NTNU
Norwegian University of
Science and Technology

Parametrization of multi-dimensional Markov chains for rock type modeling

Steinar Nerhus

Master of Science in Physics and Mathematics
Submission date: June 2009
Supervisor: Håkon Tjelmeland, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Description

The candidate will study multi-dimensional Markov chains for modeling of 2D images of rock types in petroleum reservoirs. In particular he will focus on how to make a suitable parametric model for this task and estimate model parameters by maximum likelihood from training images.

Assignment given: 19. January 2009
Supervisor: Håkon Tjelmeland, MATH

Preface

It is with great joy that I can say that this text is in fact my master thesis! It marks the end of five years of study, and the start of a new era in my life.

About the work, the parametrization that is used is actually the third method that I tried, unfortunately the two first did not work very well. I got the idea for the third parametrization from my supervisor Håkon Tjelmeland, and luckily it turned out to work a lot better, it would be depressing to write a master thesis about something that did not work. The implementation was done partly in C and partly in R. The forward-backward algorithm was coded in C, and then I linked to this code from R so that I could use the built in optimization routine in R. I also used the powerful plotting options in R to create some of my figures. Other figures were created in Paint (not optimal, but quite easy to use). This document is written in LaTeX, of course!

I would like to thank my supervisor Håkon Tjelmeland for his very helpful and much appreciated advise. I must also send my thanks to the unknown person who called the ambulance, to the doctors who took care of me at the hospital, and to the dentist fixing my teeth, after I managed to crash with my bike on the way to school a Saturday in May. Finally I must thank my friends and fellow students, without them the experience would have been a lot more frustrating. Thanks!

Steinar Nerhus

Trondheim, June 24, 2009

Abstract

A parametrization of a multidimensional Markov chain model (MDMC) is studied with the goal of capturing texture in training images. The conditional distribution function of each row in the image, given the previous rows, is described as a one-dimensional Markov random field (MRF) that depends only on information in the immediately preceding rows. Each of these conditional distribution functions is then an element of a Markov chain that is used to describe the entire image. The parametrization is based on the cliques in the MRF, using different parameters for different clique types with different colors, and for how many rows backward we can trace the same clique type with the same color. One of the advantages with the MDMC model is that we are able to calculate the normalizing constant very efficiently thanks to the forward-backward algorithm. When the normalizing constant can be calculated we are able to use a numerical optimization routine from R to estimate model parameters through maximum likelihood, and we can use the backward iterations of the forward-backward algorithm to draw realizations from the model. The method is tested on three different training images, and the results show that the method is able to capture some of the texture in all images, but that there is room for improvements. It is reasonable to believe that we can get better results if we change the parametrization. We also see that the result changes if we use the columns, instead of the rows, as the one-dimensional MRF. The method was only tested on images with two colors, and we suspect that it will not work for images with more colors, unless there are no correlation between the colors, due to the choice of parametrization.

Contents

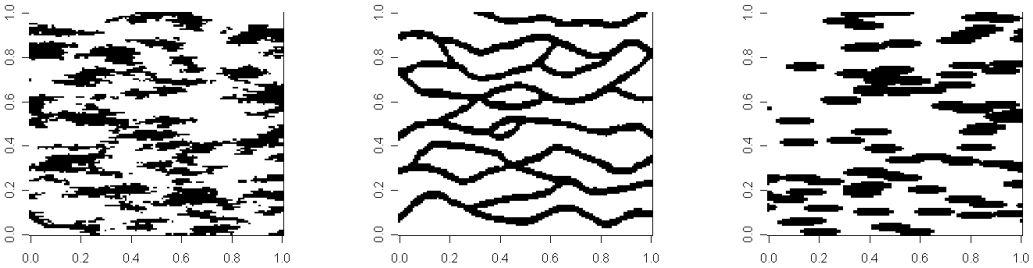
1 Introduction

If we are looking for petroleum in the Earth's subsurface, it can be to great help if we know how the rock types in the subsurface are distributed. There are certain elements that need to be in place for petroleum to be generated and stored, and if we have an image of the rocks in the subsurface we are able to look for these elements. We must for example have some kind of porous rock that can contain petroleum, like sandstone, with some dense rock above, like shale, that makes sure the petroleum does not escape. Reflection seismology is a tool to gather data about the rock types in the subsurface, and from this data one can make an image of how the rock types are distributed. The problem is that the method is not 100% accurate, and there will always be a lot of noise in the images. We want to be able to remove this noise, so that we are better able to say where petroleum can be stored.

To be able to remove the noise, we must first have some prior information about what the images should look like. One method is to look at how rock types are distributed on the surface, and assume that our image should be similar to this. An important aspect in this approach is how we say that two images are similar. It is then common to talk about texture in the images. Texture can be described as the underlying information that makes us able to say if two images are similar or not. For the human eye it is often easy to say that two images have the same texture, but it can be hard to say exactly what the texture is. If we are able to describe the texture in the images of the rocks on the surface, we could try to add this same texture into our images of the noisy data.

In statistics we have a method for this called Bayesian inversion. If the prior information about the texture is formulated as a probability distribution, called the prior model, we can draw images from this distribution that fits with the observed noisy data by using Bayesian inversion. Since the prior model should have high probabilities for images that contain the texture we are looking for, the realizations we draw should contain just this texture. What we have to do is in other words to create a probability distribution that describes the texture we want to have in the image. Since it is not always an easy task to say exactly what the texture of an image is, it is also hard to create such a probability distribution.

The common method when creating such a prior model is to use a training image. A training image is for example an image of how the rock types on the surface are distributed, but it could also be an image manufactured by a geologist or some other person that has knowledge about how rocks are typically distributed. Once we have the training image we have to create the prior model that describes the texture in the image. When we draw realizations from this model they should then have the same kind of texture as the training image. Creating such a model is not always an easy task, and in this thesis we are going to look at a general model with the goal that it should be useful for several types of texture.



(a) Sisim (b) Channel (c) Ellipse

Figure 1: The training images.

Markov random fields (Besag 1973) (MRF) have been used a lot in analysis of image textures. The problem with MRF is that it can be computationally expensive to calculate the normalizing constant if the field is large. Therefore there has been a lot of focus on how to calculate this constant efficiently, or how to get a good estimate of it. Another option is to find alternative models. One such model that has shown to capture texture well is the multidimensional Markov chain model (Qian & Titterington 1991) (MDMC). The advantage with MDMC is that we are able to use the forward-backward algorithm (Baum, Petrie, Soules & Weiss 1970) to calculate the normalizing constant even if the field is quite large. In this thesis we are going to use such a MDMC when we create our prior model. The same method was also explored by Nerhus (2008), but we will here use a different kind of parametrization.

The thesis starts by introducing some theory about Markov chains and Markov random fields, before we show how to use the forward-backward algorithm on a one-dimensional MRF. Then we show how we create a parametric multidimensional Markov chain model, and how we estimate model parameters by maximum likelihood from training images. We test our method on the three training images in Figure 1. Image 1(a) is generated by sequential indicator sampling, and we call it Sisim. Image 1(b) is hand-drawn by a geologist, and it show rivers or channels, we call it Channel. Image 1(c) is a realization from a marked Poisson random field, with ellipses as the marks, we call it Ellipse. We finish the text by displaying and discussing some results.

2 Markov chains and parametric models

As the goal of this thesis is to create a parametric model of a multidimensional Markov chain, we will start by saying a few general words about parametric models and Markov chains. Parametric models are simply models that have some parameter $\theta = \{\theta_1, \theta_2, ..., \theta_T\} \in \mathbb{R}^T$. In some cases the parameters will have a physical interpretation, it could for example

be a parameter for wind and one for temperature in a model that describes the weather. In other cases it is harder to say exactly what the parameters mean. In this text we will indicate a parametric model by writing $\pi(x|\theta)$. Note that this does not mean that we condition on the value of θ, as θ is a constant and not a stochastic variable.

If we have a sequence of stochastic variables $x = \{x_1, x_2, ..., x_n\} \in \Lambda^n$, where $x_i \in \Lambda = \{0, 1, ..., L-1\}$, we say that they form a Markov chain if they have the Markov property

$$\pi(x_i|x_{i-1}, x_{i-2}, ..., x_1, \theta) = \pi(x_i|x_{i-1}, x_{i-2}, ..., x_{i-r}, \theta). \tag{1}$$

When the conditional distribution function of x_i given all previous variables is independent of $\{x_1, x_2, ..., x_{i-r-1}\}$, as above, we say that it is an r'th order Markov chain. The full distribution function is given by

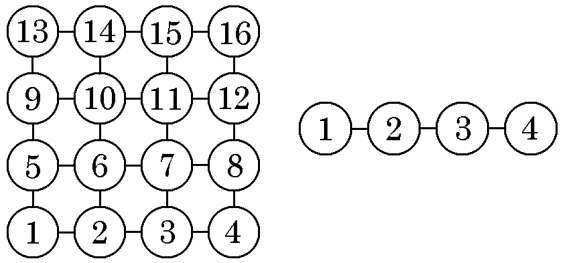
$$\pi(x|\theta) = \pi(x_1|\theta)\pi(x_2|x_1, \theta) \cdots \pi(x_r|x_{r-1}, ..., x_1, \theta) \prod_{i=r+1}^{m} \pi(x_i|x_{i-1}, ..., x_{i-r}, \theta). \tag{2}$$

Markov chains have a lot of applications, and can be found in physics, economics, social sciences, and even in music. The reason is that one can very often assume that a process has the Markov property. In most cases a Markov chain is considered a process in time, where x_i is then the value at time i. We are instead going to look at Markov chains in a spatial setting, where x_i is the value at a position i. A chain like the one above will only make sense if the spatial setting is one-dimensional, as we are then able to talk about 'previous' variables as all variables on one side of x_i. For higher dimensions we instead have something called Markov random fields, where the Markov property occurs in a bit different manner. More on this in the next section.

3 Markov random fields

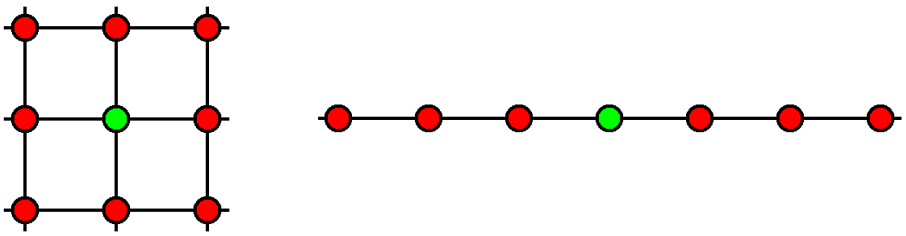
Markov random fields (MRF) are often used to describe texture in images. The image is represented on a grid $\mathcal{S}$=$\{1, 2, ..., n\}$, where the dimension of the grid usually corresponds to the dimension of the image. For images like those in Figure 1 we would use a grid like the one in Figure 2(a), while the grid in Figure 2(b) is better suited for one-dimensional images. These figures also show how the nodes can be numbered, we use $i \in \{1, 2, ..., n\}$ to denote node number i in the grid. Each node will have a value (color), and we let a stochastic variable $x_i \in \Lambda = \{0, 1, ..., L-1\}$ denote the value of node number i. We let $x = \{x_1, x_2, ..., x_n\} \in \Lambda^n$ be a multivariate stochastic variable describing the entire image. If we want the value of several nodes in a subset $C \subseteq \mathcal{S}$ of the grid we use the notation $x_C = \{x_i, i \in C\}$, and for all nodes in $\mathcal{S}$ that are not in C we use $x_{-C} = \{x_i, i \notin C\}$. One such subset that we are going to use later is $\{i, i+1, ..., i+k\}$, this will be denoted by $\{i : i+k\}$, so that $x_{i:i+k} = \{x_i, x_{i+1}, ..., x_{i+k}\}$.

Another subset that we are going to use is what we call a neighborhood. A neighborhood $\delta_i \subseteq \mathcal{S}$ is a subset of the grid around node number i. The set $\delta = \{\delta_1, \delta_2, ..., \delta_n\}$ of



(a) (b)

Figure 2: Example of a two-dimensional grid (a) and a one-dimensional grid (b). The numbers show one way to number the nodes.

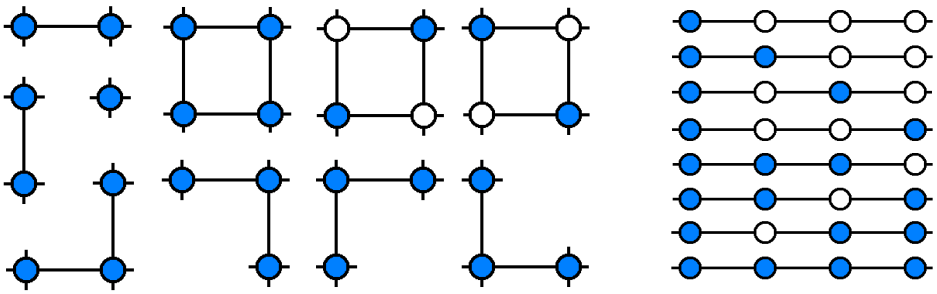


(a) (b)

Figure 3: Some examples of neighborhoods, the red nodes compose the neighborhood of the green node. Note that the green node is not part of its own neighborhood. Figure (a) shows a typical neighborhood in a two-dimensional grid, while Figure (b) is a typical neighborhood in a one-dimensional grid.

the neighborhoods around all nodes in $\mathcal{S}$ is called the neighborhood system. The requirements for a neighborhood system is that a node can not be part of its own neighborhood, $i \notin \delta_i$, and if i is in the neighborhood of j, then j must also be in the neighborhood of i, that is $i \in \delta_j \Leftrightarrow j \in \delta_i$. If $i \in \delta_j$ we say that i and j are neighbors. Figure 3 show some examples of neighborhoods. Neighborhoods can also be empty, in which case the nodes are independent, or the neighborhood can consist of all other nodes in the grid.

Related to neighborhoods we have something called cliques. A clique $c \subseteq \mathcal{S}$ is a subset of the nodes in the grid such that each node in c is neighbor with all other nodes in c, or $i \in \delta_j \forall i, j \in c, i \neq j$. When we talk about cliques we will either be talking about clique types, or just cliques. The difference is that with clique type we do not care about the position of the clique in the grid, only what the clique looks like (i.e. we do not care about translation). We will use the notation $\bar{c}$ when we mean the clique type of the clique c. The set of all cliques in $\mathcal{S}$ is called the clique system $\mathcal{C}$, and we will use $\bar{\mathcal{C}}$ to indicate the set of all different clique types. In most cases a neighborhood will have several different clique types related to it, in Figure 4 we show the different clique types for the neighborhoods in Figure 3.



(a) (b)

Figure 4: The different clique types, up to translation, corresponding to the neighborhoods from Figure 3, the blue nodes are part of the clique, white nodes are not. If we place the green center node of the neighborhood in Figure 3(a) over any of the blue nodes in one of the cliques in (a), all other nodes in the clique should be covered by the red neighborhood. This means that the node is neighbor with all other nodes in the clique. The same is true for the neighborhood in Figure 3(b) and the cliques in (b).

We have a MRF if the value of node number i, given all other nodes in $\mathcal{S}$, only depends on nodes in the neighborhood δ_i, that is if

$$\pi(x_i|x_{-i},\theta) = \pi(x_i|x_{\delta_i},\theta), \qquad \forall i \in \{1,...,n\}. \tag{3}$$

We call this the Markov property. We need to know this conditional distribution function for all nodes i. It can be difficult to create a model from this formulation, and to illustrate why let us rewrite the distribution functions $\pi(x_i|x_{-i},\theta)$ by using Bayes rule, getting

$$\pi(x_i|x_{-i},\theta) = \frac{\pi(x|\theta)}{\sum_{x_i}\pi(x|\theta)}. \tag{4}$$

The notation $\sum_{x_i}$ means to sum over all possible values of the variable x_i. To get a valid model we have to choose the conditional distribution functions $\pi(x_i|x_{-i},\theta)$ such that they are mathematically consistent with the joint distribution function $\pi(x|\theta)$. This can be difficult when there are a lot of variables, as there often are, and it would in other words be better first to find $\pi(x|\theta)$, as we could then calculate the conditional distribution functions afterwards. Thanks to the Hammersley-Clifford theorem (Besag 1973) we are able to do just this if we make one important assumption. We must assume that if $\pi(x_i|\theta) > 0$ for all x_i, then $\pi(x|\theta) > 0$ for all x (all values of x are possible). This is called the positivity condition. Under this assumption the theorem states that we have a MRF on the form (3), with respect to the neighborhood system δ, if and only if the joint distribution function is given by

$$\pi(x|\theta) = \frac{\exp\left\{-\sum_{c\in\mathcal{C}} V_c(x_c|\theta)\right\}}{z(\theta)}, \tag{5}$$

where $\mathcal{C}$ is the clique system corresponding to δ. The sum in the exponent is over all cliques c in the clique system $\mathcal{C}$. The functions $V_c(x_c|\theta)$ are called potential functions, as in some physical cases they correspond to potential energy. The normalizing constant $z(\theta)$ is given by

$$z(\theta) = \sum_x \exp\left\{-\sum_{c\in\mathcal{C}} V_c(x_c|\theta)\right\},$$

and is generally unattainable, as $x \in \Lambda^n$ and therefore has L^n possible values. For the images in Figure 1, where $L = 2$ and $n = 125 \times 125$, this would be a sum with about 10^{4700} terms!

It is quite easy to define $\pi(x|\theta)$, all we have to do is to define the cliques and the potential functions. Once this is done we can use (4) to find the conditional distribution functions. It should be noted that

$$\begin{aligned}\pi(x_i|x_{-i},\theta) \propto \pi(x|\theta) &= \exp\left\{-\sum_{c\in\mathcal{C}} V_c(x_c|\theta)\right\} \\ &= \exp\left\{-\sum_{c\in\mathcal{C}:i\in c} V_c(x_c|\theta) - \sum_{c\in\mathcal{C}:i\notin c} V_c(x_c|\theta)\right\} \\ &\propto \exp\left\{-\sum_{c\in\mathcal{C}:i\in c} V_c(x_c|\theta)\right\},\end{aligned}$$

where the sum in the exponent is over all cliques that contain i, since all potential functions of cliques that does not contain i will be constants. The cliques containing i must by definition only contain nodes that are in the neighborhood of i, so the function will only depend on x_{δ_i}, fulfilling (3).

3.1 Defining a MRF by defining potential functions

Defining a MRF means that we have to define the potential functions. We want a parametric probability distribution, so the potential functions should contain a parameter θ. The potential function should also focus on the fact that we wish to capture the texture in the image, so we must identify some way to do this. We are going to focus on when a set of nodes has the same value. If we for different sets of nodes can say something about how often the set should have the same value, for all of the possible values, then we should also be able to say something about the texture in the entire image when we combine this information. As we are working with MRF it is obvious to choose the different clique types as the sets of nodes to look at, and the parameter to give us information about the color, so in this text we will use potential functions on the form

$$V_c(x_c|\theta) = \sum_{l\in\Lambda} \theta_{\bar{c},l} I(x_i = l; \forall i \in c), \tag{6}$$

where

$$I(x_i = l; \forall i \in c) = \begin{cases} 1 & \text{if} \quad x_i = l; \forall i \in c, \\ 0 & \text{else.} \end{cases}$$

This means that if all nodes in the clique c have the same value l, the potential function will return the parameter $\theta_{\bar{c},l}$. Each parameter can then be interpreted as the potential for a clique type to have a certain value on all nodes. Having this information for all clique types and all colors will hopefully be enough to capture the texture in an image.

3.2 One-dimensional MRF

We are now going to look at a one-dimensional MRF where the neighborhood is k nodes on each side of i, or $\delta_i = \{i-k : i-1, i+1 : i+k\}$. Figure 3(b) shows an example where $k = 3$, and Figure 4(b) show the different clique types corresponding to this neighborhood. Note that such a neighborhood will have 2^k different clique types. To see this we refer to Figure 4(b), where we see that the leftmost node is always part of the clique (blue), while the each of the last $k = 3$ nodes are either part of the clique (blue) or not part of the clique (white). To get all different clique types, all combinations of blue/white must be used on these last nodes. Thus there are 3 nodes where each has two possible 'values', and we end up with $2^3 = 8$ different clique types.

We are later going to use a one-dimensional MRF based on such a neighborhood as part of a multidimensional Markov chain model. This means that we must show that the MRF can be formulated as a Markov chain. To show this we need to show that it has the Markov property (1), which we can show by finding the distribution functions $\pi(x_i|x_{1:i-1}, \theta)$. Using Bayes rule we get

$$\pi(x_i|x_{1:i-1}, \theta) = \frac{\pi(x_{1:i}|\theta)}{\pi(x_{1:i-1}|\theta)} \propto \pi(x_{1:i}|\theta) = \sum_{x_{i+1:n}} \pi(x|\theta) = \sum_{x_{i+1:n}} \exp\left\{ -\sum_{c \in \mathcal{C}} V_c(x_c|\theta) \right\}. \tag{7}$$

Let us now move all exponentials with potential functions of cliques that does not contain any of the nodes $\{i : n\}$ outside the sum. We then have to separate all cliques into two sets, one set $C_{-i:n} = \{c \in \mathcal{C} : c \bigcap \{i : n\} = \varnothing\}$ with all cliques that does not contain any of the nodes $\{i : n\}$, and one set $C_{i:n} = \mathcal{C} \setminus C_{-i:n}$ with all other cliques. We can then rewrite (7) by

$$\begin{aligned} \pi(x_i|x_{1:i-1}, \theta) &\propto \sum_{x_{i+1:n}} \exp\left\{ -\sum_{c \in C_{-i:n}} V_c(x_c|\theta) - \sum_{c \in C_{i:n}} V_c(x_c|\theta) \right\} \\ &= \exp\left\{ -\sum_{c \in C_{-i:n}} V_c(x_c|\theta) \right\} \sum_{x_{i+1:n}} \exp\left\{ -\sum_{c \in C_{i:n}} V_c(x_c|\theta) \right\} \\ &\propto \sum_{x_{i+1:n}} \exp\left\{ -\sum_{c \in C_{i:n}} V_c(x_c|\theta) \right\}. \end{aligned}$$

What we have left now in the exponent is a sum over all cliques that contain at least one of the nodes $\{i : n\}$. Some of these cliques will also contain some of the nodes $\{i-k : i-1\}$, since these nodes are in the neighborhood of i, but none will contain any of the nodes $\{1 : i-k-1\}$. This means that x_i given $x_{1:i-1}$ is independent of $x_{1:i-k-1}$, which is the same as the Markov property of a k'th order Markov chain.

4 Forward-backward for one-dimensional MRF

The forward-backward algorithm (Baum et al. 1970) is a recursive method that lets us calculate the normalizing constant of certain probability functions, called general factorisable models by Reeves & Pettitt (2004). Once we are able to calculate the normalizing constant we can also calculate likelihoods, and we are able to draw samples in a very efficient manner (Friel & Rue 2007). We are now going to show how the method can be derived for a one-dimensional MRF like the one described in Section 3.2.

The algorithm iterates through the grid one node at the time, and to derive it we need to formulate the distribution function (5) by

$$\pi(x|\theta) = \frac{\exp\left\{-\sum_{i=1}^{n-k} g_i(x_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(x_{i:n}|\theta)\right\}}{z(\theta)}, \tag{8}$$

where

$$\begin{aligned} g_i(x_{i:i+k}|\theta) &= \sum_{c\in C_i} V_c(x_c|\theta), \qquad i \in \{1 : n-k\}, \\ g_i(x_{i:n}|\theta) &= \sum_{c\in C_i} V_c(x_c|\theta), \qquad i \in \{n-k+1 : n\}, \end{aligned}$$

and here the set $C_i \subset \mathcal{C}$ is defined by $C_i = \{c \in \mathcal{C} : i \in c, c\bigcap\{1 : i-1\} = \varnothing\}$. This means that it is the set of all cliques that contain i but none of the nodes $\{1 : i-1\}$. If $k = 3$ this would be one each of the types of cliques shown in Figure 4(b), and the leftmost node in the clique would be node number i. We are still summing over all cliques $c \in \mathcal{C}$, the only difference from (5) is that we have selected the order in which to do the sum. It will be clear why we want it on this form when we now show how to derive the forward-backward algorithm. The first part of the algorithm, the forward iterations, let us calculate the normalizing constant. The backward iterations can be used to draw a sample from the distribution function.

4.1 Calculating the normalizing constant

The distribution function for the MRF is given by (8). Let us now define

$$h(x|\theta) = \exp\left\{-\sum_{i=1}^{n-k} g_i(x_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(x_{i:n}|\theta)\right\}.$$

The normalizing constant is then given by

$$z(\theta) = \sum_x h(x|\theta) = \sum_{x_1} \cdots \sum_{x_n} h(x|\theta).$$

It is very computational expensive to calculate this directly since x has L^n possible values. What we instead do is that we calculate it recursively by taking one of the sums $\sum_{x_i}$ at the time. Let us see how and why this works by doing it step by step. We start by summing out x_1 from $h(x, \theta)$

$$\begin{aligned} h(x_{2:n}|\theta) &= \sum_{x_1} h(x|\theta) = \sum_{x_1} \exp\left\{ -\sum_{i=1}^{n-k} g_i(x_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(x_{i:n}|\theta) \right\} \\ &= \exp\left\{ -\sum_{i=2}^{n-k} g_i(x_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(x_{i:n}|\theta) \right\} \sum_{x_1} \exp\left\{ -g_1(x_{1:1+k}|\theta) \right\}. \end{aligned}$$

Here we use the notation $h(x_{2:m}, \theta)$ to indicate that x_1 is not part of the variable after we have summed it out. On the second line we moved all terms not containing x_1 outside the sum. The important thing to note here is that the sum over x_1 on the second line now can be described as a function of the variable $x_{2:1+k}$, we denote this function by

$$z_1(x_{2:1+k}|\theta) = \sum_{x_1} \exp\left\{ -g_1(x_{1:1+k}|\theta) \right\},$$

ending up with

$$h(x_{2:m}|\theta) = \exp\left\{ -\sum_{i=2}^{m-k} g_i(x_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(x_{i:n}|\theta) \right\} z_1(x_{2:1+k}|\theta).$$

We call $z_1(x_{2:1+k}|\theta)$ the first forward variable. It has L^k possible values, one for each value of $x_{2:1+k}$. If we calculate all of these values we can use them in the next iteration when we sum out x_2 from $h(x_{2:n}|\theta)$

$$\begin{aligned} h(x_{3:n}|\theta) &= \sum_{x_2} h(x_{2:n}|\theta) = \sum_{x_2} \left[\exp\left\{ -\sum_{i=2}^{n-k} g_i(x_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(x_{i:n}|\theta) \right\} z_1(x_{2:1+k}|\theta) \right] \\ &= \exp\left\{ -\sum_{i=3}^{n-k} g_i(x_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(x_{i:n}|\theta) \right\} \sum_{x_2} \left[\exp\left\{ -g_2(x_{2:2+k}|\theta) \right\} z_1(x_{2:1+k}|\theta) \right] \\ &= \exp\left\{ -\sum_{i=3}^{n-k} g_i(x_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(x_{i:n}|\theta) \right\} z_2(x_{3:2+k}|\theta). \end{aligned}$$

We see that the second forward variable is given by

$$z_2(x_{3:2+k}|\theta) = \sum_{x_2} \left[\exp\left\{ -g_2(x_{2:2+k}|\theta) \right\} z_1(x_{2:1+k}|\theta) \right].$$

Repeating this possess we get that forward variable number i is given by

$$z_i(x_{i+1:i+k}|\theta) = \sum_{x_i} \left[\exp\left\{-g_i(x_{i:i+k}|\theta)\right\} z_{i-1}(x_{i:i-1+k}|\theta)\right], \quad i \in \{2 : n-k\},$$

and that

$$h(x_{i:n}|\theta) = \exp\left\{-\sum_{j=i}^{n-k} g_j(x_{j:j+k}|\theta) - \sum_{j=n-k+1}^{n} g_j(x_{j:n}|\theta)\right\} z_{i-1}(x_{i:i+k-1}|\theta),\ i \in \{2 : n-k\}. \tag{9}$$

For the last few iterations the forward variable is given by

$$z_i(x_{i+1:n}|\theta) = \sum_{x_i} \left[\exp\left\{-g_i(x_{i:n}|\theta)\right\} z_{i-1}(x_{i:n}|\theta)\right],\ \ i \in \{n-k+1 : n\},$$

and

$$h(x_{i:n}|\theta) = \exp\left\{-\sum_{j=i}^{n} g_j(x_{j:n}|\theta)\right\} z_{i-1}(x_{i:n}|\theta), \quad i \in \{n-k+1 : n\}. \tag{10}$$

The last forward variable will then be equal to the normalizing constant, since we have summed out all x_i from $h(x|\theta)$

$$z(\theta) = z_n(\theta). \tag{11}$$

Since we in each iteration need to do the L^k different calculations of the forward variables, and each is a sum with L terms, we end up with a total of $\mathcal{O}(nL^{k+1})$ calculations. Thus we are restricted mostly by the size k of our neighborhood, but also by how many values L each node can have. Remember that each value is intended to represent a rock type, and typically there are not a lot of different rock types, so in most cases k is the main limiting factor.

4.2 Likelihood

We can find the likelihood of a parameter θ given a realization y of the MRF by simply calculating

$$L(\theta|y) = \frac{1}{z(\theta)} \exp\left\{-\sum_{i=1}^{n-k} g_i(y_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(y_{i:n}|\theta)\right\}, \tag{12}$$

where $z(\theta)$ can be calculated as above. This can for example be used in maximum likelihood estimation of the parameter θ.

4.3 Sampling by backward iterations

To draw a sample from the MRF we can start by drawing a sample of x_n, and then iterate backwards while drawing samples depending on what we already drew. Drawing

a sample of x_n means that we need the marginal probability $\pi(x_n|\theta)$, which we find by summing out all other variables $x_{1:n-1}$ from $\pi(x|\theta)$. This gives us

$$\pi(x_n|\theta) = \sum_{x_{1:n-1}} \pi(x|\theta) = \sum_{x_{1:n-1}} \frac{h(x|\theta)}{z(\theta)} = \frac{h(x_n|\theta)}{z(\theta)}.$$

Inserting (10) into this, with $i = n$, we get

$$\pi(x_n|\theta) = \frac{\exp\{-g_n(x_n|\theta)\}\, z_{n-1}(x_n|\theta)}{z_n(\theta)}.$$

For the next steps we need the conditional distribution function

$$\pi(x_i|x_{i+1:n}, \theta) = \frac{\pi(x_{i:n}|\theta)}{\pi(x_{i+1:n}|\theta)} = \frac{h(x_{i:n}|\theta)}{h(x_{i+1:n}|\theta)}.$$

We have $h(x_{i:n}|\theta)$ given from either (9) or (10), depending on the value of i. If $n - k < i < n$ we get

$$\begin{aligned}\pi(x_i|x_{i+1:n}, \theta) &= \frac{\exp\left\{-\sum_{j=i}^{n} g_j(x_{j:n}|\theta)\right\} z_{i-1}(x_{i:n}|\theta)}{\exp\left\{-\sum_{j=i+1}^{n} g_j(x_{j:n}|\theta)\right\} z_i(x_{i+1:n}|\theta)} \\ &= \frac{\exp\{-g_i(x_{i:n}|\theta)\}\, z_{i-1}(x_{i:n}|\theta)}{z_i(x_{i+1:n}|\theta)}.\end{aligned}$$

If $1 < i \leq n - k$ we will have

$$\begin{aligned}\pi(x_i|x_{i+1:n}, \theta) &= \frac{\exp\left\{-\sum_{j=i}^{n-k} g_j(x_{j:j+k}|\theta) - \sum_{j=n-k+1}^{n} g_j(x_{j:n}|\theta)\right\} z_{i-1}(x_{i:i+k-1}|\theta)}{\exp\left\{-\sum_{j=i+1}^{n-k} g_j(x_{j:j+k}|\theta) - \sum_{j=n-k+1}^{n} g_j(x_{j:n}|\theta)\right\} z_i(x_{i+1:i+k}, \theta)} \\ &= \frac{\exp\{-g_i(x_{i:i+k}|\theta)\}\, z_{i-1}(x_{i:i+k-1}|\theta)}{z_i(x_{i+1:i+k}|\theta)}.\end{aligned}$$

Finally, for $i = 1$ we get

$$\pi(x_1|x_{2:m}, \theta) = \frac{\exp\{-g_1(x_{1:1+k}|\theta)\}}{z_1(x_{2:1+k}|\theta)}.$$

Since each of these probability functions only have L possible values, $\pi(x_i = l|x_{i+1:n}, \theta)$ for $l = \{0 : L - 1\}$, we are able to calculate the probability of each, and it is easy to draw a sample from them. In the worst case we have to do L computations at each step, which means $\mathcal{O}(nL)$ computations total. Since we use the forward variables during the sampling procedure, we have to store them during the forward iterations. This means that we must be able to store nL^k values.

4.4 Logarithmic version

In many cases we will get numerical problems if we try to compute the forward variables as shown above. This is due to the fact that the probability $\pi(x|\theta)$ will be very small when x has a large amount of possible values. In these cases we can instead store the forward variables on logarithmic scale, as this helps us avoid the numerical problems. We will now show how this can be done. We start with the first forward variable, and define

$$\begin{aligned} \upsilon_1(x_{2:1+k}|\theta) &= \log(z_1(x_{2:1+k}|\theta)), \\ \Rightarrow z_1(x_{2:1+k}|\theta) &= \mathrm{e}^{\upsilon_1(x_{2:1+k}|\theta)}. \end{aligned}$$

For the other forward variables we get

$$\begin{aligned} z_i(x_{i+1:i+k}|\theta) &= \sum_{x_i} \mathrm{e}^{-g_i(x_{i:i+k}|\theta)+\upsilon_{i-1}(x_{i:i-1+k}|\theta)} \quad \text{if} \quad 1 < i \leq n-k, \\ \Rightarrow \upsilon_i(x_{i+1:i+k}|\theta) &= \log\left(\sum_{x_i} \mathrm{e}^{-g_i(x_{i:i+k}|\theta)+\upsilon_{i-1}(x_{i:i-1+k}|\theta)}\right) \quad \text{if} \quad 1 < i \leq n-k, \end{aligned}$$

and

$$\begin{aligned} z_i(x_{i+1:n}|\theta) &= \sum_{x_i} \mathrm{e}^{-g_i(x_{i:n}|\theta)+\upsilon_{i-1}(x_{i:n}|\theta)} \quad \text{if} \quad n-k < i \leq n, \\ \Rightarrow \upsilon_i(x_{i+1:n}|\theta) &= \log\left(\sum_{x_i} \mathrm{e}^{-g_i(x_{i:n}|\theta)+\upsilon_{i-1}(x_{i:n}|\theta)}\right) \quad \text{if} \quad n-k < i \leq n. \end{aligned}$$

To calculate $\upsilon_i(x_{i+1:i+k}|\theta)$ without getting numerical problems we can use a simple trick. If we in the exponent add and subtract a constant q we get

$$\log\left(\sum_{x_i} \mathrm{e}^{-g_i(x_{i:i+k}|\theta)+\upsilon_i(x_{i:i-1+k}|\theta)+q-q}\right) = q + \log\left(\sum_{x_i} \mathrm{e}^{-g_i(x_{i:i+k}|\theta)+\upsilon_i(x_{i:i-1+k}|\theta)-q}\right).$$

Now if we choose $q = \max_{x_i}(-g_i(x_{i:i+k}|\theta) + \upsilon_{i-1}(x_{i:i-1+k}|\theta))$ we will end up with

$$\upsilon_i(x_{i+1:i+k}|\theta) = q + \log(\mathrm{e}^0 + \mathrm{e}^Q) = q + \log(1 + \mathrm{e}^Q),$$

where we have $Q = \min_{x_i}(-g_i(x_{i:i+k}|\theta)+\upsilon_{i-1}(x_{i:i-1+k}|\theta))-q$. This means that we always will have $Q \leq 0$. It is easy to substitute these variables into the sampling procedure, depending on i we get

$$\begin{aligned} \pi(x_n|\theta) &= \exp\left\{-g_n(x_n|\theta) + \upsilon_{n-1}(x_n|\theta) - \upsilon_n(\theta)\right\}, \\ \pi(x_i|x_{i+1:n},\theta) &= \exp\left\{-g_i(x_{i:n}|\theta) + \upsilon_{i-1}(x_{i:n}|\theta) - \upsilon_i(x_{i+1:n}|\theta)\right\}, \quad n-k < i < n, \\ \pi(x_i|x_{i+1:n},\theta) &= \exp\left\{-g_i(x_{i:i+k-1}|\theta) + \upsilon_{i-1}(x_{i:i+k-1}|\theta) - \upsilon_i(x_{i+1:i+k}|\theta)\right\}, \quad 1 < i \leq n-k, \\ \pi(x_1|x_{2:n},\theta) &= \exp\left\{-g_1(x_{1:1+k}|\theta) - \upsilon_1(x_{2:1+k}|\theta)\right\}. \end{aligned}$$

For the likelihood we will get

$$L(\theta|y) = \exp\left\{-\sum_{i=1}^{n-k} g_i(y_{i:i+k}|\theta) - \sum_{i=n-k+1}^{n} g_i(y_{i:n}|\theta) - \upsilon_n(\theta)\right\}. \tag{13}$$

This version of the forward-backward algorithm is very useful.

5 The multidimensional Markov chain model

A multidimensional Markov chain (MDMC) is, as the name might reveal, a Markov chain where each element of the chain is also a Markov chain. This means that each conditional distribution function (1) is a Markov chain, with x_i as a set of stochastic variables. Generally this can be done any number og times, but we are only going to look at the two-dimensional case. Our motivation is to use a MDMC model to describe a two-dimensional image. Qian & Titterington (1991) showed that this can be used to capture texture in much the same way as a two-dimensional MRF. We let the image x be represented on a two-dimensional grid $\mathcal{S}$ of size $m \times n$, and use $x_{ij} \in \Lambda$ to denote the value of node number j from the left on row number i from the top, which means that $i \in \{1, ..., m\}$ and $j \in \{1, ..., n\}$. We then let $x_i = \{x_{i1}, ..., x_{in}\} \in \Lambda^n$ represent a row, while $x = \{x_1, ..., x_m\} \in \Lambda^{mn}$ represents the entire image.

We let the conditional probability function for each of the rows, given the previous rows, have the Markov property (1), by letting it depend only on the r immediately preceding rows. The full distribution function of the image is then given by (2). For each row we will use a one-dimensional MRF based on a neighborhood like the one described in 3.2, except that we will also include information from the previous rows. Since a one-dimensional MRF can be formulated as a Markov chain, as described in Section 3.2, we now have a two-dimensional Markov chain. Figure 5 illustrates how the model works.

5.1 Parametrization

We said that each row will condition on the r previous rows. The information from the previous rows must then be added into the one-dimensional MRF. To make sure we get a valid model we must add the information into the potential functions. We have a lot of options for how we want to do this, and how good our model becomes could possibly depend a lot on how we do it. We choose to use potential functions on the form (6). By using this formulation we can say that we ask the following question: Do the nodes in the clique have the same value, and if they do, what value is it? The answer to the question is given by what parameter is returned, as each parameter $\theta_{\bar{c},l}$ represents that a type of clique $\bar{c}$ has the same value l on all nodes. The value of the parameter then represents the potential for this value on all nodes. When we now include information from the previous rows it is natural to extend this question, so that it not only includes

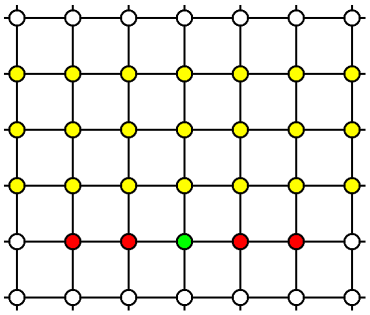


Figure 5: This figure illustrates how the MDMC model works. The distribution of each row, given all previous rows, is a one-dimensional MRF that conditions only on r previous rows (yellow). The MRF itself is defined by a neighborhood system where the neighborhood of a node (green) is the k closest nodes on each side (red). In this example $r = 3$ and $k = 2$.

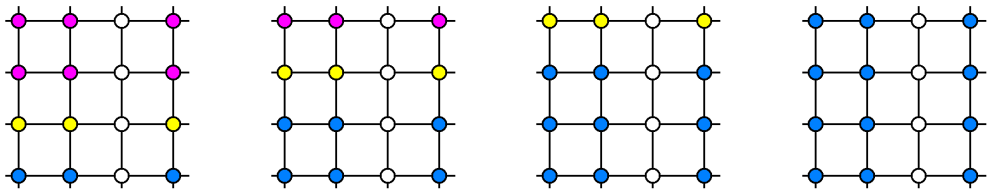
the nodes in the clique, but also nodes from the previous rows. The question we are going to ask is: If the nodes in the clique have the same value, then how many rows backward can we trace the same clique still having the same value? To get the answer we again use the parameters, by letting the parameter that is returned also represent how many rows backward we can trace the clique with the same value. One parameter will then represent that we can go no rows backward, another will represent that we can go one row backwards (but not more), and so on, while also taking into account the clique type and the color of the nodes. As we only condition on r rows, we will stop counting if we get to the r'th row. Mathematically we can write

$$V_c(x_{i,c}|\theta) = \begin{cases} \theta_{\bar{c},l,\rho} & \text{if} \quad x_{ij} = l \forall i,j \in \{i-\rho : i, c\}, \\ 0 & \text{else}, \end{cases}$$

where ρ is the smallest integer that satisfy $x_{i-\rho-1,j} \neq l$ for $j \in c$ and $0 \leq \rho \leq r$. Figure 6 gives an example of how this parametrization works.

A question now is, how many parameters does such a model give us? Well, we have 2^k different clique types $\bar{c}$, L different values for l, $r+1$ possible values for ρ, and this gives us $T = (r+1)L2^k$ parameters. If we want to use large values for k, r or L we might need to reduce the number of parameters somehow. One way to do this is to ignore some of the clique types. We can for example ignore all clique types that contain more than d nodes. This should not hurt the model too much, as large cliques are somewhat redundant since they can be separated into sets of smaller cliques. As an example of this we again use Figure 4(b), where we see that the first seven cliques are all a subset of the last and largest clique. The number of different clique types that we are not ignoring is then given by

$$\eta_{k,d} = \sum_{s=0}^{d-1} \frac{k!}{s!(k-s)!}. \tag{14}$$



(a) (b) (c) (d)

Figure 6: These figures illustrate when we get different parameters $\theta_{\bar{c},l,\rho}$ for a clique c depending on the $r = 3$ previous rows. The clique is the blue nodes in the lowest row, we assume that all nodes in this clique have the same value l (otherwise no parameter is returned). In (a) at least one of the yellow nodes has a different value than l, which means we can go $\rho = 0$ rows backward. The value of the pink nodes are irrelevant. In (b) the blue nodes on the second lowest row also have the value l, while at least one of the yellow nodes above has a different value. This means we are able to go $\rho = 1$ rows backward. The pink nodes are again irrelevant. Also in (c) the blue nodes have the value l, while at least one of the yellow nodes has a different value. Thus we get $\rho = 2$. For (d) all blue nodes have the value l, and $\rho = r = 3$. In all four figures the white nodes are not part of the clique, and thus their value is irrelevant.

To show how we get to this result we once again refer to Figure 4(b). We see that there are three cliques with two nodes, one for each way that the right blue node can be combined with two white nodes (two white nodes since one blue plus two white means total three nodes, which is the size of k). There will also be three cliques with three nodes, as we get one for each way that the two right blue nodes can be combined with one white node. There are one each of the other cliques, as three white nodes can only be combined in one way with zero blue nodes, and visa versa. In other words, if the cliques shall have $s+1$ nodes total, we will get one clique for each way s blue nodes can be combined with $k - s$ white nodes. This number is given by

$$\frac{(s + (k - s))!}{s!(k - s)!} = \frac{k!}{s!(k - s)!}.$$

Adding this up for all clique types where $s + 1 \leq d$ gives us (14). The number of parameters are then $T = (r + 1)L\eta_{k,d}$.

5.2 Parameter estimation by maximum likelihood

To calculate the likelihood of a parameter θ given a training image y we use the forward backward algorithm on one row at the time to calculate (12), and then we use

$$L(\theta|y) = L(\theta|y_1) \cdots L(\theta|y_r, y^*_{1:r-1}) \prod_{i=r+1}^{m} L(\theta|y_i, y^*_{i-r:i-1}),$$

where a * indicates the rows that are not part of the one-dimensional MRF. We want the value of θ that maximizes $L(\theta|y)$, and to estimate this value we use a built in optimization

routine from R called *nlm*. *Nlm* is short for *non linear minimization*, and uses a Newton-like algorithm. Since it is a minimization method we instead have to find the value of θ that minimizes $-L(\theta|y)$, this is of course the same value that maximizes $L(\theta|y)$. To be able to calculate the likelihood we must use logarithms, otherwise the values are just too small/big and we get numerical problems. Logarithms will work since the value of θ that maximizes $L(\theta|y)$ will also maximize $\log(L(\theta|y))$. Using logarithms we get

$$\log(L(\theta|y)) = \log(L(\theta|y_1)) + \cdots + \log(L(\theta|y_r, y^*_{1:r-1})) + \sum_{i=r+1}^{m} \log(L(\theta|y_i, y^*_{i-r:i-1})).$$

We have in Section 4.4 already explained how $L(\theta|y_i, y^*_{i-r:i-1})$ can be calculated by a logarithmic form of the forward-backward algorithm. If we take the logarithm of Equation (13) we get

$$\log(L(\theta|y_i, y^*_{i-r:i-1})) = -\sum_{j=1}^{n-k} g_j(y_{i,j:j+k}|y^*_{i-r:i-1}, \theta) - \sum_{j=n-k+1}^{n} g_j(y_{i,j:n}|y^*_{i-r:i-1}, \theta) - \upsilon_n(\theta).$$

Thus what we do is that we just sum over the potential functions for the entire row, and then subtract the normalizing constant.

6 Results

We are now going to see if this method is able to capture the texture in the three training images from Figure 1. Since these images only have two different colors we have $x_i \in \{0, 1\}$ and thus $L = 2$. The method is defined by k, r and d, which is the size of the neighborhood, the number of rows we condition on, and the largest number of nodes we use in our cliques. After specifying a value for these we run the optimization routine to get the estimated value of θ that maximizes $L(\theta|y)$, where y is one of the training images in Figure 1. We can then use this value of θ to draw samples from $\pi(x|\theta)$ by the forward-backward algorithm. The training images have a size of 125×125 pixels, and the realizations we create will have the same size. To get rid of any potential edge errors we first create a realization of size 145×145, and then remove a border of 10 pixels.

The realizations must then be evaluated to see if they contain the same texture as the training image. One way to do this is to just look at the realizations. In some cases this is very efficient, especially when the realizations are obviously wrong, but in some cases it can be hard to say if the images have the same texture. It is then better to create some tests that look for properties in the images, and see if the realizations have the same value on these properties as the training image. The properties we have looked

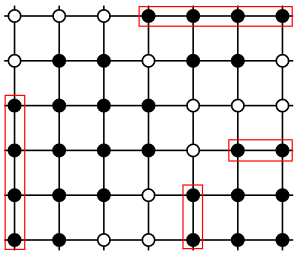


Figure 7: This image is used to illustrate two things. 1) The red rectangles show horizontal and vertical pairs and quadruplets. If all nodes inside the rectangle are black we say that it is a black pair/quadruplet. To calculate for example vp we simply count how many vertical black pairs there are in the image, and divide by the total number of vertical pairs. 2) The image has three black objects. From any node inside an object we can reach all other nodes in the object by going only up, down, left or right, without having to cross any white nodes. The image has four white objects if we count them in a similar manner.

at are

pb :	Percent of the nodes in the image that are black.
vp :	Percent of vertical pairs that are black.
hp :	Percent of horizontal pairs that are black.
vq :	Percent of vertical quadruplets that are black.
hq :	Percent of horizontal quadruplets that are black.
o :	Number of black objects.
mho :	Mean height of black objects.
mwo :	Mean width of black objects.
lo :	Size of the largest black object.

The first five of these are all local properties, they look at just a few nodes at the time and give the percentages of how often the property occurs in the image. A vertical pair is a set of two nodes where one is right above or under the other. For horizontal pairs the nodes are right next to each other on the same row. Horizontal and vertical quadruplets are the same thing, but with four nodes. Figure 7 illustrates what we mean.

The last four properties are more global, where large sets of nodes (objects) are considered at once. A black object is defined as a set of nodes where you from any node can reach any of the other nodes in the object by going only up, down, left or right without having to cross any white nodes. In Figure 7 there are three black objects. The hight and width of an object is simply the number of nodes from the top to the bottom and from the left side to the right side of the object. The size of the object is the number of nodes in it.

It should be noted that even if the realizations get the same values on these properties as the training image does, we still can not conclude that it is a perfect realization of the texture, since there are endless other properties that we have not tested, and many

	Sisim	Channel	Ellipse
pb	0.333	0.277	0.295
vp	0.297	0.252	0.280
hp	0.238	0.214	0.232
vq	0.265	0.228	0.266
hq	0.170	0.150	0.169
o	83	3	36
mho	13.78	112.0	25.33
mwo	4.614	40.0	6.361
lo	444	2352	618

Table 1: The training images scores at the nine properties. The first five local properties are shown as the percent of the entire image. Realizations from the models we create will be tested and compared to these numbers.

might give different values for the realizations than for the training image. However the properties we have chosen should at least give us an indication of how well the method works.

We expect that we will get better results for the local properties than for the global properties, since the model is very locally defined with the neighborhoods and the cliques. It might also be differences in the vertical and horizontal properties, since the model is separated into a set of horizontal MRF. If there are any difference we expect the we are better at capturing the horizontal than the vertical properties. Another thing that will be interesting to study is how hq is effected when we use cliques with less than 4 nodes. The results should be better for this property if we use cliques with 4 or more nodes.

When we compare realizations with the training image it is not enough to just look at a few realizations, so what we do is that we create 500 realizations and test all of them. We can then make histograms displaying how the scores from the realizations are distributed, and we can compare this to the score of the training image. Table 1 show what scores the training images have on the nine properties.

We separate the results into three parts, one for each training image. For each image we try thirteen models, as shown in Table 2. In the first four models we change k while r and d are unchanged. Then we change d and keep the others unchanged in the next four models, before we have four models where we change r. In the last model we see what happens if we use the columns, instead of the rows, as the one-dimensional MRF. In practice this means that we just rotate the training image 90 degrees before we run the optimization. Note that model 3 and 6, and model 8 and 11, are actually the same. This is because we want to see how k, r and d effect the results, and this is easier to see with such a setup. We will therefore refer to model 3/6 or 8/11 if we comment on these.

Model	k	r	d	T
1	2	4	3	40
2	3	4	3	70
3	4	4	3	110
4	5	4	3	160
5	4	4	2	50
6	4	4	3	110
7	4	4	4	150
8	4	4	5	160
9	4	2	5	96
10	4	3	5	128
11	4	4	5	160
12	4	5	5	192
13	4	4	5	160

Table 2: The different models we try for the training images. In models number 1-4 we change k, the size of the neighborhood, while in models number 5-8 we change the size d of the largest cliques we use. In models number 9-12 we change the number of rows r that each of the one-dimensional MRF condition on. The last model is the same as model number 8 and 11, but we have used the columns for the one-dimensional MRF instead of the rows (rotating the training image 90 degrees before we run the optimization). The models has T parameters.

6.1 Sisim

The sisim image is different from the two other training images since it is a bit more unclear, and it is not easy to say exactly what the texture in the image is. This means that it is probably the easiest texture to get decent realizations from, since it will not matter too much if the realizations have some small errors.

Figure 8 show how the models perform on the property tests compared to the training image. We see that there are few large changes when we change k, r or d, and this is probably due to the inherent noise in the images. Only the fifth model, where we use $d = 2$, and the thirteenth model where we use the columns as the one-dimensional MRF, is really different from the others. In general it seems like the small local properties *pb*, *vp* and *hp* fit well with the training image, while for the larger local properties *vq* and *hq*, in addition to the global properties, the values seem to be either to high or to low. The number of objects seems to be too large, and this is of course also linked to the fact that the width and the height of the objects are to small. If we take a look at realizations from some of the models, shown in Figure 9, we see that in most realizations there are too many small objects of only a few pixels in size, and this explains why there are too many objects and their average size is too small. The larger objects seem mostly to have the right shape and size, except that some of them are too large, something that is also seen in the *lo* test, where we see that the largest objects most of the time are larger than in the training image. It does not look like there are any large differences in the performance of the vertical and the horizontal properties, except that *hq* seems to be a little better than *vq* compared to the training image.

When we increase k in models number 1-4 it seems like the width of the objects improves a little, and this should be expected. What is not expected is that also the height seems to improve a little. This is probably related to the fact that the number of objects decreases, we probably get fewer of the smallest objects, and thus the average size gets a little larger. It might also look like there is a very small trend in all the local properties, where the value of the properties decreases a bit as k increases, and this indicates that there are less black in the images. It does not look like the largest objects are effected very much by the change of k. In general it seems like we want large neighborhoods.

In models number 5-8 we change d, and even if this does not seem to have a very strong effect on the local properties, it is clear in the global properties that we want to use cliques with more than $d = 2$ nodes. Realizations from model number 5 are shown in Figure 9(b), and compared to the other realizations we see that these seem to have more objects of only a few pixels in size. When we use cliques with more nodes this problem seems to disappear.

In the next four models we see that changing r will not have a very strong effect, especially not for the local properties. We would expect that the height of the objects improve, and it seems that it does so a little, but not very much. In fact the width of

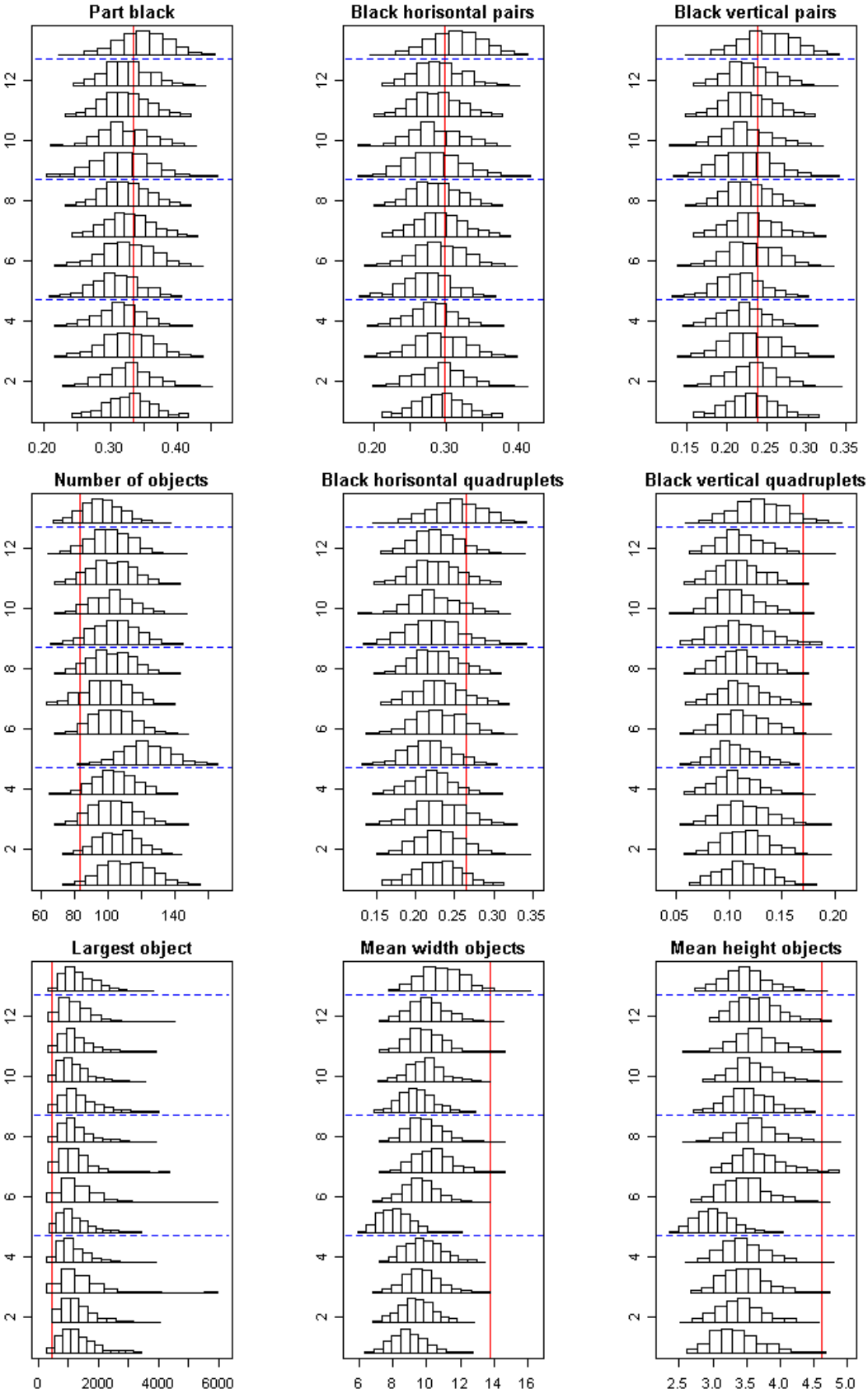
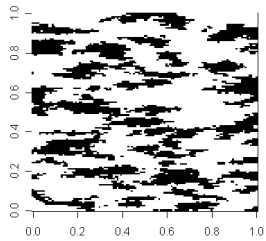
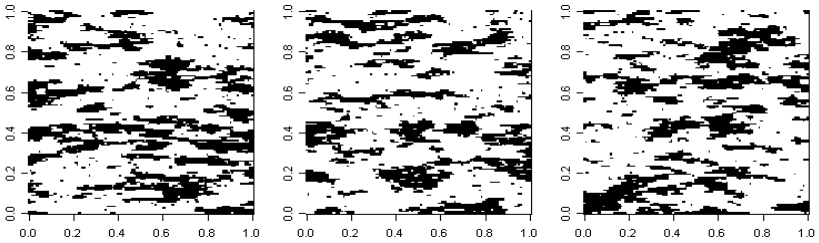


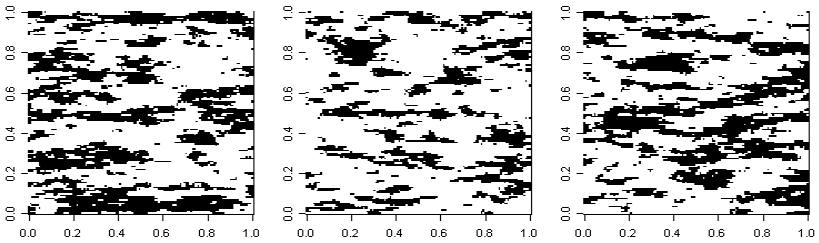
Figure 8: Test results for the Sisim image. The red line indicate the score for the training image, while the histograms show how the scores are distributed for 500 realizations from each of the models. The x-axis show the value of the property, while the y-axis is used to show what model the histogram corresponds to. The blue line separates the models that are used to study k, d, or r. The models are specified in Table 2.



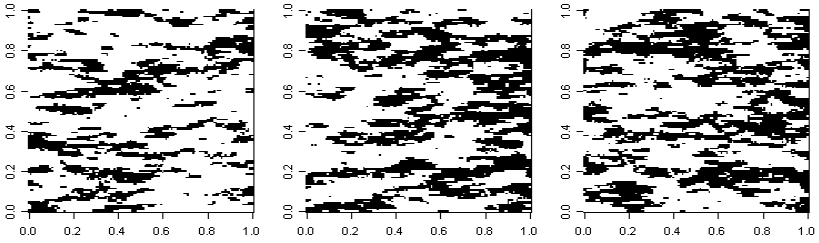
(a) Training image



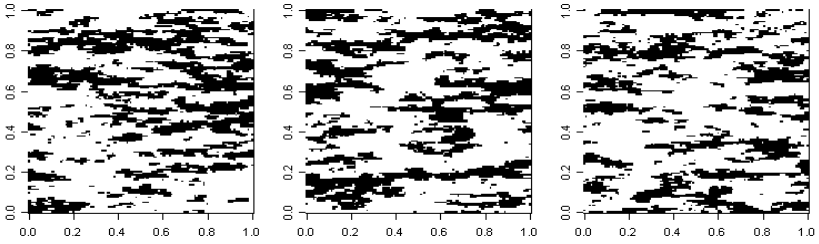
(b) Realizations from model number 5



(c) Realizations from model number 8/11



(d) Realizations from model number 9



(e) Realizations from model number 13

Figure 9: Realizations from some of the models when the Sisim image was used as training image. The training image is shown for comparison. It is hard to see clear differences by just looking at the realizations, but one difference when compared to the training image is that the realizations have more really small black objects, of only a few pixels in size. This is especially true for model number 5, the one that performed worst on the property tests in Figure 8. Another difference is that there are some quite large black objects in the realizations.

the objects seem to improve just as much. As a result of the change in width and height we see that the number of objects also decreases a little. The overall impression is that we get better results by increasing r, but the gain is very small, so it might not be worth it compared to the fact that the models get more parameters.

We see that model number 13 performs better on some of the properties, especially *o*, *hq*, *vq* and *mwo*. The local properties all increase compared to the other models, which means we have more black in the images. At the same time we get fewer objects, and this indicates that the objects in general are getting larger. However the largest object does not increase a lot, so it is apparently the smaller objects that increase in size. It is unexpected that the horizontal properties *hq* and *mwo* improve when the one-dimensional MRF are vertical and not horizontal. This might indicate that if the texture contains objects that are longest in the horizontal direction (as the Sisim image does), then we should let the one-dimensional MRF be vertical. Realizations of model number 13 are shown in Figure 9(e).

6.2 Channel

The Channel image describes rivers or channels where sediments have been deposited. The texture is, unlike the Sisim image, quite clear. An important aspect of the Channel image is that the channels can not just suddenly end. This means that the channels should either run into each other, or out through the edge of the image. The easiest way to see if this property holds is to look at realizations, but it will also be reflected in the mean width of the objects, as this should be close to the width of the image (125 pixels). We see from *mwo* in Figure 10 that this is not true, and in the realizations in Figure 11 we see, as expected, that some objects suddenly end. There are even isolated objects in the middle of some images.

One of the things we notice from the property tests in Figure 10 is that the local properties are not as good as they were for the Sisim image. If we look at local properties and global properties at the same time we realize that the models that perform well on local properties perform bad on global properties, and visa versa, indicating that it is problematic to get good values for both at the same time. Comparing horizontal and vertical properties we see that the method has problems with *vq*, while *hq* seems to be about right in most models. Another thing to note is that this is the only image where the largest object is not too large compared to the training image, which is not unexpected since the largest object in the training image is quite large.

When we change k it is clear that all local properties increase, which in general mean more black in the images. At the same time the number of objects are decreasing, and the largest object keeps getting larger. The width of the objects are also improving a little, but not as much as we might have hoped, and it actually looks like the height of the objects improve more than the width. It is a little bit unfortunate that *pb*, *hp* and

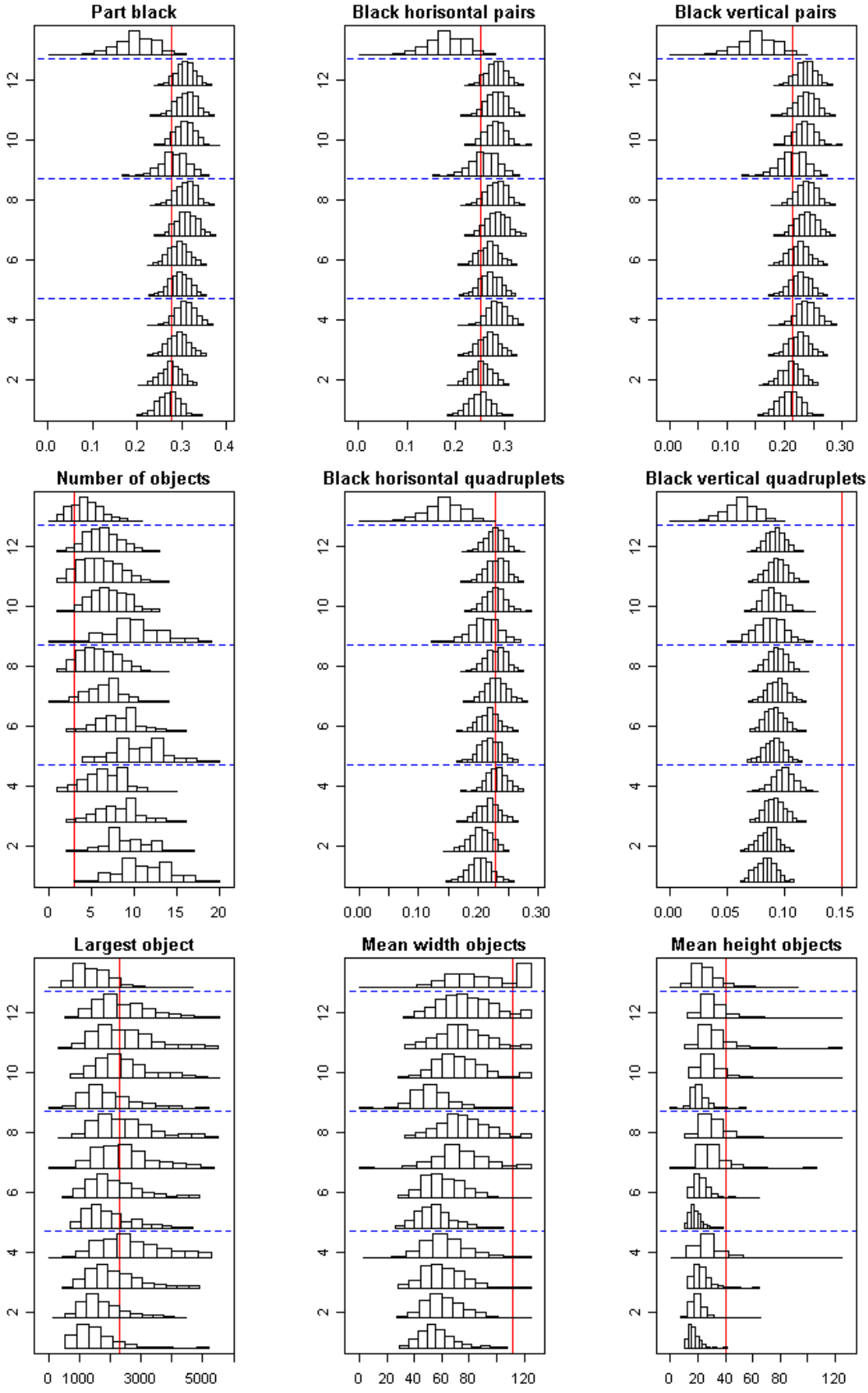
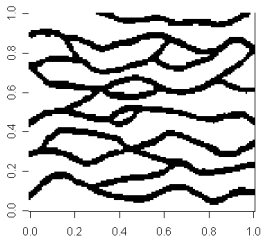
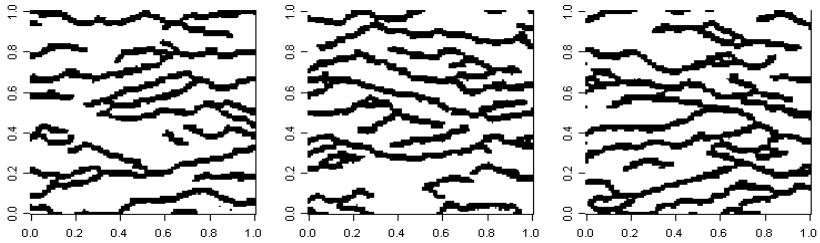


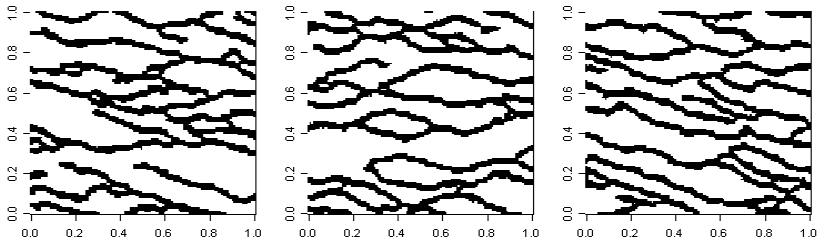
Figure 10: Test results for the Channel image, using the same setup as in Figure 8. The models are specified in Table 2.



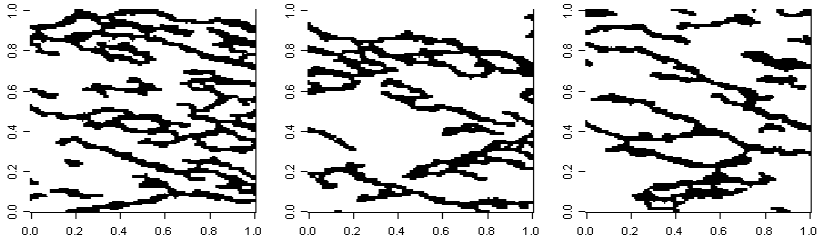
(a) Training image



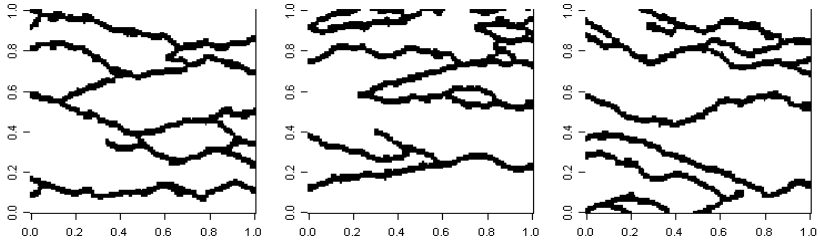
(b) Realizations from model number 5



(c) Realizations from model number 8/11



(d) Realizations from model number 9



(e) Realizations from model number 13

Figure 11: Realizations from some of the models, using the Channel image as training image. The training image is shown for comparison. A general problem for this texture is that the channels suddenly end in the middle of the image, something we see in all these realizations.

vp moves away from the value of the training image, while the other properties moves toward the value of the training image. This indicates that the objects are more connected, but maybe too much connected, leading to images with too much black. Anyway, we are quite happy as long as the global properties gets better, so we want to use large neighborhoods even if this means images with a little too much black.

Increasing d has pretty much the same effect as increasing k, since *pb*, *hp* and *vp* all move away from the value of the training image, while the other properties move towards the value of the training image. We now see that the width of the objects improve a lot more than when we just increased k, which means that d is important to connect the channels so that they do not end in the middle of the image, something we can see in Figure 11(b) and 11(c), where we look at models number 5 and 8/11. This change is reflected in all properties, as all have a change in performance when we increase d, especially when we go from $d = 3$ to $d = 4$, and it is therefore clear that we want $d \geq 4$.

When we change r we once again observe the same thing as when we changed k and d, that *pb*, *hp* and *vp* all move away from the value of the training image, while the other properties move towards the value of the training image. However, the change is only noticeable when we go from $r = 2$ to $r = 3$, as the last three models perform pretty much the same on all properties. The reason is that when we condition on only two rows we do not get enough information about how thick the channels should be. The result is that the realizations have channels without the right thickness, something we can see in Figure 11(d). Once we have enough rows to get this information we do not gain a lot by conditioning on any more rows, and thus the result does not change a lot in the last three models. The shape of the objects in the training image can in other words tell us something about what value we should choose for r.

The Channel image is, with the channels going from one side to the other, very different vertically and horizontally, and we see that the results change a lot when we use the columns instead of the rows as the one-dimensional MRF in model number 13. The easiest way to see what the difference is, is by looking at the realizations in Figure 11(e). We see that there are too much white space between the channels, which explain why the local properties give such small values (less black in the images). The positive thing is that *mwo* is increasing, more of the objects go through the entire image, but from the realizations we see that we still have some dead ends.

6.3 Ellipse

The Ellipse image is a realization of a marked Poisson random field. This means that objects (marks), of equal size and shape, are uniformly distributed inside the image, and the number of objects to be used is randomly drawn from a Poisson distribution. In our case the objects are ellipses, thus the name of the image. Some of the objects will of course overlap, creating larger objects, but there will always be some isolated objects. This means that one of the important aspects of the texture is the size and shape of

the objects, especially important is the fact that we do not want objects that are too small. MRF usually have problems with textures where there are many objects of equal size, and we expect this to also be true for our MDMC model. To get the best size of the objects we expect that we must use large neighborhoods and condition on several previous rows.

Looking at the properties in Figure 12 we see that the average size of the objects at least seems to be about right, but when we look at realizations from some of the models in Figure 13 we see that the objects vary much in size, with some objects smaller than the ellipses in the training image. It seems like most models perform well on both local and global properties, and there does not seem to be any differences in the vertical and horizontal properties. We also note that for this image the realizations does not contain too many objects compared to the training image, something that was a problem for the two other training images.

Increasing k does not seem to have any effect at all on the local properties, while it clearly effects the average size and the number of objects. We get fewer objects and their average size gets larger, while the amount of black in the images stay the same. We see that compared to the training image these changes in the global properties are for the worse, which indicates that increasing k alone is not enough to improve the results.

When we in the next four models increase d we see that the performance improves for pretty much all properties, and especially when we go from $d = 3$ to $d = 4$. We also saw this for the Channel image, so it is clear that we should try to use $d \geq 4$. Models number 7 and 8/11 are actually performing almost perfect compared to the training image, but when we look at realizations from model number 8/11 in Figure 13(c) we see that it actually does not look quite like the training image.

It is problematic to say anything about the effect of changing r, as model number 8/11 and 12 breaks any possible trends. We would expect the results of model number 12 to be more similar to those of model 8/11, so we even ran the optimization routine one more time for both models, just to make sure we still got the same result. The result stayed the same however, and we have the problem of trying to explain why the result is as it is. It is weird that when we use $r = 4$ we get such a good result, but when we increase r by one the result is suddenly a lot worse. We are unable to see any reasonable explanations.

The realizations of model number 13 perform almost the same on the property tests as those of model number 8/11. Only one of the properties, *vq*, give worse results, while the other either improve or stay the same. Realizations from the model are shown in Figure 13(e), and these does in fact look more like the training image than those from the other models. The edges of the objects are more similar to those in the training image, mostly because the top and bottom of the objects are straighter. Thus this image also

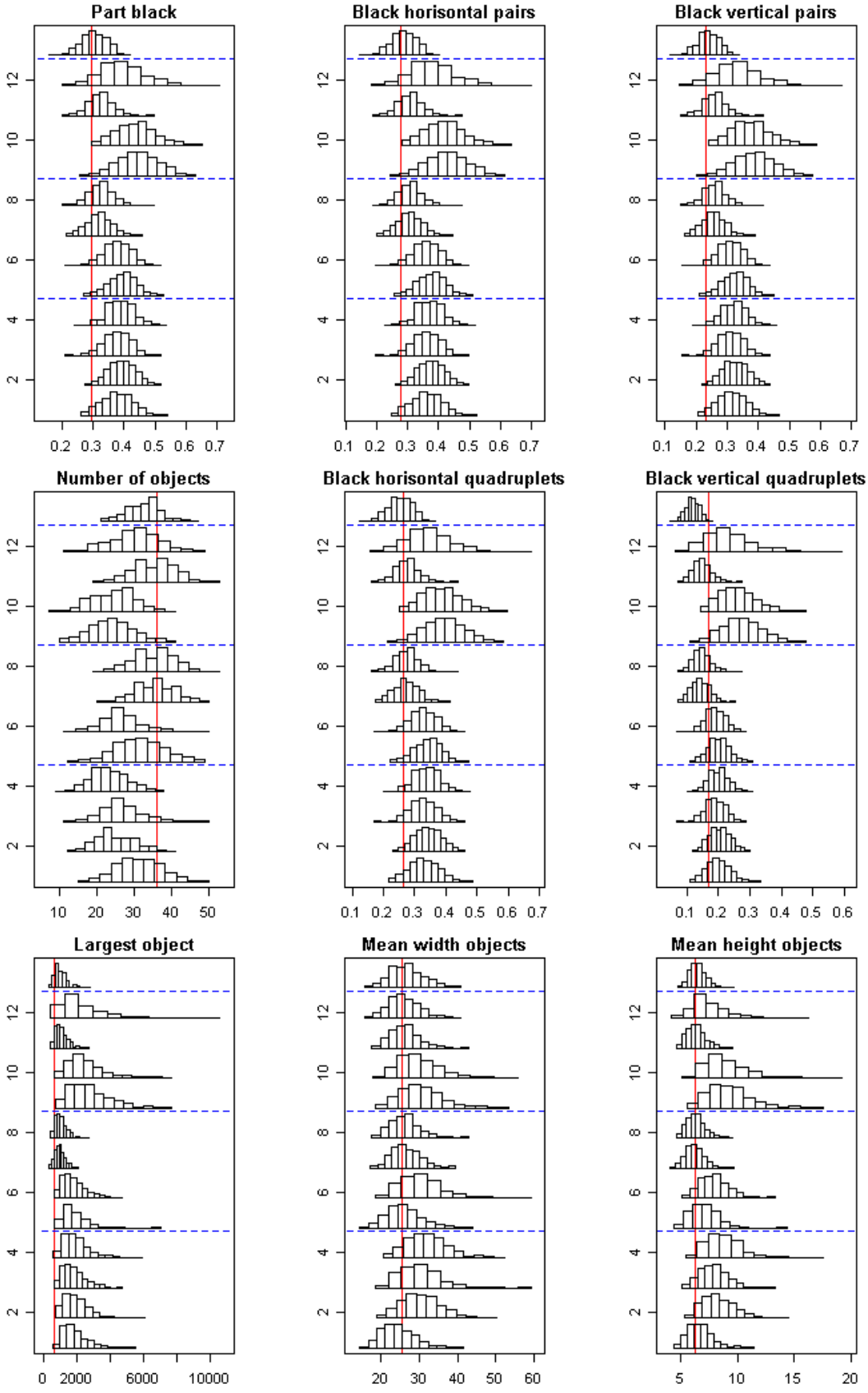
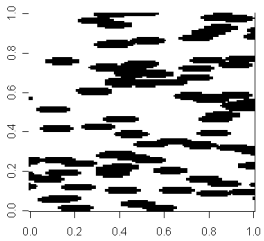
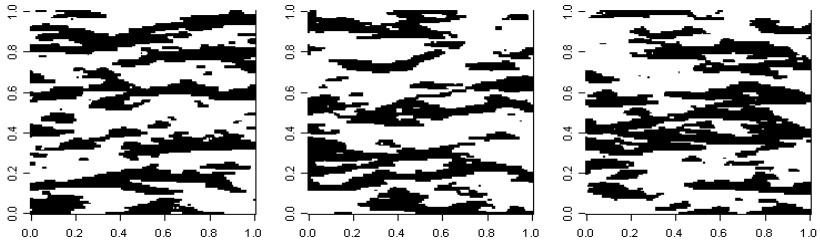


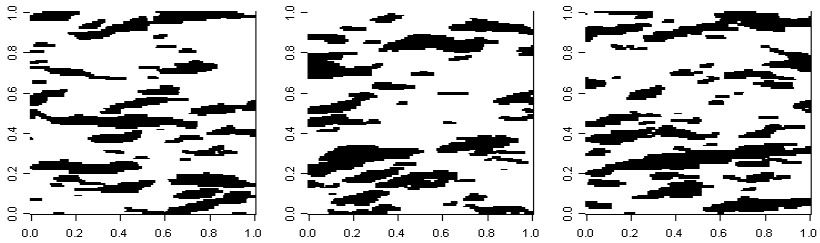
Figure 12: Test results for the Ellipse image for the thirteen models specified in Table 2. The setup is the same as for the two other training images.



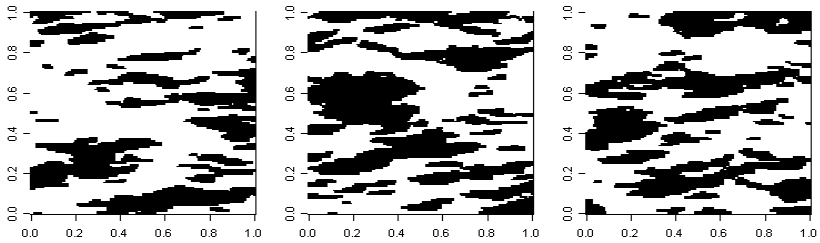
(a) Training image



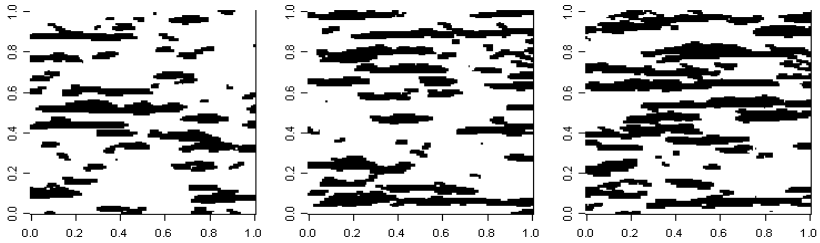
(b) Realizations from model number 5



(c) Realizations from model number 8/11



(d) Realizations from model number 9



(e) Realizations from model number 13

Figure 13: Realizations from some of the models, using the Ellipse image as training image. The training image is shown for comparison. MRF are known to have trouble if the texture contains many objects of equal size, and it seems the MDMC model also have this problem.

indicate that when the objects are longest in the horizontal direction, then we should let the one-dimensional MRF be vertical.

7 Closing remarks

In this thesis we have studied a parametrization of a multidimensional Markov chain model with the goal of capturing the texture in training images. The results above show us that the method is able to capture a lot of the information in the training images, but that there is always something missing, something that goes wrong, so it is reasonable to believe that the model can somehow be improved to give better results. If we want to use the method we have to choose what values of k, r and d we want to use. The results does not always give us any clear answers here, but we are at least able to learn something.

The general trend seems to be that increasing k leads to better results. There are some properties in some of the images that get worse as k increases, but this might be caused by other factors, for example that we used $d = 3$ in the four models where we changed k. The results improve a lot as we increase d from $d = 3$ to $d = 4$, so it is at least clear that we want to use $d \geq 4$. However the results does not change as much from $d = 4$ to $d = 5$, so it seems $d = 4$ is an appropriate value. When it comes to r it seems that we should choose to condition on just enough rows so that we get the height of most of the objects in the training image. For images like Channel this means the thickness of the channels, while in Ellipse it is the height of the ellipses. If the objects are more varied in size, like in the Sisim image, or the objects are to high to get the entire height, we will just have to settle for a value, probably choosing the largest value we can while keeping a decent number of parameters in the model. Overall it seems like model number 7 perform quite good for all three training images, so if unsure about what values to use for k, r and d it can be a good idea to first try $k = 4$, $r = 4$ and $d = 4$.

We do not have a lot of information about what happens if we increase k, r or d beyond these values, but it is reasonable to believe that the results might improve. However this will also give us more parameters, and the optimization will take more time, so any improvements must be weighted against this loss of efficiency.

It is not easy to say when we should use the columns and when we should use the rows as the one-dimensional MRF. It obviously depends on the texture in the training image, and the best thing is probably to try both. The Sisim and Ellipse images seems to indicate that if the objects are longest in the horizontal direction, then we should let the one-dimensional MRF be vertical. However the Channel image also has objects that are longest in the horizontal direction, and the best results here was obtained when the MRF was horizontal. At least we know for a fact that the method gives different results when we change from rows to columns, so if it does not work well to use the rows one might as well try to use the columns and hope the results are better.

One thing to think about, that we did not have the time to test, is how this parametrization would work if we had more than two colors in the training image. When we created the parametrization we had the three training images in mind, and did not really think about the case where images have more colors. Since the parametrization only takes into account one color at the time on the nodes in the cliques, we do not really gather any information about how different colors relate to each other. This is not such a big deal when we only have two colors, since if we get to the edge of a black object we then always have to use white in the next object. If we had three colors, say black white and gray, we would have to choose between gray or white whenever we got to the edge of a black object. With the current parametrization this information will not be captured by the parameters. To capture such information we must also use parameters for cliques where part of the clique has one color, and the other part has another color.

One way to improve the model is therefore to also include parameters that explain cliques with more than just one color on the nodes. The approach could be similar to the one we have used, where we trace how many rows backward we find the same clique with the same colors, but some other method would probably be better, one that also takes into account what color it changes into. It is clear that this would give us more parameters, and to reduce the number of parameters it seems like it can be a good idea to ignore certain clique types, but instead of just ignoring the largest clique types we could for example ignore certain clique types of all sizes. What clique types to ignore could for example be selected based on information in the training image.

References

Baum, L. E., Petrie, T., Soules, G. & Weiss, N. (1970), 'A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains', *The Annals of Mathematical Statistics* **41**(1), 164–171.

Besag, J. (1973), 'Spatial interaction and the statistical analysis of lattice systems', *Journal of the Royal Statistical Society, series B* **36**(2), 192–236.

Friel, N. & Rue, H. (2007), 'Recursions computing and simulation-free inference for general factorizable models', *Biometrika* **94**(3), 661–672.

Nerhus, S. (2008), Automatic parametrization for binary multidimentional Markov random fields. Technical report, Department of Mathematical Science, Norvegian University of Sience and Technology.

Qian, W. & Titterington, D. M. (1991), 'Multidimensional Markov chain models for image textures', *Journal of the Royal Statistical Society, series B* **53**(3), 661–674.

Reeves, R. & Pettitt, A. N. (2004), 'Efficient recursions for general factorisable models', *Biometrika* **91**(3), 751–757.