versus average probabilities in Figure 15 show that all four runs find a model that fit the dataset quite well. A visualization of the edge probabilities for the four different runs based on the dataset $\mathbf{x}_4$ is shown in Figure 16. We again see that the resulting graphs are not the same, but that they have strong edges in the same areas. If we now look at some of the edges in the true graph in Figure 1 we see that the only edge in the true graph that is strong in all four runs is the edge $(10, 11)$, but several of the edges from the true graph is strong in three of the graphs; $(2, 3)$, $(10, 12)$ and $(12, 14)$. The only edges of the true graph that is not identified in any of the four runs are the edges $(11, 14)$ and $(2, 6)$. If we look at the parameter values associated with the cliques $\{2, 6\}$ and $\{11, 14\}$ in Table 2 we see that the values of these parameters is small compared to the associated values to other cliques of level two. The level two clique $\{3, 6\}$ does also have a small associated value, but this edge is discovered by two of the chains.

We now move on to simulations based on the dataset $\mathbf{x}_5$ and the tuning parameters are now set to $\sigma^2_{one} = 0.45$, $\sigma^2_{all} = 0.065$ and $\sigma^2_{AR} = 0.0185$. In Figure 17 we see the QQ-plots for four different runs of the algorithm based on the dataset $\mathbf{x}_5$. We see that this time the fit of the models found by our algorithm is not as good as for the previous tests. When the dataset contains only 100 samples and the number of possible outcomes is as large as 16384 the dataset gives very little information compared to the prior model. If we look at the edge probabilities in Figure 18 we see that the algorithm now discovers a lot more edges than before. The thickness of the edges is now scaled down to 0.5 of what is used for earlier graphs. This because of the large number of strong edges in these graphs. Despite the high number of strong edges we can see that some edges are strong in only one or two of the graphs and not in the others. Examples are the edges $(7, 11)$ and $(5, 9)$ that are present in one and two graphs, respectively. For these four runs we see that proposing to add an edge is accepted a lot more often than proposing to remove an edge. The result is as we can see that several edges are present in the graph

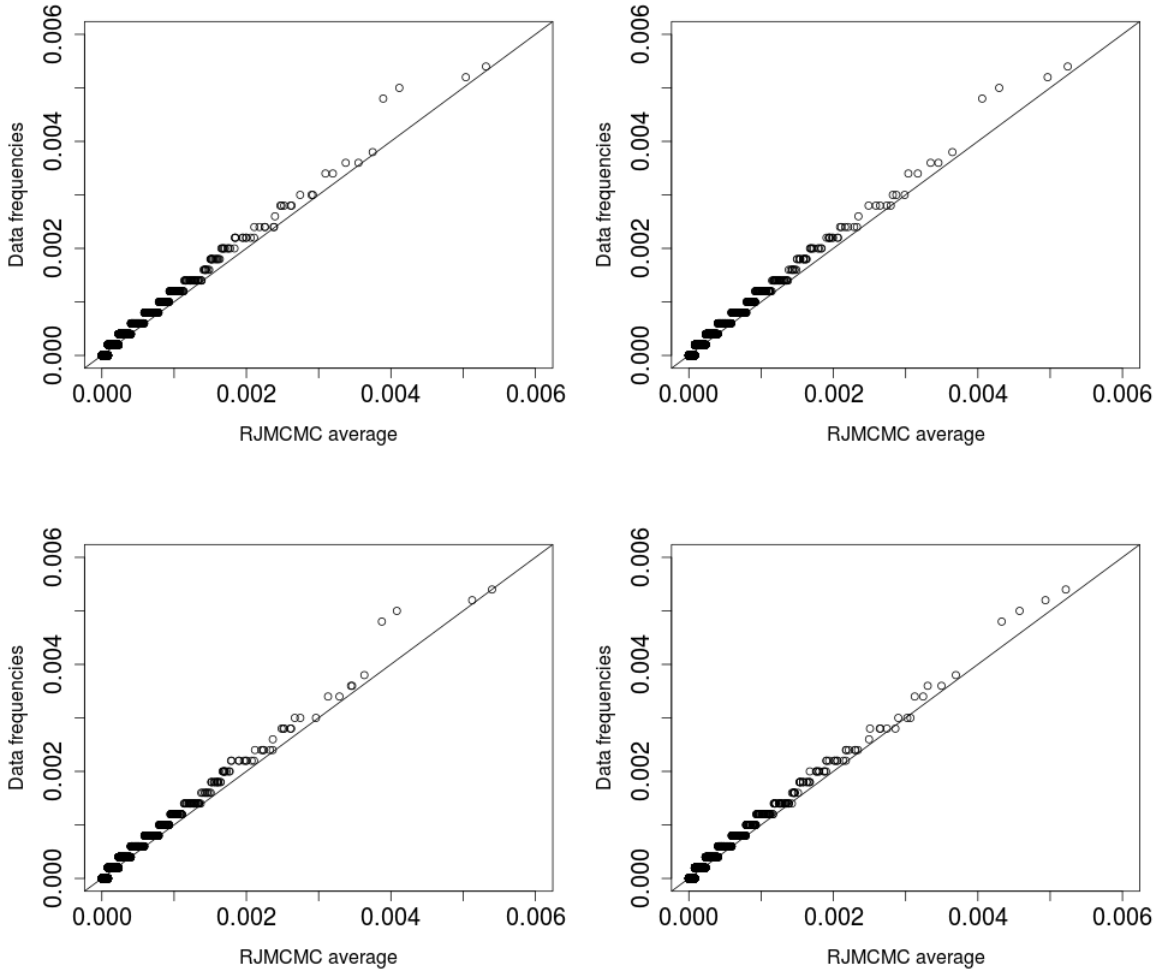

Figure 15: QQ-plot of average probabilities from four different RJMCMC runs compared with the frequencies of values in dataset $\mathbf{x}_4$

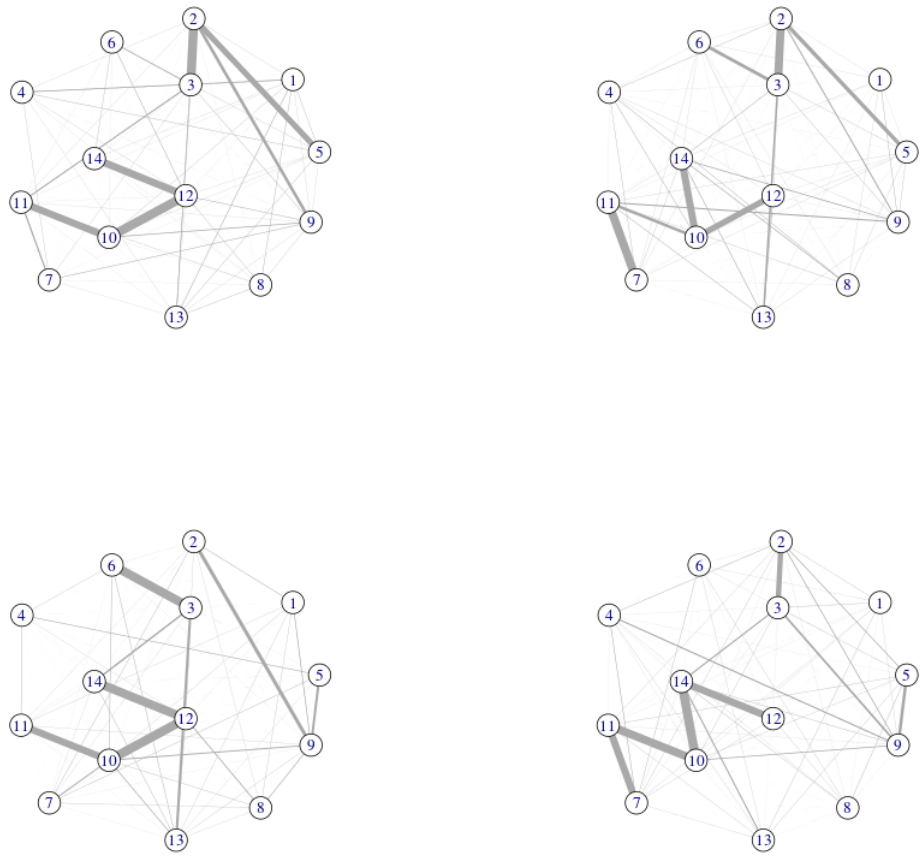

Figure 16: Probabilities of the different edges in each of the four RJMCMC runs for dataset $\mathbf{x}_4$, a thick edge represent an edge that is present for most of the steps during the run while a thin edge represents an edge that is rarely present in during the run.

for most of the iterations.

The even smaller dataset $\mathbf{x}_6$ is also used in four runs of the algorithm. The tuning parameters are now $\sigma^2_{one} = 1.35$, $\sigma^2_{all} = 0.13$ and $\sigma^2_{AR} = 0.014$. The QQ-plots of average probabilities and data frequencies can be seen in Figure 19 and we see that the fit this time is worse than for the previous tests. The edge probabilities in Figure 20 show the same problem as we saw for dataset $\mathbf{x}_5$; a great number of edges are strong in the resulting model. Again we can also find edges that are strong in some of the runs but not in others. Examples are the edges $(7, 11)$, $(4, 14)$ and $(5, 9)$.

For this second example graph we see the same problems as we did for the first graph, but this time we also see that the method is not capable of finding as good fits to the datasets when the number of data points is small compared to the number of possible outcomes of the MRF. As the number of vertices increases we need more and more data points to get an accurate representation of the underlying distribution. The convergence problems of the algorithm may be due to the acceptance rate for adding or removing edges being too low. We were not able to get an acceptance rate for adding or removing edges above 0.1 for any of the runs for any of the six datasets. Still we were able to get models that fit the largest dataset quite well.

8 Closing Remarks

In this thesis simulations of parameter values and graph structures for Markov random fields were considered in a Bayesian setting and a slightly altered version of the spike and slab prior used by Chen and Welling (2012) was used as the prior model. The method presented was a reversible jump Markov chain Monte Carlo method. Proposal distributions for the different types of moves were presented and the algorithm was tested for two different example graphs. One with few vertices and few edges, the other with more vertices but still fairly few edges. Simulations were done based on three different simulated datasets from the Markov random field defined on each of the example graphs. For the smallest example graph the algorithm could find good fits for all three datasets, but the resulting graph structures were not the same in each run. It seems that the chain is not able to move between different modes of the posterior, which results in the algorithm not reaching convergence. Still some of the edges from the true graph were found by the algorithm in all of the runs. For the larger example graph the algorithm found models that fit the largest dataset well, but the performance was not as good for the two smallest datasets. We see the symptoms of the algorithm not reaching

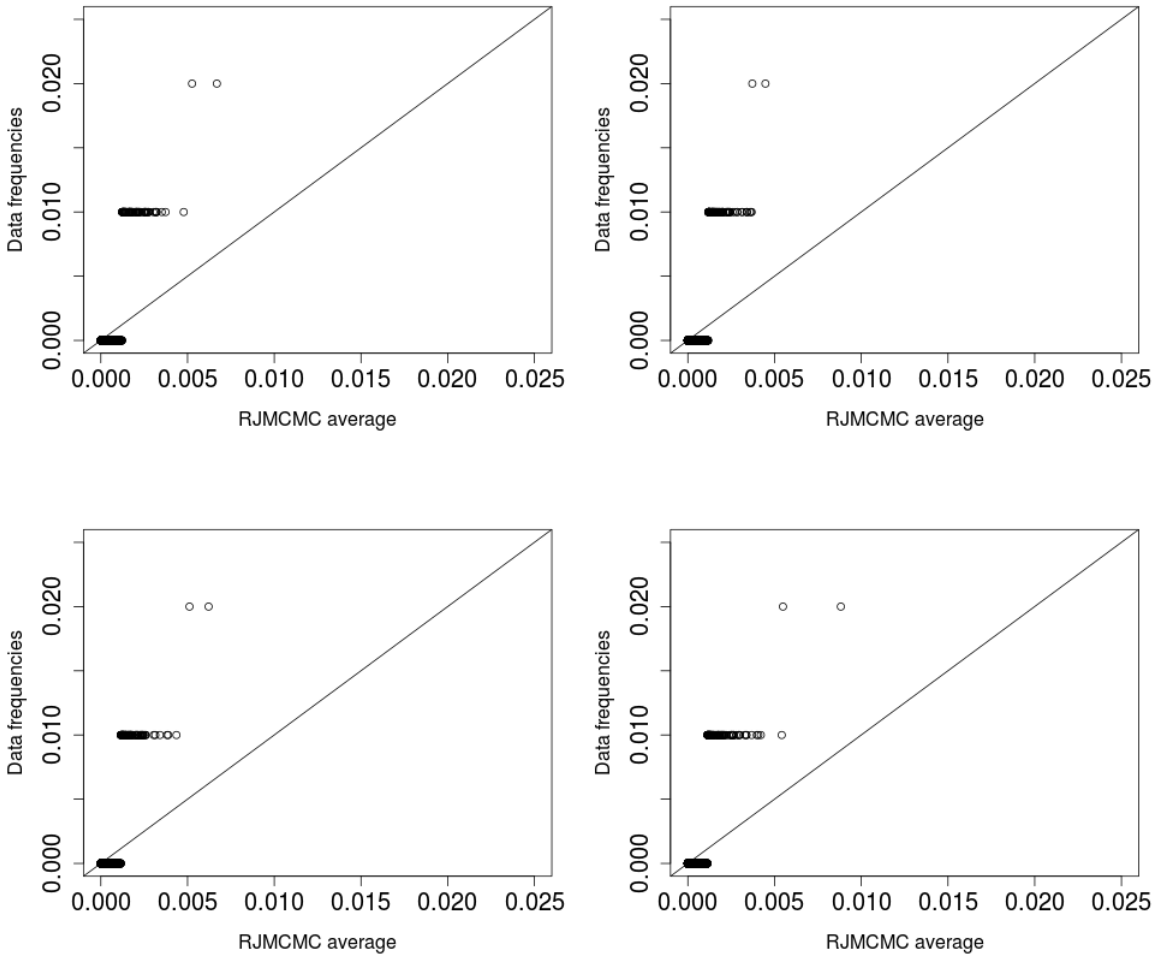

Figure 17: QQ-plot of average probabilities from four different RJMCMC runs compared with the frequencies of values in dataset $\mathbf{x}_5$

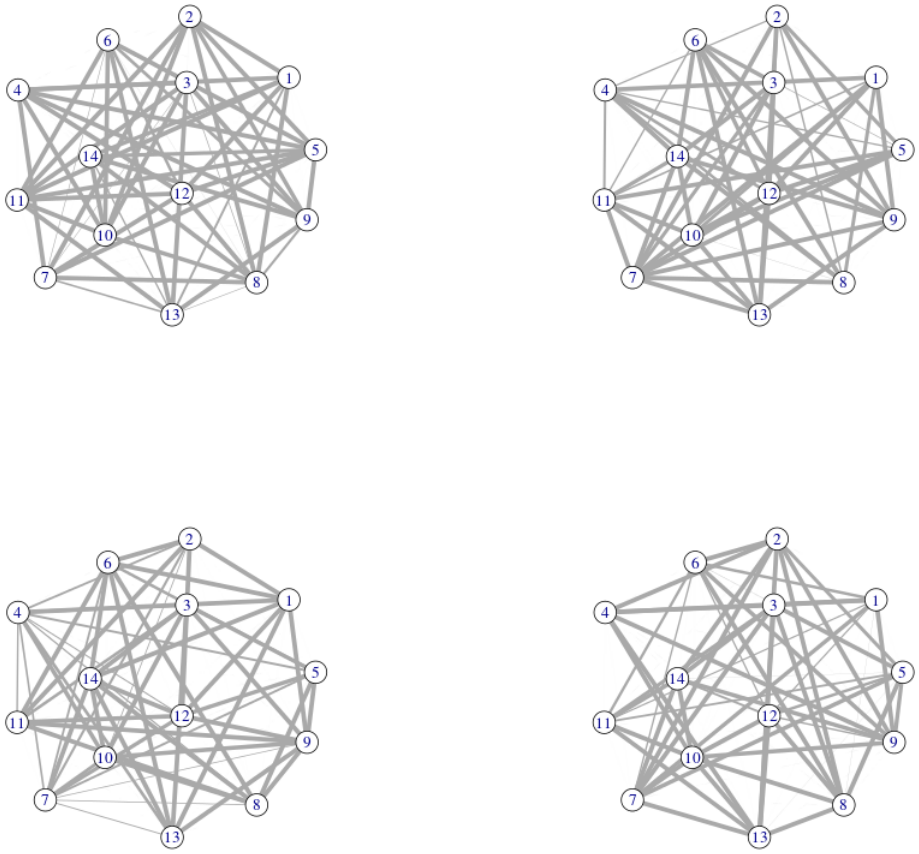

Figure 18: Probabilities of the different edges in each of the four RJMCMC runs for dataset $\mathbf{x}_5$, a thick edge represent an edge that is present for most of the steps during the run while a thin edge represents an edge that is rarely present in during the run. The edge thickness is scaled down by 0.5 compared to previous plots because of the large number of strong edges.

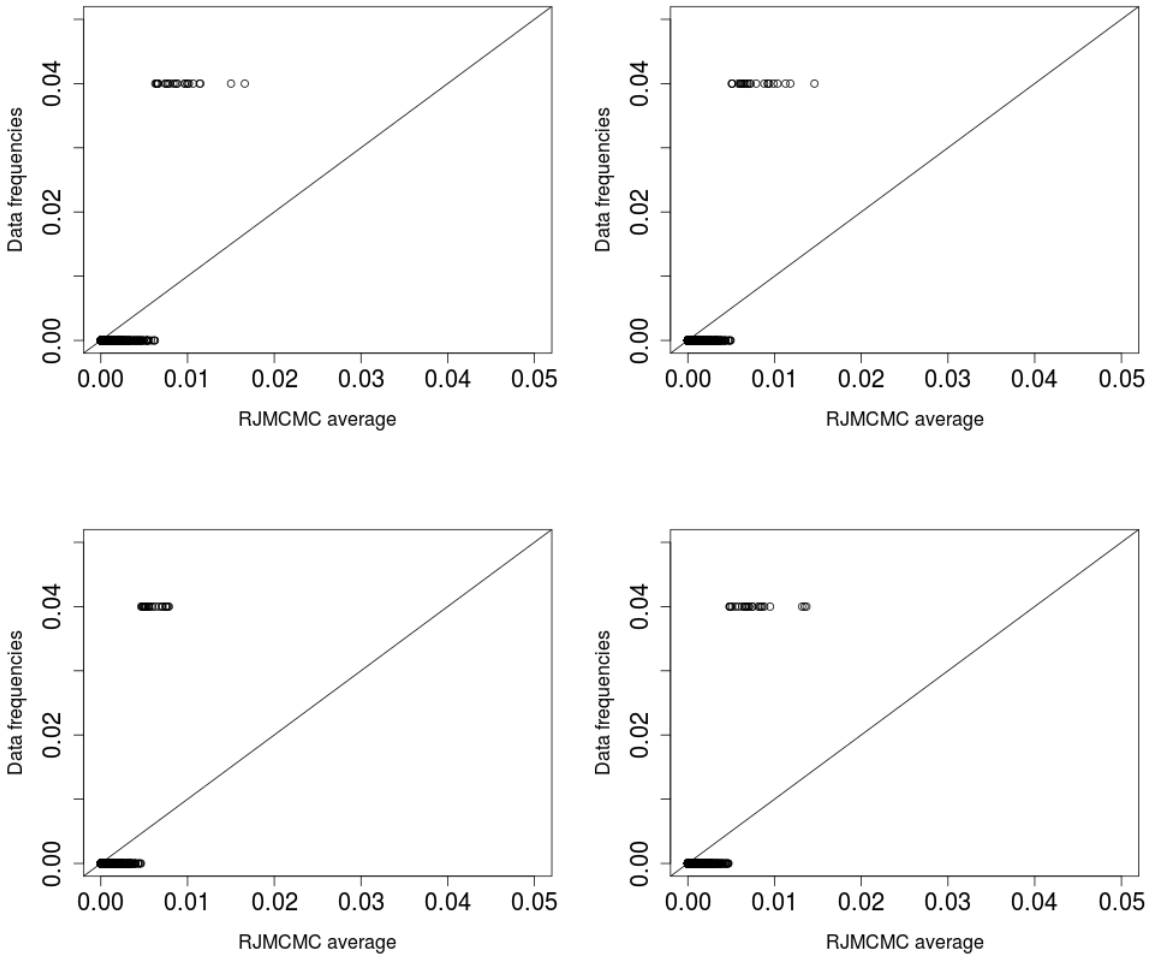

Figure 19: QQ-plot of average probabilities from four different RJMCMC runs compared with the frequencies of values in dataset $\mathbf{x}_6$

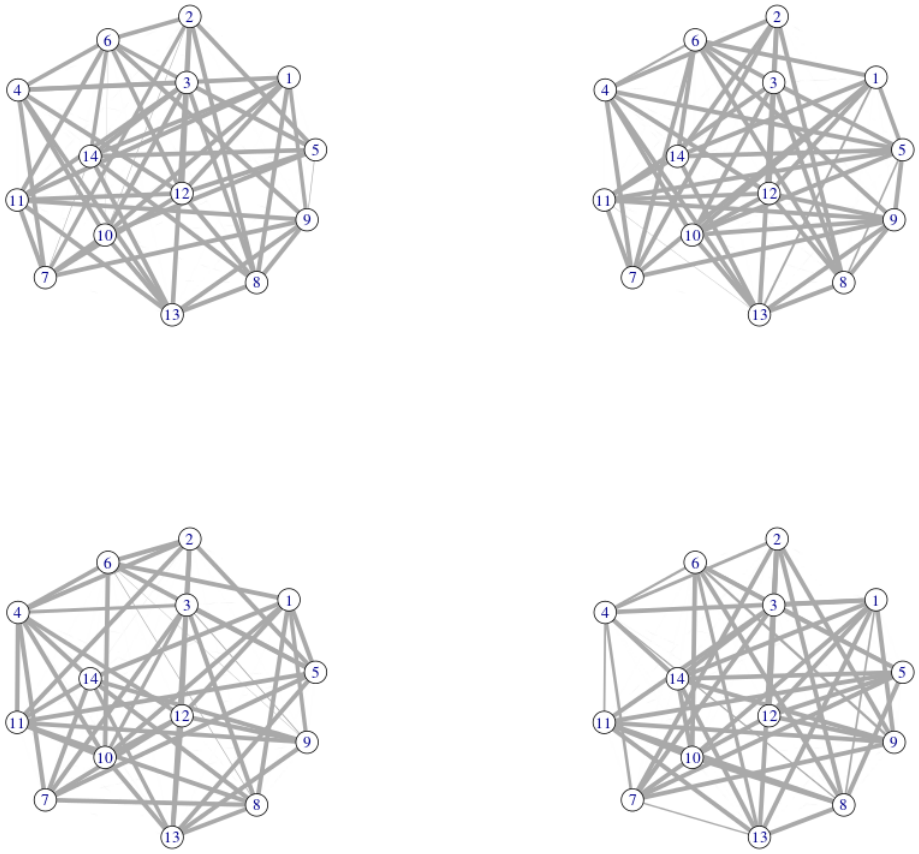

Figure 20: Probabilities of the different edges in each of the four RJMCMC runs for dataset $\mathbf{x}_6$, a thick edge represent an edge that is present for most of the steps during the run while a thin edge represents an edge that is rarely present in during the run. The edge thickness is also here scaled down by 0.5 compared to previous plots because of the large number of strong edges.

convergence for this example graph as well.

Setting a tuning parameter for the proposal distribution for the parameters that were added or removed when moving between models of different sizes was hard, and even after lots or trial and error the acceptance rate for adding or removing edges was still very low. For the smallest datasets from the second example graph we also saw that the number of times adding an edge is accepted is almost twice as high as the number of times removing an edge is accepted. This might bee the reason why the edges resulting from simulations were not the same as the true ones, and that the simulations ended up in different modes of the posterior.

A natural continuation of the work in this thesis is to find better proposal distributions. A different proposal distribution could make the acceptance rate when adding or removing edges higher and possibly solve the problem of the Markov chain not exploring the whole sample space. If the proposal distribution were to take into consideration the values of all parameters associated with the vertices included in cliques added when proposing a new edge and set the values of the new parameters according to this it could make the acceptance rate when adding or removing edges a bit larger.

Another possibility is to explore different prior models. The choice of a good prior model is an essential part of Bayesian analysis. The strength of the prior relative to the likelihood will have a major effect on results and a different prior might affect the performance of the algorithm. A prior model that is constructed to encourage sparse graphs as dimension increases is presented by Jones et al. (2005). It also might be possible to restrict the space of possible models to not include the most complicated models, this will make it hard to correctly learn all of the edges correctly but it might allow us to learn the most important edges more reliably. This approach is mentioned by Koller and Friedman (2009).

If the convergence problems of the method is resolved it would be natural to use an approximation for the normalizing constant, or look into the possibility of the Markov structure simplifying the computation of the normalizing constant as described by Friel and Rue (2007). Solutions to the problem of the intractable normalizing constant where the calculation of this constant is avoided is discussed in Murray and Ghahramani (2004). The next step thereafter would be to apply the method to a real dataset.

References

Y. Chen and M. Welling. Bayesian structure learning for Markov random fields with a spike and slab prior. arXiv:1206.1088v2 [stat.ML], 2012.

N. Friel and H. Rue. Recursive computing and simulation-free inference for general factorizable models. *Biometrica*, 94:661–672, 2007.

D. Gamerman and H. F. Lopes. *Markov Chain Monte Carlo*. Chapman & Hall/CRC, Taylor & Francis Group, LLC, 6000 Broken Sound Parkway NW, Suite 300, 2006.

P. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrica*, 82:711–732.

M.A. Hurn, O. K. Husby, and H. Rue. A tutorial on image analysis. In J.Møller, editor, *Spatial Statistics and Computational Methods*. Library of Congress Cataloging-in-Publication Data, 2003.

B. Jones, C. Carvalho, A. Dobra, C. Hans, C. Carter, and M. West. Experiments in stochastic computation for high-dimensional graphical models. *Statistical Science*, pages 388–400, 2005.

R. Kindermann and J.L. Snell. *Markov random fields and their applications*. American Mathematical Society, Providence, Rhode Island, 1980.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, Cambridge, Massachusetts, 2009.

I. Murray and Z. Ghahramani. Bayesian learning in undirected graphical models: Approximate MCMC algorithms. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 392–399, Arlington, Virginia, 2004. AUAI Press.

S. Parise and M. Welling. Bayesian model scoring in Markov random fields. In *NIPS*, pages 1073–1080, 2006.

S. M. Ross. *Introduction to Probability Models*. Elsevier Inc., 30 Corporate Drive, Suite 400, Burlington, MA 01803,USA, ninth edition, 2007.

R. Waagepetersen and D. Sorensen. A tutorial on reversible jump MCMC with a view toward applications in QTL-mapping. *International Statistical Rewiev*, 69:49–61, 2001.