NORGES TEKNISK-NATURVITENSKAPELIGE
UNIVERSITET

An approximate forward-backward algorithm applied to binary Markov random fields

by

Haakon Michael Austad and Håkon Tjelmeland

PREPRINT
STATISTICS NO. 11/2011

NORWEGIAN UNIVERSITY OF SCIENCE AND
TECHNOLOGY
TRONDHEIM, NORWAY

This report has URL http://www.math.ntnu.no/preprint/statistics/2011/S11-2011.pdf
Håkon Tjelmeland has homepage: http://www.math.ntnu.no/∼haakont
E-mail: haakont@stat.ntnu.no
Address: Department of Mathematical Sciences, Norwegian University of Science and Technology, N-7491 Trondheim, Norway.

An approximate forward-backward algorithm applied to binary Markov random fields

Haakon Michael Austad and Håkon Tjelmeland
Department of Mathematical Sciences
Norwegian University of Science and Technology

Abstract

In this report we propose a new approximate version of the well known forward-backward algorithm and use this to perform approximate inference on Markov random fields. We construct the approximate forward-backward algorithm by adapting approximation results for pseudo-Boolean functions. By using an approximation of the energy function which minimizes the error sum of squares we construct a forward-backward algorithm which is computationally viable. We also show how our approach gives us upper and lower bounds as well as an approximate Viterbi algorithm. Through two simulation examples and a real data example we demonstrate the accuracy and flexibility of the algorithm.

Key words: Markov random fields, pseudo-Boolean functions, forward-backward algorithm, approximate inference

1 Introduction

In statistics in general and perhaps especially in spatial statistics we often find ourselves with distributions known only up to an unknown normalization constant. Calculating this normalizing constant typically involves high dimensional summation or integration. This is the case for the class of discrete distributions known as discrete Markov random fields (MRF).

A common situation in spatial statistics is that we have some unobserved latent field x for which we have noisy observations y. We model x as an MRF with unknown parameters θ around which we want to do inference of some kind. If we are Bayesians we could imagine adding some prior for our parameters θ and studying the posterior distribution $p(\theta|y)$. A frequentist approach could involve finding a maximum likelihood estimator for our parameters. Independently or in combination with these investigations we might want to perform simulations and generate samples from $p(x|\theta)$ for some values of θ. Without the normalizing constant however, all these become non-trivial tasks.

There are a number of techniques that have been proposed to overcome this problem. The normalizing constant can be estimated by running Markov chain Monte Carlo (MCMC) which can then be combined with various techniques to produce maximum likelihood estimates, see for instance Geyer and Thompson (1992), Gelman and Meng (1998) and Gu and Zhu (2001). Other approaches take advantage of the fact that exact sampling can be done, see Møller et al. (2006). In the present report however, we focus on the class of deterministic methods, where by deterministic we mean that repeating the estimation process yields the same estimate. In Reeves and Pettitt (2004) the authors devise a computationally efficient algorithm for handling so called general factorisable models of which MRFs are a common example. This algorithm, which we refer to from here on as the forward-backward algorithm, grants a large computational saving in calculating the normalizing constant by exploiting the factorisable structure of the models. For MRFs defined on a lattice this allows for calculation of the normalizing constant on lattices with up to around 20 rows for models with first order neighborhoods. In Friel and Rue (2007) and Friel et al. (2009) the authors construct approximations for larger lattices by doing computations for a number of sub-lattices using the algorithm in Reeves and Pettitt (2004).

The energy function of an MRF is an example of a so called pseudo-Boolean function. In general, a pseudo-Boolean function is a function of the following type, $f : \{0,1\}^n \to \mathbb{R}$. A full representation of a pseudo-Boolean function requires 2^n terms. Finding approximate representations of pseudo-Boolean functions that require fewer coefficients is a well studied field, see Hammer and Holzman (1992) and Grabisch et al. (2000). In Hammer and Rudeanu (1968) the authors show how any pseudo-Boolean function can be expressed as a binary polynomial in n variables. Tjelmeland and Austad (2012) expressed the energy function of MRFs in this manner and by dropping small terms during the forward part of the forward-backward algorithm constructed an approximate MRF.

Our approach and the main contribution of this report is to apply and modify methods from pseudo-Boolean function approximation to design an approximate forward-backward algorithm. By approximating the binary polynomial representing the distribution before summing out each variable we get an algorithm less restricted by the correlation structure of the model, thus capable of handling MRFs defined on large lattices and MRFs with larger neighborhood structures. For the MRF application this approximation defines an approximate MRF for which we can calculate the normalizing constant or evaluate the likelihood as well as generate realizations. With our approach to approximating MRFs we also show how we can construct upper and lower bounds for the normalizing constant, and thus the likelihood, as well as construct an approximate Viterbi algorithm, see Künsch (2001).

The report has the following layout. In Section 2 we formally introduce pseudo-Boolean functions and their polynomial representation and show some results for approximative representations. Section 3 details the forward-backward algorithm for calculating the normalizing constant and likelihood for an MRF, using the notation established in Section 2. Then in Section 4 we show how we can modify approximation results for pseudo-Boolean functions to construct our approximative forward backward algorithm. In Section 5 we extend this to the Viterbi algorithm. Section 6 includes a number of examples demonstrat-

ing the accuracy of our new approximations. Finally in Section 7 we include some closing comments and conclusions.

2 Pseudo-Boolean functions

In this section we introduce pseudo-Boolean functions and discuss various aspects of approximating pseudo-Boolean functions using the results of Hammer and Holzman (1992) and Grabisch et al. (2000). We end the section by showing how we can calculate the approximation for a particular design of the approximating function.

2.1 Definitions and notation

Let $x = (x_1, \ldots, x_n) \in \Omega = \{0,1\}^n$ be a vector of binary variables and let $N = \{1, \ldots, n\}$ be the corresponding list of indices. Then for any subset $\Lambda \subseteq N$ we associate an incidence vector x of length n whose kth element is 1 if $k \in \Lambda$ and 0 otherwise. We refer to an element of x, x_k, as being "on" if it has value 1 and "off" if it is 0. A pseudo-Boolean function f, of dimension $\dim(f) = n$, is a function that associates a real numbered value to each vector, $x \in \{0,1\}^n$, i.e $f : \{0,1\}^n \to \mathbb{R}$. The simplest representation of a pseudo Boolean function is simply a list, using some ordering, of the 2^n values we associate with the incidence vectors. Hammer and Rudeanu (1968) showed that any pseudo-Boolean function can be expressed uniquely as a binary polynomial,

$$f(x) = \sum_{\Lambda \subseteq N} \beta^{\Lambda} \prod_{k \in \Lambda} x_k, \tag{1}$$

where β^{Λ} are real coefficients which we refer to as interactions. We define the degree of f, $\deg(f)$ as the degree of the polynomial and call x_k a nuisance variable if $f(x_1, \ldots, x_k = 0, \ldots, x_n) = f(x_1, \ldots, x_k = 1, \ldots, x_n)$ for all x_{-k}, where $x_{-k} = (x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n)$. Note that x_k being a nuisance variable is equivalent to $\beta^{\Lambda} = 0$ for all Λ where $k \in \Lambda$.

In general the representation of a function in this manner requires 2^n coefficients. In some cases one or more β^{Λ} might be zero and in this case a reduced representation of the pseudo-Boolean function can be defined by excluding some or all the terms in the sum in (1) where $\beta^{\Lambda} = 0$. Thus we get,

$$f(x) = \sum_{\Lambda \in S} \beta^{\Lambda} \prod_{k \in \Lambda} x_k, \tag{2}$$

where S is a set of subsets of N at least containing all $\Lambda \subseteq N$ for which $\beta^{\Lambda} \neq 0$. We say that our representation of f is dense if for all $\Lambda \in S$ all subsets of Λ are included in S. The minimal dense representation of f is thereby (2) with,

$$S = \{\lambda \subseteq N : \beta^{\Lambda} \neq 0 \text{ for some } \Lambda \supseteq \lambda\}. \tag{3}$$

Throughout this report we restrict the attention to dense representations of pseudo-Boolean functions. Note however that some of the theorems below are valid also without this restriction.

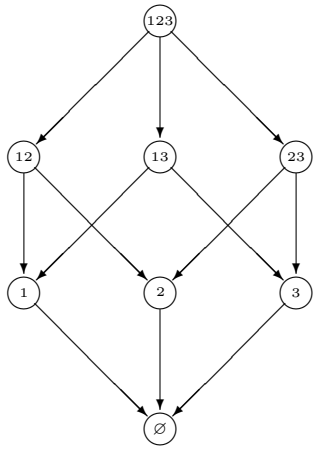

Figure 1: DAG representation of the pseudo-Boolean function in (4). Nodes can be thought of as either representing the set of interactions S or the set of states Ω.

Before we proceed further we introduce a small example to illustrate some properties and notation regarding pseudo-Boolean functions. We will refer to this example to illustrate properties of pseudo-Boolean functions throughout the report. Let $n = 3$, so $N = \{1, 2, 3\}$ and assume that all interactions are non-zero so $S = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$ and,

$$f(x_1, x_2, x_3) = \beta^{\varnothing} + \beta^1 x_1 + \beta^2 x_2 + \beta^3 x_3 + \beta^{12} x_1 x_2 + \beta^{13} x_1 x_3 + \beta^{23} x_2 x_3 + \beta^{123} x_1 x_2 x_3. \quad (4)$$

We have found that when working with pseudo-Boolean functions, it is useful to visualize the set S as a directed acyclic graph (DAG). Each node λ in the graph represents an interaction β^{λ} and each node has a vertex connecting it to all nodes where $|\Lambda| = |\lambda| - 1$ and $\Lambda \subset \lambda$, see Figure 1. We refer to these nodes as the children of λ and the nodes $\Lambda \supset \lambda$, such that $|\Lambda| = |\lambda| + 1$, as the parents of λ. Note that the number of children of each node will be equal to $|\lambda|$ and the number of parents of each node will be equal to $n - |\lambda|$, assuming the interaction parameters of all parents to be non-zero. The graph representation of our example function (4) can be seen in Figure 1. It can also be useful to think of the graph in Figure 1 as representing the set Ω, each node λ in this case representing the configuration of x where $x_k = 1$ if $k \in \lambda$ and $x_k = 0$ otherwise. So for instance node $\{1, 2\}$ represents the state $x = (1, 1, 0)$. We must note an important difference between representing the set S and the set Ω in this manner. The set S will not necessarily include all $\lambda \subseteq N$, thus the graph need not be full as in Figure 1. If β^{123} where zero for instance, then the node $\{1, 2, 3\}$ would not be present. This will never be the case if we let the graph represent Ω. For this set we, for obvious reasons, always include all configurations of x.

We now introduce some notation we need later in the report. We define the subset S_λ as the set $S_\lambda = \{\Lambda \in S : \lambda \subseteq \Lambda\}$. Think of this as all interactions that include the interaction λ. Using the graph representation of S, S_λ consists of all nodes starting at node λ and moving up in the graph. So, in our example, $S_{\{1,2\}} = \{\{1, 2\}, \{1, 2, 3\}\}$. Equivalently

for the set Ω, we define the subset Ω_λ as the set $\Omega_\lambda = \{x \in \Omega : x_k = 1,\ \forall k \in \lambda\}$. So this is the set of all x where a given selection of x_k are on. Using the graph representation, we again find these nodes by starting at node λ and moving up. For our example function we have for instance $\Omega_{\{1,2\}} = \{\{1,1,0\},\{1,1,1\}\}$. We later also need the complements of these two subsets, $S_\lambda^c = S \backslash S_\lambda$ and $\Omega_\lambda^c = \Omega \backslash \Omega_\lambda$. Lastly we define $S_\lambda^0 = \{\Lambda \in S : \lambda \cap \Lambda = \varnothing\}$ and $\Omega_\lambda^0 = \{x \in \Omega : x_k = 0,\ \forall k \in \lambda\}$. If we think of the sets S_λ and Ω_λ as the sets where λ is on, then S_λ^0 and Ω_λ^0 are the sets where λ is off. Again, using our example we have for instance $S_{\{1,2\}}^0 = \{\varnothing, 3\}$ and $\Omega_{\{1,2\}}^0 = \{\{0,0,0\},\{0,0,1\}\}$. Note that in general $S_\lambda^c \neq S_\lambda^0$ and equivalently $\Omega_\lambda^c \neq \Omega_\lambda^0$.

2.2 Approximating pseudo-Boolean functions

Since for a general pseudo-Boolean function, the number of interactions in our representation grows exponentially with the dimension n, it is natural to ask if we can find an approximate representation of the function that reduces the number of interactions required for storage. We could choose some set $\tilde{S} \subseteq S$ to define our approximation, thus choosing which interactions to leave, $\tilde{S}$, and which to remove, $S \backslash \tilde{S}$. For a given $\tilde{S}$ our interest lies in the best such approximation according to some criteria. We define $A_{\tilde{S}}(f(x)) = \tilde{f}(x) = \sum_{\Lambda \in \tilde{S}} \tilde{\beta}^\Lambda \prod_{k \in \Lambda} x_k$ as the operator which returns the approximation that, for some given approximation set $\tilde{S}$, minimizes the error sum of squares (SSE),

$$\mathrm{SSE}(f, \tilde{f}) = \sum_{x \in \Omega} \Big(f(x) - \tilde{f}(x) \Big)^2. \tag{5}$$

We find the best approximation by taking partial derivatives with respect to $\tilde{\beta}^\lambda$ for all $\lambda \in \tilde{S}$ and setting these expressions equal to zero. This gives us a system of linear equations,

$$\sum_{x \in \Omega_\lambda} \tilde{f}(x) = \sum_{x \in \Omega_\lambda} \left[\sum_{\Lambda \in \tilde{S}} \tilde{\beta}^\Lambda \prod_{k \in \Lambda} x_k \right] = \sum_{x \in \Omega_\lambda} f(x),\ \forall\ \lambda \in \tilde{S}. \tag{6}$$

Existence and uniqueness of a solution is assured since we have $|\tilde{S}|$ linearly independent equations and $|\tilde{S}|$ unknown variables. Clearly if $\tilde{S} \supseteq S$, then the best approximation is the function itself, $\tilde{f}(x) = f(x)$.

It is common practice in statistics and approximation theory in general to approximate higher order terms by lower order terms. A natural way to design an approximation would be to let $\tilde{S}$ include all interactions of degree less than or equal to some value k. In Hammer and Holzman (1992) the authors focus on approximations of this type and proceed to show how the resulting system of linear equations through clever reorganization can be transformed into a lower triangular system. They solve this for $k = 1$ and $k = 2$ as well as proving a number of useful properties. Grabisch et al. (2000) proceed to solve this for a general value of k.

In the present report we consider a similar design of the set $\tilde{S}$, namely the case where $\tilde{S}$ is a dense subset of S. So if $\lambda \in \tilde{S}$, then all $\Lambda \subset \lambda$ must also be included in $\tilde{S}$.

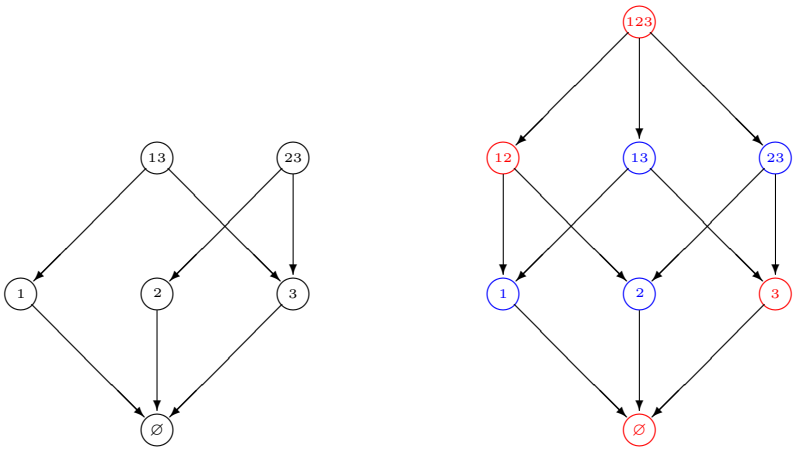

Figure 2: Left: DAG representation of the pseudo-Boolean function defined over the set $\tilde{S} = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,3\}, \{2,3\}\}$. Right: Distribution of the error $f(x) - \tilde{f}(x)$ when $S = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\}$ and $\tilde{S} = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,3\}, \{2,3\}\}$. For states represented by a red node $f(x) - \tilde{f}(x) = \frac{\beta^{12}}{2^2}$ while for states represented by a blue node, $f(x) - \tilde{f}(x) = -\frac{\beta^{12}}{2^2}$.

Although this may at first sound restrictive we will see that it is not inhibitive for our application on MRFs later in the report. Consider our example in (4). We could choose $\tilde{S} = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,3\}, \{2,3\}\}$, see the DAG representation to the left in Figure 2. Clearly the approximation using all interactions up to degree k is a special case of our class of approximations. Our motivation for studying this particular design of $\tilde{S}$ will become clear as we study the forward-backward algorithm. We now proceed to prove some useful properties of this approximation. Note that the first two theorems where proved in Hammer and Holzman (1992) (using different proofs), and the proofs and theorems are valid for our class of approximations as well. We include these theorems here with proofs for completeness and insight.

Theorem 1. The above approximation $A_{\tilde{S}}(f(x))$ is a linear operator, i.e. for any constants $a, b \in \mathbb{R}$ and pseudo-Boolean functions $g(x)$ and $h(x)$ defined over S, we have that $A_{\tilde{S}}(ag(x) + bh(x)) = aA_{\tilde{S}}(g(x)) + bA_{\tilde{S}}(h(x))$.

Proof. Let $\tilde{f}(x) = A_{\tilde{S}}(f(x))$, $\tilde{g}(x) = A_{\tilde{S}}(g(x))$ and $\tilde{h}(x) = A_{\tilde{S}}(h(x))$. We show the theorem by inserting $f(x) = ag(x) + bh(x)$ and $\tilde{f}(x) = a\tilde{g}(x) + b\tilde{h}(x)$ in (6),

$$\sum_{x \in \Omega_\lambda} \tilde{f}(x) = \sum_{x \in \Omega_\lambda} \left[a\tilde{g}(x) + b\tilde{h}(x) \right] = \sum_{x \in \Omega_\lambda} f(x) = \sum_{x \in \Omega_\lambda} \left[ag(x) + bh(x) \right], \ \forall \ \lambda \in \tilde{S}.$$

This is clearly satisfied if we require,

$$\sum_{x\in\Omega_\lambda} \tilde{g}(x) = \sum_{x\in\Omega_\lambda} g(x),\ \forall\ \lambda \in \tilde{S},$$
$$\sum_{x\in\Omega_\lambda} \tilde{h}(x) = \sum_{x\in\Omega_\lambda} h(x),\ \forall\ \lambda \in \tilde{S}.$$

Each of these systems of equations contains $|\tilde{S}|$ equations and $|\tilde{S}|$ variables and thus have a unique solution. Since we know that (6) has a unique solution, this completes the proof. □

Since each interaction term in a pseudo-Boolean function is a pseudo-Boolean function in itself, this theorem is important because it means that we can approximate a pseudo-Boolean function by approximating each of the interaction terms involved in the function individually. Also, since the best approximation of a pseudo-Boolean function is itself, we only need to worry about how to approximate the interaction terms we want to remove.

Theorem 2. Assume we have two approximations of $f(x)$, $A_{\tilde{S}}(f(x))$ and $A_{\tilde{\tilde{S}}}(f(x))$, such that $\tilde{\tilde{S}} \subseteq \tilde{S} \subseteq S$. Then $A_{\tilde{\tilde{S}}}(A_{\tilde{S}}(f(x))) = A_{\tilde{\tilde{S}}}(f(x))$.

Proof. Let $\tilde{f}(x) = A_{\tilde{S}}(f(x)) = \sum_{\Lambda\in\tilde{S}} \tilde{\beta}^\Lambda \prod_{k\in\Lambda} x_k$ and $\tilde{\tilde{f}}(x) = A_{\tilde{\tilde{S}}}(f(x)) = \sum_{\Lambda\in\tilde{\tilde{S}}} \tilde{\tilde{\beta}}^\Lambda \prod_{k\in\Lambda} x_k$. Again, we prove the theorem by studying the equations that specify the solutions. For $\tilde{f}(x) = A_{\tilde{S}}(f(x))$ we have,

$$\sum_{x\in\Omega_\lambda} A_{\tilde{S}}(f(x)) = \sum_{x\in\Omega_\lambda} f(x),\ \forall\ \lambda \in \tilde{S}. \tag{7}$$

Correspondingly, for $A_{\tilde{\tilde{S}}}(\tilde{f}(x))$ we get,

$$\sum_{x\in\Omega_\lambda} A_{\tilde{\tilde{S}}}(\tilde{f}(x)) = \sum_{x\in\Omega_\lambda} \tilde{f}(x),\ \forall\ \lambda \in \tilde{\tilde{S}}. \tag{8}$$

Since $\tilde{f}(x) = A_{\tilde{S}}(f(x))$ and $\tilde{\tilde{S}} \subseteq \tilde{S}$ we can combine (7) and (8) to get,

$$\sum_{x\in\Omega_\lambda} A_{\tilde{\tilde{S}}}(\tilde{f}(x)) = \sum_{x\in\Omega_\lambda} \tilde{f}(x) = \sum_{x\in\Omega_\lambda} A_{\tilde{S}}(f(x)) = \sum_{x\in\Omega_\lambda} f(x),\ \forall\ \lambda \in \tilde{\tilde{S}}. \tag{9}$$

This is the same set of equations as for $A_{\tilde{\tilde{S}}}(f(x))$, so since the solution exist and is unique, we get the same approximation. □

This theorem shows that an iterative scheme for calculating the approximation is possible. The next two theorems show useful properties regarding the error introduced by the approximation.

Theorem 3. Assume again that we have two approximations of $f(x)$, $A_{\tilde{S}}(f(x))$ and $A_{\tilde{\tilde{S}}}(f(x))$, such that $\tilde{\tilde{S}} \subseteq \tilde{S} \subseteq S$. Letting $\tilde{f}(x) = A_{\tilde{S}}(f(x))$ and $\tilde{\tilde{f}}(x) = A_{\tilde{\tilde{S}}}(f(x))$, we then have $\mathrm{SSE}(f, \tilde{\tilde{f}}) = \mathrm{SSE}(f, \tilde{f}) + \mathrm{SSE}(\tilde{f}, \tilde{\tilde{f}})$.

Proof. Expanding $\text{SSE}(f,\tilde{\tilde{f}}) = \sum_{x\in\Omega}\left(f(x)-\tilde{\tilde{f}}(x)\right)^2$ we get,

$$\sum_{x\in\Omega}\left(f(x)-\tilde{\tilde{f}}(x)\right)^2 = \sum_{x\in\Omega}\left(f(x)-\tilde{f}(x)+\tilde{f}(x)-\tilde{\tilde{f}}(x)\right)^2$$

$$= \sum_{x\in\Omega}\left(f(x)-\tilde{f}(x)\right)^2 + \sum_{x\in\Omega}\left(\tilde{f}(x)-\tilde{\tilde{f}}(x)\right)^2 + \sum_{x\in\Omega}(f(x)-\tilde{f}(x))(\tilde{f}(x)-\tilde{\tilde{f}}(x))$$

$$= \text{SSE}(f,\tilde{f}) + \text{SSE}(\tilde{f},\tilde{\tilde{f}}) + \sum_{x\in\Omega}(f(x)-\tilde{f}(x))\tilde{f}(x) - \sum_{x\in\Omega}(f(x)-\tilde{f}(x))\tilde{\tilde{f}}(x).$$

To prove the theorem it is sufficient to show that,

$$\sum_{x\in\Omega}(f(x)-\tilde{f}(x))\tilde{f}(x) - \sum_{x\in\Omega}(f(x)-\tilde{f}(x))\tilde{\tilde{f}}(x) = 0. \tag{10}$$

First recall that we from (6) know that,

$$\sum_{x\in\Omega_\lambda}\left[f(x)-\tilde{f}(x)\right] = 0,\ \forall\ \lambda\in\tilde{S}. \tag{11}$$

Also, since $\tilde{\tilde{S}}\subseteq\tilde{S}$,

$$\sum_{x\in\Omega_\lambda}\left[f(x)-\tilde{f}(x)\right] = 0,\ \forall\ \lambda\in\tilde{\tilde{S}}. \tag{12}$$

We study the first term, $\sum_{x\in\Omega}(f(x)-\tilde{f}(x))\tilde{f}(x)$, expand the expression for $\tilde{f}(x)$ outside the parenthesis and change the order of summation,

$$\begin{aligned}\sum_{x\in\Omega}(f(x)-\tilde{f}(x))\tilde{f}(x) &= \sum_{x\in\Omega}\left((f(x)-\tilde{f}(x))\sum_{\Lambda\in\tilde{S}}\tilde{\beta}^\Lambda\prod_{k\in\Lambda}x_k\right)\\ &= \sum_{\Lambda\in\tilde{S}}\left[\tilde{\beta}^\Lambda\sum_{x\in\Omega}\left(\prod_{k\in\Lambda}x_k(f(x)-\tilde{f}(x))\right)\right]\\ &= \sum_{\Lambda\in\tilde{S}}\left[\tilde{\beta}^\Lambda\sum_{x\in\Omega_\Lambda}\left(f(x)-\tilde{f}(x)\right)\right]\\ &= 0,\end{aligned}$$

where the last transition follows from (11). Using (12) we can correspondingly show that $\sum_{x\in\Omega}(f(x)-\tilde{f}(x))\tilde{\tilde{f}}(x) = 0$. □

The next Theorem gives some useful insight into how we can calculate $\text{SSE}(f,\tilde{f})$.

Theorem 4. Given a pseudo-Boolean function $f(x)$ and an approximation $\tilde{f}(x)$ constructed as described, the error sum of squares can be written as,

$$\sum_{x\in\Omega}\left[f(x)-\tilde{f}(x)\right]^2 = \sum_{\Lambda\in S\setminus\tilde{S}}\left[\beta^\Lambda\sum_{x\in\Omega_\Lambda}(f(x)-\tilde{f}(x))\right] \tag{13}$$

Proof. We study the error sum of squares,

$$\begin{aligned}\sum_{x\in\Omega}\left[f(x)-\tilde{f}(x)\right]^2 &= \sum_{x\in\Omega}\left[(f(x)-\tilde{f}(x))f(x)\right]-\sum_{x\in\Omega}\left[(f(x)-\tilde{f}(x))\tilde{f}(x)\right]\\ &= \sum_{x\in\Omega}\left[\sum_{\Lambda\in S}\beta^{\Lambda}(f(x)-\tilde{f}(x))\prod_{k\in\Lambda}x_k\right]-\sum_{x\in\Omega}\left[\sum_{\Lambda\in\tilde{S}}\tilde{\beta}^{\Lambda}(f(x)-\tilde{f}(x))\prod_{k\in\Lambda}x_k\right]\\ &= \sum_{\Lambda\in S}\beta^{\Lambda}\left[\sum_{x\in\Omega_\Lambda}(f(x)-\tilde{f}(x))\right]-\sum_{\Lambda\in\tilde{S}}\tilde{\beta}^{\Lambda}\left[\sum_{x\in\Omega_\Lambda}(f(x)-\tilde{f}(x))\right].\end{aligned}$$

The second sum is always zero by (12). Since $\tilde{S}\subseteq S$. The first sum can be further split into two parts,

$$\sum_{\Lambda\in S}\beta^{\Lambda}\left[\sum_{x\in\Omega_\Lambda}(f(x)-\tilde{f}(x))\right]=\sum_{\Lambda\in\tilde{S}}\beta^{\Lambda}\left[\sum_{x\in\Omega_\Lambda}(f(x)-\tilde{f}(x))\right]+\sum_{\Lambda\in S\backslash\tilde{S}}\beta^{\Lambda}\left[\sum_{x\in\Omega_\Lambda}(f(x)-\tilde{f}(x))\right], \tag{14}$$

where once again the first sum is zero. □

Note that this theorems tells us that the error can be expressed as a sum over the β's that we remove when constructing our approximation. Also, note the special case where we assume $\tilde{S}=S\backslash\lambda$, i.e we remove only one interaction β^{λ}. Then,

$$\sum_{x\in\Omega}\left[f(x)-\tilde{f}(x)\right]^2=\beta^{\lambda}\left[\sum_{x\in\Omega_\lambda}(f(x)-\tilde{f}(x))\right]. \tag{15}$$

With these theorems in hand we can go from S to $\tilde{S}$ by removing all nodes in $S\backslash\tilde{S}$. Theorems 1 and 2 allow us to remove these interactions iteratively one at a time. We start by removing the interaction (or one of, in the case of several) β^{λ} with highest degree and approximate it by the set containing all $\Lambda\subset\lambda$. In other words, if the interaction has degree $k=|\lambda|$ we design the $k-1$ order approximation of that interaction term. Grabisch et al. (2000) gives us the expression for this,

$$\tilde{\beta}^{\Lambda}=\begin{cases}\beta^{\Lambda}+(-1)^{|\lambda|-1-|\Lambda|}\left(\frac{1}{2}\right)^{|\lambda|-|\Lambda|}\beta^{\lambda} & \forall\Lambda\subset\lambda,\\ \beta^{\Lambda} & \forall\Lambda\not\subset\lambda.\end{cases} \tag{16}$$

We then proceed iteratively until we reach the set of interest $\tilde{S}$.

Returning to our example in (4), let $\tilde{\tilde{S}}=\{\varnothing,\{1\},\{2\},\{3\},\{1,3\},\{2,3\}\}$, so we want to remove interactions $\{1,2,3\}$ and $\{1,2\}$. We want to approximate $f(x)$ by,

$$\tilde{\tilde{f}}(x_1,x_2,x_3)=\tilde{\tilde{\beta}}^{\varnothing}+\tilde{\tilde{\beta}}^{1}x_1+\tilde{\tilde{\beta}}^{2}x_2+\tilde{\tilde{\beta}}^{3}x_3+\tilde{\tilde{\beta}}^{13}x_1x_3+\tilde{\tilde{\beta}}^{23}x_2x_3. \tag{17}$$

We accomplish this by first removing β^{123} and getting a temporary approximation,

$$\tilde{f}(x_1,x_2,x_3)=\tilde{\beta}^{\varnothing}+\tilde{\beta}^{1}x_1+\tilde{\beta}^{2}x_2+\tilde{\beta}^{3}x_3+\tilde{\beta}^{12}x_1x_2+\tilde{\beta}^{13}x_1x_3+\tilde{\beta}^{23}x_2x_3, \tag{18}$$

and then removing $\tilde{\beta}^{12}$. Removing β^{123} and $\tilde{\beta}^{12}$ is done by calculating the second and first order approximations respectively. We get these directly from (16),

$$\begin{aligned} A_{\tilde{S}}(\beta^{123}x_1x_2x_3) =& \frac{1}{8}\beta^{123} - \frac{1}{4}\beta^{123}x_1 - \frac{1}{4}\beta^{123}x_2 - \frac{1}{4}\beta^{123}x_3 \\ &+ \frac{1}{2}\beta^{123}x_1x_2 + \frac{1}{2}\beta^{123}x_1x_3 + \frac{1}{2}\beta^{123}x_2x_3, \\ A_{\tilde{S}}(\tilde{\beta}^{12}x_1x_2) =& -\frac{1}{4}\tilde{\beta}^{12} + \frac{1}{2}\tilde{\beta}^{12}x_1 + \frac{1}{2}\tilde{\beta^{1}2}x_2. \end{aligned}$$

Thus our full approximative pseudo-Boolean function becomes,

$$\begin{aligned} \tilde{\tilde{f}}(x_1,x_2,x_3) = &\left(\beta^{\varnothing} - \frac{1}{4}\beta^{12}\right) + \left(\beta^{1} + \frac{1}{2}\beta^{12}\right)x_1 + \left(\beta^{2} + \frac{1}{2}\beta^{12}\right)x_2 \\ &+ \left(\beta^{3} - \frac{1}{4}\beta^{123}\right)x_3 + \left(\beta^{13} + \frac{1}{2}\beta^{123}\right)x_1x_3 + \left(\beta^{23} + \frac{1}{2}\beta^{123}\right)x_2x_3. \end{aligned}$$

The next theorem shows us how the approximation error is distributed among the different $x \in \Omega$.

Theorem 5. Given the approximation $A_{\tilde{S}}(f(x)) = \tilde{f}(x)$, when $S \backslash \tilde{S} = \lambda$ the error becomes,

$$f(x) - \tilde{f}(x) = (-1)^{|\lambda| - \sum_{k\in\lambda} x_k} \frac{\beta^{\lambda}}{2^{|\lambda|}}. \tag{19}$$

Proof. Using (16) we can rewrite the error,

$$\begin{aligned} f(x) - \tilde{f}(x) &= \sum_{\Lambda\in S} \beta^{\Lambda} \prod_{k\in\Lambda} x_k - \sum_{\Lambda\in\tilde{S}} \tilde{\beta}^{\Lambda} \prod_{k\in\Lambda} x_k \\ &= \beta^{\lambda} \prod_{k\in\lambda} x_k + \sum_{\Lambda\in\tilde{S}} \left[(\beta^{\Lambda} - \tilde{\beta}^{\Lambda}) \prod_{k\in\Lambda} x_k \right] \\ = \beta^{\lambda} \prod_{k\in\lambda} x_k + \sum_{\Lambda\in\tilde{S}:\Lambda\subset\lambda} &\left[(-1)^{|\lambda|-|\Lambda|} \left(\frac{1}{2}\right)^{|\lambda|-|\Lambda|} \beta^{\lambda} \prod_{k\in\Lambda} x_k \right] \\ &= \beta^{\lambda} \prod_{k\in\lambda} x_k + \frac{\beta^{\lambda}}{2^{|\lambda|}} \sum_{\Lambda\in\tilde{S}:\Lambda\subset\lambda} \left[(-1)^{|\lambda|-|\Lambda|} 2^{|\Lambda|} \prod_{k\in\Lambda} x_k \right]. \end{aligned}$$

Clearly, x is either in Ω_λ or x is in Ω_λ^c. Checking the first case first, $x \in \Omega_\lambda$,

$$\begin{aligned} f(x) - \tilde{f}(x) &= \beta^\lambda + \frac{\beta^\lambda}{2^{|\lambda|}} \sum_{\Lambda \in \tilde{S}: \Lambda \subset \lambda} \left[(-1)^{|\lambda|-|\Lambda|} 2^{|\Lambda|} \right] \\ &= \frac{\beta^\lambda}{2^{|\lambda|}} \sum_{\Lambda \in \tilde{S}: \Lambda \subseteq \lambda} \left[(-1)^{|\lambda|-|\Lambda|} 2^{|\Lambda|} \right] \\ &= \frac{\beta^\lambda}{2^{|\lambda|}} \sum_{|\Lambda|=0}^{|\lambda|} \left[\binom{|\lambda|}{|\Lambda|} (-1)^{|\lambda|-|\Lambda|} 2^{|\Lambda|} \right] \\ &= \frac{\beta^\lambda}{2^{|\lambda|}}, \end{aligned}$$

where we have used that,

$$\sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k = (a+b)^n. \tag{20}$$

If $x \in \Omega_\lambda^c$, then one or more of x_k for $k \in \lambda$ are off. Denote $\Lambda^* \subset \lambda$ as the interaction with the highest degree that remains on, and note that this will be unique. We can then write,

$$\begin{aligned} f(x) - \tilde{f}(x) &= \frac{\beta^\lambda}{2^{|\lambda|}} \sum_{\Lambda \in \tilde{S}: \Lambda \subseteq \Lambda^*} \left[(-1)^{|\lambda|-|\Lambda|} 2^{|\Lambda|} \right] \\ &= \frac{\beta^\lambda}{2^{|\lambda|}} \sum_{|\Lambda|=0}^{|\Lambda^*|} \left[\binom{|\Lambda^*|}{|\Lambda|} (-1)^{|\lambda|-|\Lambda|} 2^{|\Lambda|} \right] \\ &= (-1)^{|\lambda|-|\Lambda^*|} \frac{\beta^\lambda}{2^{|\lambda|}} \sum_{|\Lambda|=0}^{|\Lambda^*|} \left[\binom{|\Lambda^*|}{|\Lambda|} (-1)^{|\Lambda^*|-|\Lambda|} 2^{|\Lambda|} \right] \\ &= (-1)^{|\lambda|-|\Lambda^*|} \frac{\beta^\lambda}{2^{|\lambda|}} \\ &= (-1)^{|\lambda|-\sum_{k\in\lambda} x_k} \frac{\beta^\lambda}{2^{|\lambda|}}. \end{aligned}$$

This proves the theorem. □

To illustrate this result, let $S = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\}$ and $\tilde{S} = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,3\}, \{2,3\}\}$, so $S \backslash \tilde{S} = \{1,2\}$. The distribution of the error is illustrated on the right in Figure 2.

As the absolute value of the error is the same for all states, if we sum the squared error over all states we get,

$$\mathrm{SSE}(f, \tilde{f}) = 2^n \left(\frac{\beta^\lambda}{2^{|\lambda|}} \right)^2. \tag{21}$$

We note that these results are in agreement with Theorem 4 since $|\Omega_\lambda| = 2^{n-|\lambda|}$. For our example in (17) with $S\backslash\tilde{S} = \{\{1,2\},\{1,2,3\}\}$, the errors for removing the nodes will be,

$$\sum_{x\in\Omega}\left(\beta^{123}x_1x_2x_3 - \tilde{A}(\beta^{123}x_1x_2x_3)\right) = \sum_{x\in\Omega}\left(\frac{1}{8}\beta^{123}\right) = 8\left(\frac{\beta^{123}}{8}\right)^2,$$

$$\sum_{x\in\Omega}\left(\tilde{\beta}^{12}x_1x_2 - \tilde{\tilde{A}}(\tilde{\beta}^{12}x_1x_2)\right) = \sum_{x\in\Omega}\left(\frac{1}{4}\tilde{\beta}^{12}\right) = 8\left(\frac{\tilde{\beta}^{12}}{4}\right)^2.$$

And since $\tilde{\beta}^{12} = \beta^{12} + \frac{1}{2}\beta^{123}$ the total error becomes,

$$\sum_{x\in\Omega}\left(f(x) - \tilde{\tilde{f}}(x)\right)^2 = 8\left(\frac{\beta^{123}}{8}\right)^2 + 8\left(\frac{\tilde{\beta}^{12}}{4}\right)^2 = 8\left(\frac{\beta^{123}}{8}\right)^2 + 8\left(\frac{\beta^{12}}{4} + \frac{\beta^{123}}{8}\right)^2. \quad (22)$$

Studying the error gives some insight into what the approximation does. The error from removing each node is spread as evenly as possible among the other states. We can think of the approximation as distributing the interactions we want to remove among the interactions we want to keep.

2.3 Second order interaction removal

In this section we discuss pseudo-Boolean function approximation for a specific choice of $\tilde{S}$, which is of particular interest for the forward-backward algorithm. We show how we can construct a new way of solving the resulting system of equations and term this approximation the second order interaction removal (SOIR) approximation.

Assume we choose $\tilde{S} = S^c_{\{i,j\}}$. In other words we want to remove all interactions involving both i and j and approximate these by all lower order interactions. Using Theorem 1 we can redefine $f(x)$ to contain only the interactions β^Λ where $\Lambda \in S_{\{i,j\}}$, since we only need to focus on the interactions we want to remove. Thus,

$$f(x) = \sum_{\Lambda\in S_{\{i,j\}}}\beta^\Lambda\prod_{k\in\Lambda}x_k \approx \tilde{f}(x) = \sum_{\Lambda\in\tilde{S}}\tilde{\beta}^\Lambda\prod_{k\in\Lambda}x_k, \quad (23)$$

and as before we know that to minimize the error sum of squares, the approximation must fulfill (6). We could of course proceed as in the previous section, iteratively removing one interaction at the time until we reach our desired approximation. We here illustrate a slightly different approach which takes advantage of the particular structure of $\tilde{S}$. This allows us to calculate the approximation even faster than before, and it also, as we will see, gives us an explicit expression for the error. We will see in Section 2.4 how this in turn allows us to construct upper and lower bounds for pseudo-Boolean functions.

We rewrite the error $f(x) - \tilde{f}(x)$,

$$f(x) - \tilde{f}(x) = \sum_{\Lambda \in S_{\{i,j\}}} \beta^{\Lambda} \prod_{k \in \Lambda} x_k - \sum_{\Lambda \in \tilde{S}} \tilde{\beta}^{\Lambda} \prod_{k \in \Lambda} x_k$$

$$= \sum_{\Lambda \in S_{\{i,j\}}} \left[\left(\beta^{\Lambda} x_i x_j - (\tilde{\beta}^{\Lambda \backslash \{i,j\}} + \tilde{\beta}^{\Lambda \backslash \{i\}} x_j + \tilde{\beta}^{\Lambda \backslash \{j\}} x_i) \right) \prod_{k \in \Lambda \backslash \{i,j\}} x_k \right]. \tag{24}$$

The easiest way to convince ourselves that we can always organize the terms in this manner is to observe that every $\Lambda \in S_{\{i,j\}}$ contains both i and j, thus for each of these interactions there will be a unique triplet of interactions in $\tilde{S}$, $\beta^{\Lambda \backslash \{j\}}$ including i but not j, $\beta^{\Lambda \backslash \{i\}}$ including j but not i and $\beta^{\Lambda \backslash \{i,j\}}$ that contains neither. Since $\tilde{S}$ is chosen to be the complement of $S_{\{i,j\}}$ and S is dense, our approximation set contains all these triplets.

To ease the notation we define,

$$\Delta f^{\Lambda}(x_i, x_j) = \beta^{\Lambda} x_i x_j - (\tilde{\beta}^{\Lambda \backslash \{i,j\}} + \tilde{\beta}^{\Lambda \backslash \{i\}} x_j + \tilde{\beta}^{\Lambda \backslash \{j\}} x_i), \ \forall \ \Lambda \in S_{\{i,j\}}. \tag{25}$$

Next we insert our expression for $f(x) - \tilde{f}(x)$ into (6) and switch the order of summation,

$$\sum_{x \in \Omega_\lambda} \left[f(x) - \tilde{f}(x) \right] = \sum_{x \in \Omega_\lambda} \left[\sum_{\Lambda \in S_{\{i,j\}}} \left[\Delta f^{\Lambda}(x_i, x_j) \prod_{k \in \Lambda \backslash \{i,j\}} x_k \right] \right]$$

$$= \sum_{\Lambda \in S_{\{i,j\}}} \left[\sum_{x \in \Omega_\lambda} \left[\Delta f^{\Lambda}(x_i, x_j) \prod_{k \in \Lambda \backslash \{i,j\}} x_k \right] \right]$$

$$= \sum_{\Lambda \in S_{\{i,j\}}} \left[\sum_{x \in \Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}} \Delta f^{\Lambda}(x_i, x_j) \right] = 0, \ \forall \ \lambda \in \tilde{S}. \tag{26}$$

We now proceed to show that we can find a solution satisfying the equations in (26) where each of the sums $\sum_{x \in \Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}} \Delta f^{\Lambda}(x_i, x_j)$ is zero. Obviously for each Λ the function $\Delta f^{\Lambda}(x_i, x_j)$ only has four possible values, $\Delta f^{\Lambda}(x_i = 0, x_j = 0)$, $\Delta f^{\Lambda}(x_i = 1, x_j = 0)$, $\Delta f^{\Lambda}(x_i = 0, x_j = 1)$ and $\Delta f^{\Lambda}(x_i = 1, x_j = 1)$. Thus the sum, $\sum_{x \in \Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}} \Delta f^{\Lambda}(x_i, x_j)$, simply includes each of these values multiplied by the number of times they occur. We now study more closely in what combinations they can occur. Note that obviously $\Lambda \backslash \{i,j\}$ never contains i or j. Consider first the case where λ and thereby $\lambda \cup (\Lambda \backslash \{i,j\})$ does not contain i or j, then,

$$\sum_{x \in \Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}} \Delta f^{\Lambda}(x_i, x_j) = \frac{|\Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}|}{4} (\Delta f^{\Lambda}(x_i = 0, x_j = 0) + \Delta f^{\Lambda}(x_i = 1, x_j = 0)$$

$$+ \Delta f^{\Lambda}(x_i = 0, x_j = 1) + \Delta f^{\Lambda}(x_i = 1, x_j = 1)). \tag{27}$$

Next assume $\lambda \cup (\Lambda \backslash \{i,j\})$ contains i but not j, then,

$$\sum_{x \in \Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}} \Delta f^{\Lambda}(x_i, x_j) = \frac{|\Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}|}{2} \left(\Delta f^{\Lambda}(x_i = 1, x_j = 0) + \Delta f^{\Lambda}(x_i = 1, x_j = 1) \right). \tag{28}$$

Similarly if $\lambda \cup (\Lambda \backslash \{i,j\})$ contains j but not i, then,

$$\sum_{x \in \Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}} \Delta f^{\Lambda}(x_i, x_j) = \frac{|\Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}|}{2} \left(\Delta f^{\Lambda}(x_i = 0, x_j = 1) + \Delta f^{\Lambda}(x_i = 1, x_j = 1) \right). \tag{29}$$

The final case is the case where $\lambda \cup (\Lambda \backslash \{i,j\})$ contains both i and j. However this last instance will never occur in our setting, since any interaction containing i and also j is removed from the graph and thus is not in $\tilde{S}$. We can now reach the conclusion that if we require,

$$\begin{aligned}
&\Delta f^{\Lambda}(x_i = 0, x_j = 0) + \Delta f^{\Lambda}(x_i = 1, x_j = 0) + \\
&\Delta f^{\Lambda}(x_i = 0, x_j = 1) + \Delta f^{\Lambda}(x_i = 1, x_j = 1) = 0, && (30)\\
&\Delta f^{\Lambda}(x_i = 1, x_j = 0) + \Delta f^{\Lambda}(x_i = 1, x_j = 1) = 0, && (31)\\
&\Delta f^{\Lambda}(x_i = 0, x_j = 1) + \Delta f^{\Lambda}(x_i = 1, x_j = 1) = 0, && (32)
\end{aligned}$$

for all $\Lambda \in S_{\{i,j\}}$, the sums in (27), (28) and (29) will all be zero for all $\lambda \in \tilde{S}$ as well. Thus, we have fulfilled equation (26) and found our approximation. There exist a solution that satisfies equations (30), (31) and (32), since, as functions of the parameters β^{Λ}, these are three linearly independent equations and $\Delta f^{\Lambda}(x_i, x_j)$ is a function of three parameters, $\beta^{\Lambda \backslash \{i,j\}}$, $\beta^{\Lambda \backslash \{i\}}$ and $\beta^{\Lambda \backslash \{j\}}$. We once again take a look at our simple example to help illustrate this result. As before let $f(x)$ be defined as in (4) and let our approximation set of interest be $\tilde{S} = \{\varnothing, 1, 2, 3, \{1,3\}, \{2,3\}\}$, so we are removing the second order interaction β^{12}. As we know we only need to focus on the interactions we want to remove, it is sufficient to focus on,

$$f(x) = \beta^{12} x_1 x_2 + \beta^{123} x_1 x_2 x_3. \tag{33}$$

We now reorganize our equations $\sum_{x \in \Omega_\lambda} (f(x) - \tilde{f}(x))$ in the following manner,

$$\begin{aligned}
&\sum_{x \in \Omega_\lambda} f(x) - f(\tilde{x}) \\
&= \sum_{x \in \Omega_\lambda} \left[(\beta^{12} x_1 x_2 - \tilde{\beta}^{\varnothing} - \tilde{\beta}^{1} x_1 - \tilde{\beta}^{2} x_2) + (\beta^{123} x_1 x_2 - \tilde{\beta}^{3} - \tilde{\beta}^{13} x_1 - \tilde{\beta}^{23} x_2) x_3 \right] \\
&= \sum_{x \in \Omega_\lambda} \left[\Delta f^{12}(x_1, x_2) + \Delta f^{123}(x_1, x_2) x_3 \right] \\
&= \sum_{x \in \Omega_\lambda} \Delta f^{12}(x_1, x_2) + \sum_{x \in \Omega_{\lambda \cup 3}} \Delta f^{123}(x_1, x_2),
\end{aligned}$$

which has to be zero for all $\lambda \in \tilde{S}$. For $\Lambda = \{1, 2\}$, (30), (31) and (32) become,

$$\begin{aligned} 4\tilde{\beta}^{\varnothing} + 2\tilde{\beta}^{1} + 2\tilde{\beta}^{2} &= \beta^{12}, \\ 2\tilde{\beta}^{\varnothing} + 2\tilde{\beta}^{1} + \tilde{\beta}^{2} &= \beta^{12}, \\ 2\tilde{\beta}^{\varnothing} + \tilde{\beta}^{1} + 2\tilde{\beta}^{2} &= \beta^{12}. \end{aligned}$$

For $\Lambda = \{1, 2, 3\}$ we get,

$$\begin{aligned} 4\tilde{\beta}^{3} + 2\tilde{\beta}^{13} + 2\tilde{\beta}^{23} &= \beta^{123}, \\ 2\tilde{\beta}^{3} + 2\tilde{\beta}^{13} + \tilde{\beta}^{23} &= \beta^{123}, \\ 2\tilde{\beta}^{3} + \tilde{\beta}^{13} + 2\tilde{\beta}^{23} &= \beta^{123}. \end{aligned}$$

Solving these two systems of equations yields our approximation. The important conclusion from all of this, is that solving our linear system of equations in (26) is equivalent to solving equations (30), (31) and (32). This in turn is equivalent to approximating a second order interaction by its two first order children, and their mutual zero order child. So instead of solving one big system, we can solve a number of very small systems. Thus we can write up what the approximation is in general,

$$\begin{aligned} \tilde{\beta}^{\Lambda \setminus \{i,j\}} &= -\frac{1}{4}\beta^{\Lambda}, \\ \tilde{\beta}^{\Lambda \setminus i} &= \frac{1}{2}\beta^{\Lambda}, \\ \tilde{\beta}^{\Lambda \setminus j} &= \frac{1}{2}\beta^{\Lambda}, \end{aligned} \tag{34}$$

for all $\Lambda \in S_{\{i,j\}}$. This solution corresponds to the solution we would get using the iterative scheme of the previous section, but it is much faster to calculate and also has the advantage of giving us a nice explicit expression for the error. Inserting (34) into (25) we get an expression for the function $\Delta f^{\Lambda}(x_1, x_2)$,

$$\Delta f^{\Lambda}(x_1, x_2) = (x_1 x_2 + \frac{1}{4} - \frac{1}{2}x_j - \frac{1}{2}x_i)\beta^{\Lambda}. \tag{35}$$

Inserting this in (24) we get

$$f(x) - \tilde{f}(x) = (x_i x_j + \frac{1}{4} - \frac{1}{2}x_j - \frac{1}{2}x_i) \sum_{\Lambda \in S_{\{i,j\}}} \left[\beta^{\Lambda} \prod_{k \in \Lambda \setminus \{i,j\}} x_k \right]. \tag{36}$$

Note that the absolute value of the parenthesis outside the sum is always $\frac{1}{4}$ and thus the absolute value of $f(x) - \tilde{f}(x)$ does not depend on x_i or x_j. Using (36) we can also find an expression for the error sum of squares,

$$\mathrm{SSE}(f, \tilde{f}) = \sum_{x \in \Omega} (f(x) - \tilde{f}(x))^2 = \frac{1}{4} \sum_{x \in \Omega_{\{i,j\}}} \left[\sum_{\Lambda \in S_{\{i,j\}}} \beta^{\Lambda} \prod_{k \in \Lambda \setminus \{i,j\}} x_k \right]^2. \tag{37}$$

Another useful observation gained from (36) is that we can easily construct an upper bound for the maximum error,

$$\max_x |f(x) - \tilde{f}(x)| \leqslant \frac{1}{4} \sum_{\Lambda \in S_{\{i,j\}}} |\beta^\Lambda|, \tag{38}$$

which of course also allows us to give a maximum bound for the error sum of squares,

$$SSE(f, \tilde{f}) \leqslant 2^{n-4} \left(\sum_{\Lambda \in S_{\{i,j\}}} |\beta^\Lambda| \right)^2. \tag{39}$$

To help understand the error function in (36) we expand our small example. Assume we expand n from 3 to 4, however we still want $\tilde{S}$ to be $\tilde{S} = S \backslash S_{\{1,2\}}$. Let the graph in Figure 3 represent the states Ω. When we calculate our approximation we then get the following distribution of the error,

$$\begin{aligned}
f(x) - \tilde{f}(x) &= \pm\frac{1}{4}\beta^{12}, \ \forall \ x \in \Omega^0_{\{3,4\}}. \\
f(x) - \tilde{f}(x) &= \pm\frac{1}{4}(\beta^{12} + \beta^{123}), \ \forall \ x \in \Omega^0_{\{4\}} \backslash \Omega^0_{\{3,4\}}. \\
f(x) - \tilde{f}(x) &= \pm\frac{1}{4}(\beta^{12} + \beta^{124}), \ \forall \ x \in \Omega^0_{\{3\}} \backslash \Omega^0_{\{3,4\}}. \\
f(x) - \tilde{f}(x) &= \pm\frac{1}{4}(\beta^{12} + \beta^{123} + \beta^{124} + \beta^{1234}), \ \forall \ x \in \Omega_{\{3,4\}}.
\end{aligned}$$

This is illustrated by the colors in the graph in Figure 3. The four sets above are represented by red, blue, yellow and green respectively. Note also that for each of these sets $f(x) - \tilde{f}(x)$ summed over the respective set is zero, as it should be.

2.4 Upper and lower bounds for pseudo-Boolean functions

In this section we construct upper and lower bounds for pseudo-Boolean functions. These upper and lower bounds will be linked to a given approximation $A_{\tilde{S}}(f(x))$, in the sense that we want our upper and lower bound functions, $f_U(x)$ and $f_L(x)$ respectively, to be on the form,

$$f_U(x) = \sum_{\Lambda \in \tilde{S}} \beta_U^\Lambda \prod_{k \in \Lambda} x_k, \tag{40}$$

and

$$f_L(x) = \sum_{\Lambda \in \tilde{S}} \beta_L^\Lambda \prod_{k \in \Lambda} x_k. \tag{41}$$

In other words, we want the functions to be defined over a given set $\tilde{S}$ similar to the approximations in the previous section. One way of doing this is to start with our approximation and modifying it to get upper and lower bounds. We do this for general approximations

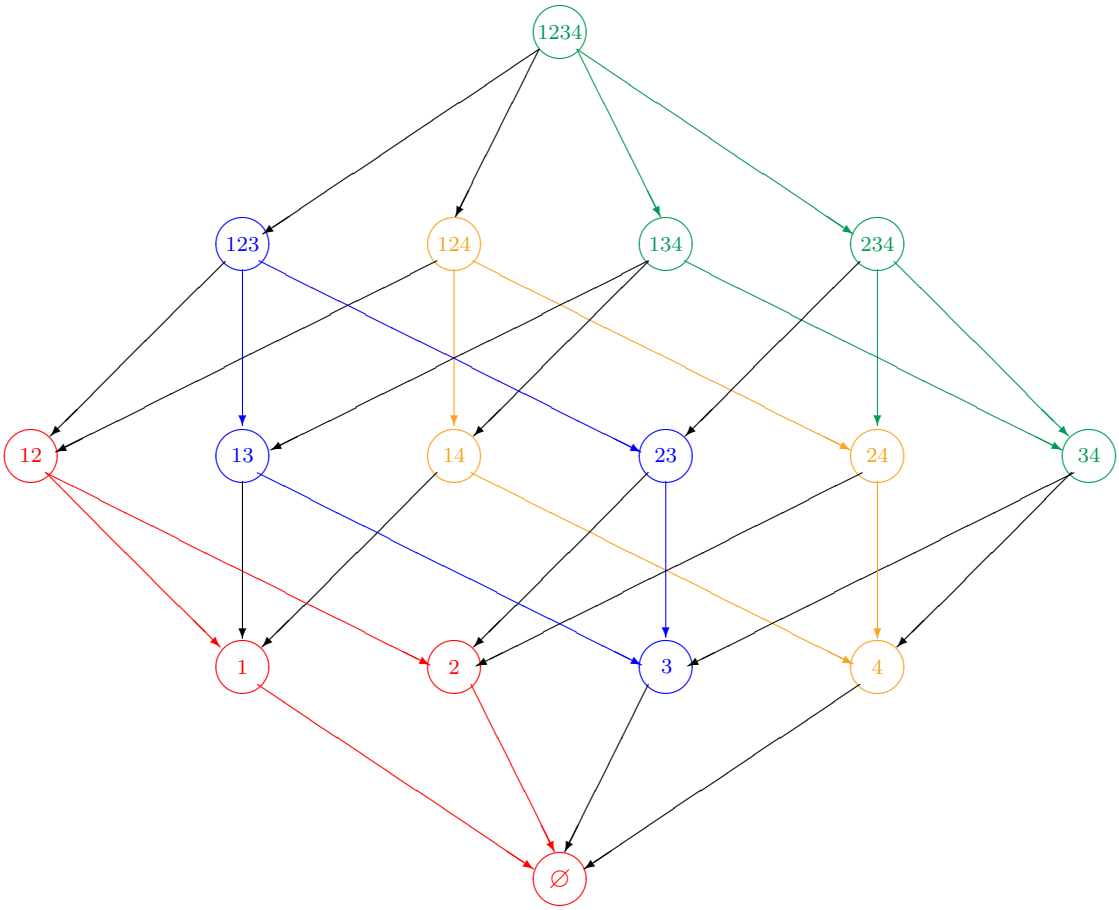

Figure 3: Graph representation of Ω when the dimension of the pseudo-Boolean function is $n = 4$. Colors illustrate the distribution of the absolute value of the error $|f(x) - \tilde{f}(x)|$, when $\tilde{S} = S \backslash S_{\{1,2\}}$. States for which the nodes have the same color have the same absolute value error.

of pseudo-Boolean functions as well as the pairwise interaction approximation described in the previous section.

For a general pseudo-Boolean function we have shown how we can remove interactions iteratively until we reach our approximation set of interest $\tilde{S}$. We have also shown how each time we remove an interaction β^λ we introduce an error in all states $x \in \Omega$,

$$f(x) - \tilde{f}(x) = \pm \frac{\beta^\lambda}{2^{|\lambda|}}. \tag{42}$$

Thus, if we define,

$$f_U(x) = \tilde{f}(x) + \left| \frac{\beta^\lambda}{2^{|\lambda|}} \right|, \tag{43}$$

$$f_L(x) = \tilde{f}(x) - \left| \frac{\beta^\lambda}{2^{|\lambda|}} \right|, \tag{44}$$

we clearly ensure that $f_L(x) \leqslant f(x) \leqslant f_U(x)$ for all $x \in \Omega$. Note that this change only influences the zero order interaction. It corresponds to adding or subtracting a term to the zero order interaction, depending on whether we are constructing upper or lower bounds respectively. This means we are essentially introducing no new computational cost. With this we can construct an iterative scheme just like in Section 2.2, adding or subtracting a term to the zero order interaction for each interaction we remove. In general though, when removing several interactions we would expect to be able to design tighter bounds by looking at the total error after removing all the interactions and then constructing upper and lower bounds rather than iteratively creating upper and lower bounds for each step. We will study how this can be done for the SOIR approximation.

Assume we have an approximation of $f(x)$, $\tilde{f}(x) = A_{\tilde{S}}(f(x))$ with $\tilde{S}$ as defined in Section 2.3. We would like to define our upper and lower bounds as $f_U(x) = \tilde{f}(x) + g(x)$ and $f_L(x) = \tilde{f}(x) + h(x)$, such that $f_L(x) \leqslant f(x) \leqslant f_U(x)$, where $g(x)$ and $h(x)$ are defined over the same set of interactions $\tilde{S}$ as $\tilde{f}(x)$. A natural approach would be to attempt the same technique as we used for the single interaction removal, adding the absolute value of the error $|f(x) - \tilde{f}(x)|$. Taking the absolute value of our expression in (36) we get,

$$\begin{aligned} |f(x) - \tilde{f}(x)| &= \left| (x_i x_j + \frac{1}{4} - \frac{1}{2}x_j + \frac{1}{2}x_i) \sum_{\Lambda \in S_{\{i,j\}}} \left[\beta^\Lambda \prod_{k \in \Lambda \setminus \{i,j\}} x_k \right] \right| \\ &= \frac{1}{4} \left| \sum_{\Lambda \in S_{\{i,j\}}} \left[\beta^\Lambda \prod_{k \in \Lambda \setminus \{i,j\}} x_k \right] \right|, \end{aligned} \tag{45}$$

We could then define, $g(x) = |f(x) - \tilde{f}(x)|$ and $h(x) = -|f(x) - \tilde{f}(x)|$. These are clearly valid upper and lower bounds, and also $|f(x) - \tilde{f}(x)|$ is independent of x_i and x_j, so we will not be introducing any interactions involving both i and j. However there is no guarantee that we can represent $|f(x) - \tilde{f}(x)|$ over our original approximation set $\tilde{S}$.

$|f(x)-\tilde{f}(x)|$ is a pseudo-Boolean function and could in general be of full degree. Thus if the dimension is too high we might not be able to represent it. The dimension of $|f(x)-\tilde{f}(x)|$ is $|\{\Lambda \in S_{\{i,j\}} : |\Lambda| = 3\}|$. In other words $\dim(|f(x)-\tilde{f}(x)|)$ is equal to the number of parents of interaction β^{ij}. We could of course simply expand $\tilde{S}$ to include the necessary missing interactions. Our primary requirement on $\tilde{S}$ was that it should not include any interactions involving i and j which is obviously fulfilled. In the appendix we show how the upper and lower bounds defined by $g(x) = |f(x)-\tilde{f}(x)|$ and $h(x) = -|f(x)-\tilde{f}(x)|$ are in fact optimal if our only requirement on $\tilde{S}$ is that it should not include any interactions involving i and j.

Of course, expanding $\tilde{S}$ could defeat the purpose of our approximative representation altogether. If the dimension of $|f(x)-\tilde{f}(x)|$ is too large we might run into trouble as representing $|f(x)-\tilde{f}(x)|$ will require $2^{\dim(|f(x)-\tilde{f}(x)|)}$ coefficients. It is entirely possible that $|\tilde{S}| > |S|$, which deems the approximative representation worthless. We therefore need to come up with another way of constructing upper and lower bounds. We observe the following,

$$|f(x)-\tilde{f}(x)| = \frac{1}{4}\left|\sum_{\Lambda\in S_{\{i,j\}}}\left[\beta^{\Lambda}\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right]\right| \leqslant \frac{1}{4}\sum_{\Lambda\in S_{\{i,j\}}}\left[|\beta^{\Lambda}|\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right]. \tag{46}$$

So if we define,

$$g(x) = \frac{1}{4}\sum_{\Lambda\in S_{\{i,j\}}}\left[|\beta^{\Lambda}|\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right],$$
$$h(x) = -\frac{1}{4}\sum_{\Lambda\in S_{\{i,j\}}}\left[|\beta^{\Lambda}|\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right],$$

$f_U(x)$ and $f_L(x)$ are clearly valid upper and lower bounds. Also, $g(x)$ and $h(x)$ are already represented by binary polynomials over sets contained within $\tilde{S}$. These are the bounds we apply in the Section 6.

3 MRFs and the forward-backward algorithm

Here we give a short introduction to binary MRFs. We explain how the forward-backward algorithm can be applied to this class of models and point out its computational limitation. For a general introduction to MRFs see Besag (1974) or Cressie (1993) and for more on the properties of MRFs and pseudo-Boolean functions see Tjelmeland and Austad (2012). For more on the forward-backward algorithm and applications to MRFs see Reeves and Pettitt (2004) and Friel and Rue (2007).

3.1 Binary Markov random fields

Assume we have a vector of n binary variables $x = \{x_1, \ldots, x_n\} \in \Omega = \{0,1\}^n$, $N = \{1, \ldots, n\}$. Let $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$ denote the neighborhood system where $\mathcal{N}_k$ denotes the set of indices of nodes that are neighbors of node x_k. As usual we require a symmetrical neighborhood system, so if $i \in \mathcal{N}_j$ then $j \in \mathcal{N}_i$, and by convention a node is not a neighbor of itself. Then x is a binary MRF with respect to a neighborhood system $\mathcal{N}$ if $p(x) > 0$ for all $x \in \Omega$ and the full conditionals $p(x_k|x_{-k})$ have the Markov property,

$$p(x_k|x_{-k}) = p(x_k|x_{\mathcal{N}_k}) \ \forall \ x \in \Omega. \tag{47}$$

where $x_{\mathcal{N}_k} = (x_i : i \in \mathcal{N}_k)$. We define a clique Λ to be a set $\Lambda \subseteq N$ such that for all i and j in Λ, $i \in \mathcal{N}_j$. We say that a clique is a maximal clique if it is not a subset of another clique. The set of all the maximal cliques we denote by $\mathcal{C}$. The Hammersley-Clifford theorem, see Besag (1974) and Clifford (1990), tells us that we can express the distribution of x either through the full conditionals in (47) or through clique potential functions,

$$p(x) = \frac{1}{c}\exp(U(x)) = \frac{1}{c}\exp\left(\sum_{\Lambda \in \mathcal{C}} U_\Lambda(x_\Lambda)\right), \tag{48}$$

where c is a normalizing constant, $U_\Lambda(x_\Lambda)$ is a potential function for a given clique Λ and $x_\Lambda = (x_i : i \in \Lambda)$. $U(x)$ is commonly referred to as the energy function. From the previous section we know that $U(x)$ is a pseudo-Boolean function and can be expressed as,

$$U(x) = \sum_{\Lambda \subseteq N} \beta^\Lambda \prod_{k \in \Lambda} x_k = \sum_{\Lambda \in S} \beta^\Lambda \prod_{k \in \Lambda} x_k, \tag{49}$$

where S is defined as in (3). For a given energy function $U(x)$, the interactions β^Λ can be calculated recursively by evaluating $U(x)$. We will see later that it is important that we only represent the pseudo-Boolean function by the non-zero coefficients, as a full representation would for practical applications require far too many terms. For more details on the relationship between the neighborhood system and the set S we refer the reader to Tjelmeland and Austad (2012). Briefly summarized, x_i and x_j being neighbors is equivalent to there existing at least one $\Lambda \subseteq N$ with $\{i, j\} \subseteq \Lambda$ and $\beta^\Lambda \neq 0$.

3.2 The forward-backward algorithm

As always the problem when evaluating the likelihood or generating samples from MRFs is that c is a function of the model parameters and in general unknown. Calculation involves a sum over 2^n terms,

$$c = \sum_{x \in \Omega} \exp\left(U(x)\right) = \sum_{x \in \Omega} \exp\left(\sum_{\Lambda \in S} \beta^\Lambda \prod_{k \in \Lambda} x_k\right). \tag{50}$$

The forward-backward algorithm, see Reeves and Pettitt (2004) and Friel and Rue (2007), calculates the sum in (50) by taking advantage of the fact that we can calculate this sum

more efficiently by factorizing the un-normalized distribution. We now cover this recursive procedure.

Clearly we can always split the set S into two parts, $S_{\{i\}}$ and $S^c_{\{i\}}$ where as before $S^c_{\{i\}} = S \backslash S_{\{i\}}$. Thus we can split the energy function in (49) into a sum of two sums,

$$U(x) = \sum_{\Lambda \in S^c_{\{i\}}} \beta^\Lambda \prod_{k \in \Lambda} x_k + \sum_{\Lambda \in S_{\{i\}}} \beta^\Lambda \prod_{k \in \Lambda} x_k. \tag{51}$$

Note that the first sum contains no interaction terms involving x_i. Letting $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$, we note that this is essentially equivalent to factorizing $p(x) = p(x_i|x_{-i})\, p(x_{-i})$, since

$$p(x_i|x_{-i}) \propto \exp\left(\sum_{\Lambda \in S_{\{i\}}} \beta^\Lambda \prod_{k \in \Lambda} x_k \right). \tag{52}$$

By summing out x_i from $p(x)$ we get the distribution of $p(x_{-i})$. Taking advantage of the split in (51) we can write this as,

$$p(x_{-i}) = \sum_{x_i} p(x) = \frac{1}{c} \exp\left(\sum_{\Lambda \in S^c_{\{i\}}} \beta^\Lambda \prod_{k \in \Lambda} x_k \right) \sum_{x_i} \exp\left(\sum_{\Lambda \in S_{\{i\}}} \beta^\Lambda \prod_{k \in \Lambda} x_k \right). \tag{53}$$

The sum over x_i can be expressed as the exponential of a new binary polynomial, i.e.

$$\exp\left(\sum_{\Lambda \subseteq \mathcal{N}_i} \check{\beta}^\Lambda \prod_{k \in \Lambda} x_k \right) = \sum_{x_i} \exp\left(\sum_{\Lambda \in S_{\{i\}}} \beta^\Lambda \prod_{k \in \Lambda} x_k \right), \tag{54}$$

where the interactions $\check{\beta}^\Lambda$ are iteratively calculated by evaluating the sum over x_i in (54), see Tjelmeland and Austad (2012). Note that this new function is a pseudo-Boolean function potentially of full degree. The number of non-zero interactions in this representation could be up to $2^{|\mathcal{N}_i|}$. Summing out x_i leaves us with a new MRF with a new neighborhood system. This is the first step in an iterative procedure for calculating the normalizing constant c. In each step we sum over one of the remaining variables by splitting the energy function as above. Repeating this procedure until we have summed out all the variables naturally yields the normalizing constant.

The computational bottleneck for this algorithm occurs when representing the sum in (54). Assume we have summed out variables $x_{1:i-1} = (x_1, \ldots, x_{i-1})$, have an MRF with a neighborhood system $\check{\mathcal{N}} = \{\check{\mathcal{N}}_i, \ldots, \check{\mathcal{N}}_n\}$ and want to sum out x_i. If $\check{\mathcal{N}}_i$ is too large we run into trouble with both memory and computation time when representing the sum corresponding to (54) since this may require up to $2^{\check{\mathcal{N}}_i}$ interaction terms. In models where $\check{\mathcal{N}}_i$ increases as we sum out variables the exponential growth causes us to run into problems very quickly. As a practical example of this consider the Ising model defined on a lattice. Assuming we sum out variables in the lexicographical order, the neighborhood will grow to the number of rows in our lattice. This thus restricts the number of rows in the lattice to < 20 for practical purposes.

4 An approximate forward-backward algorithm

In this section we apply the approximation results of Section 2 to the forward-backward algorithm described in the previous section to devise an approximate forward-backward algorithm. We then show how this approach can be extended to acquire upper and lower bounds for the approximate forward-backward algorithm.

4.1 Constructing the approximate forward-backward algorithm

To create an algorithm that is computationally viable we must seek to control $|\check{\mathcal{N}}_i| = \eta_i$ as we sum out variables. If this neighborhood becomes too large, we run into problems both with memory and computation time. Our idea is to construct an approximate representation of the MRF before summing out each variable. The approximation is chosen so that $\eta_i \leqslant \nu$, where ν is an input to our algorithm. Given a design for the approximation we then want to minimize the error sum of squares of our energy function.

Assume we have an MRF and have (approximately) summed out variables $x_{1:i-1}$, so we currently have an MRF with a neighborhood structure $\check{\mathcal{N}}$ and energy function $\check{U}(x_{i:n}) = \sum_{\Lambda\in\check{S}} \check{\beta}^{\Lambda} \prod_{k\in\Lambda} x_k$, so,

$$c = \sum_{x_{i:n}} \exp\left(\check{U}(x_{i:n})\right). \tag{55}$$

If η_i is too large we run into problems when summing over x_i. Our strategy for overcoming this problem is to first create an approximation of the energy function $\check{U}(x_{i:n})$,

$$\check{U}(x_{i:n}) = \sum_{\Lambda\in\check{S}} \check{\beta}^{\Lambda} \prod_{k\in\Lambda} x_k \approx \tilde{U}(x_{i:n}) = \sum_{\Lambda\in\tilde{S}} \tilde{\beta}^{\Lambda} \prod_{k\in\Lambda} x_k. \tag{56}$$

We control the size of η_i, by designing our approximation set $\tilde{S}$ and thus the new approximate neighborhood $\tilde{\mathcal{N}}$ in such a way that $|\tilde{\mathcal{N}}_i| = \tilde{\eta}_i \leqslant \nu$. Assuming we can do this, we could construct an approximate forward-backward algorithm where we check the size of the neighborhood η_i before summing out each variable. If this is greater than some given ν we approximate the energy function before summation. This leaves two questions; how do we choose the set $\tilde{S}$ and how do we define the approximation?

The two questions are obviously linked, however we start by looking more closely at how we may choose the set $\tilde{S}$. Our tactic is to reduce η_i by one at the time. To do this we need to design $\tilde{S}$ in such a way that i and some node j are no longer neighbors. Doing this is equivalent to requiring all interactions $\tilde{\beta}^{\Lambda}$, involving i and j to be zero. As before we denote the subset of all interactions involving i and j as $\check{S}_{\{i,j\}} \subseteq \check{S}$. We then construct our approximation set as in Section 2.3, defining $\tilde{S} = \check{S}\backslash\check{S}_{\{i,j\}}$. Our approximation is defined by the equations corresponding to (6) and using the results from Section 2.3, the solution is easily available. We can then imagine a scheme where we reduce η_i one at a time until we reach our desired size ν. This leaves the question of how to choose j. One could calculate the SSE for all possibilities of j and choose the value of j that has the minimum SSE. However this may be computationally expensive in some cases. We propose instead to

calculate the upper bound for the maximum error given by (38) and choose j based on the smallest maximum error upper bound. Our experience for all of the models we have studied is that this yields the same choice of j as the true SSE.

Note that Theorem 2 means that after reducing η_i by $\eta_i - \nu$ our approximation is still optimal for a given selection of j's. However there is no guarantee that our selection of j's is optimal. It is possible that if we looked at the error from reducing η_i by more than one at the time we might get a different optimal set of j's.

Using this approximate forward-backward algorithm we are defining an approximate MRF through a series of approximate conditional distributions,

$$\tilde{p}(x) = \tilde{p}(x_1|x_{2:n}) \dots \tilde{p}(x_{n-1}|x_n)\tilde{p}(x_n), \tag{57}$$

which is in fact a new MRF in itself. One of the aspects we wish to investigate in the results section is to what extent this distribution can mimic some of the attributes of the original MRF.

4.2 Bounds for the approximate forward-backward algorithm

It may be of interest to construct upper and lower bounds for the likelihood of an MRF. Acquiring bounds for our algorithm is useful for quantifying the approximation error. Using the results of Section 2.4 we can now easily construct an algorithm for this.

One way of finding an upper bound for the likelihood is to find a lower bound for the normalizing constant. If we can construct $c_L \leqslant c$, then clearly $p_U(x) = \frac{1}{c_L} exp(U(x)) \geqslant \frac{1}{c} exp(U(x)) = p(x)$. Our point of origin for finding c_L is the approximate forward-backward algorithm described in the previous section. Each iteration of this algorithm consists of two steps. In the first step the energy function is replaced by an approximate energy function. In the second step we sum over the chosen variable. To construct upper and lower bounds we simply change step one. Instead of replacing the energy function by an approximation we replace it with the upper and lower bounds found in Section 2.4.

Remark 1. We use the same criteria for determining which second order interaction to remove in each step in the upper and lower bound algorithm as in the approximate forward-backward. Although this was shown to be a good tactic for the approximation approach, there is no reason as to why this should be optimal for constructing upper and lower bounds, but we have been unable to come up with better schemes.

Remark 2. It should be noted that the approximation does not need to remain within the upper and lower bounds. In Section 6.1 we will see examples of this.

5 An approximate Viterbi algorithm

In this section we show how our approximate forward-backward algorithm can be modified to construct an approximate Viterbi algorithm. We briefly discuss the Viterbi algorithm and show how the approximation is constructed as well as find upper and lower bounds.

5.1 Constructing the approximate Viterbi algorithm

The Viterbi algorithm seeks to find a state $x_{\max}$ and its associated value $p(x_{\max})$, with the property that $p(x_{\max}) \geqslant p(x)$ for all $x \in \Omega$ and $p(x_{\max}) = p(x)$ for at least one value of x. Note that $p(x)$ need no longer be a distribution. It relies on the model being factorisable in the same manner as the forward-backward algorithm and proceeds in exactly the same way, except instead of sequentially summing out variables, it takes the maximum. Assume that we have taken the maximum over $x_{1:i-1}$, so we have,

$$\max_{x_{1:i-1}}[p(x)] = \exp(\check{U}_{\max}(x_{i:n})) = \exp\left(\sum_{\Lambda\in\check{S}} \check{\beta}^{\Lambda} \prod_{k\in\Lambda} x_k\right), \tag{58}$$

and now want to take the maximum over x_i i.e, $\max_{x_i}[\exp(\check{U}_{\max}(x_{i:n}))]$. As with the forward-backward algorithm the Viterbi algorithm takes advantage of the splitting of $U(x)$ in (51) to get,

$$\max_{x_i}[\exp(\check{U}_{\max}(x_{i:n}))] = \exp\left(\sum_{\Lambda\in\check{S}^c_{\{i\}}} \check{\beta}^{\Lambda} \prod_{k\in\Lambda} x_k + \max_{x_i}\left[\sum_{\Lambda\in\check{S}_{\{i\}}} \check{\beta}^{\Lambda} \prod_{k\in\Lambda} x_k\right]\right). \tag{59}$$

As with the forward-backward algorithm the max term in the exponential can be represented by a new binary polynomial and the process can be repeated iteratively until we have taken the maximum over all the variables. This yields the maximum value of $p(x_{\max})$. The argument $x_{\max}$ can be found by a backward pass, as in the forward-backward algorithm.

We construct an approximate Viterbi algorithm in the following manner. Recall that our approximate forward-backward algorithm consisted of two steps in each iteration, approximate the energy function and sum over a variable. To get an approximate Viterbi algorithm we simply replace step number two. Instead of summing over a variable we take the maximum over a variable.

5.2 Bounds for the approximate Viterbi algorithm

Just as with the forward-backward algorithm we can use our results for upper and lower bounds for pseudo-Boolean functions to find upper and lower bounds for $p(x_{\max})$. Instead of approximating the energy function before taking the maximum over each variable we replace it with an upper or lower bound. Finding an upper bound for the maximum value of a function in this manner can be of interest for instance for the rejection sampling algorithm as we will see later in Section 6.2

6 Results

In this section we present a number of examples to test our approximation. The value of our algorithm parameter ν obviously influences the time it takes to run the calculations and

how well we approximate the distribution of interest. Our goal in this section is primarily to investigate how this accuracy versus computation time relationship develops, but we also attempt to demonstrate the flexibility of our approximation in handling different types of problems and models, as we feel this is one of the greatest strengths of our approach.

We begin with a simple example using the Ising model, where we use our approximation to evaluate posterior distributions and later calculate upper and lower bounds. We then proceed to use our upper and lower bounds in combination with the approximate Viterbi algorithm to construct a rejection sampling algorithm for the Ising model using our approximation as a proposal distribution. Finally we proceed to a larger example involving reversible-jump Markov chain Monte Carlo (RJMCMC), using a data set of census counts of red deer in the Grampians Region of north-east Scotland.

All examples where run on a machine with an Intel Quad-Core Q9550 2.83GHz cpu.

6.1 Ising model example

In this section we apply our approximation to the Ising model on a square lattice, see Besag (1986). This is an MRF where the energy function can be expressed as,

$$U(x) = \theta \sum_{i \sim j} I(x_i = x_j), \tag{60}$$

where the sum is over all first order neighborhood pairs and θ is a model parameter. $I(x_i = x_j)$ is the indicator function and takes value 1 if $x_i = x_j$ and 0 otherwise. The value of θ controls how strong the interactions are between nodes in the lattice. With a low value of θ we would expect realizations to look noisy, while a high value of θ will give large areas of the same value. Representing the Ising model as a binary polynomial is done by recursively calculating the interactions. This gives us a model with first and second order interactions, for details on how this is done see Tjelmeland and Austad (2012).

The goal of this first example is simply to evaluate how well our approximation works in terms of some measure of accuracy versus run time. To do this we use the following scheme, first we simulate a perfect sample from the Ising model using coupling from the past, see Propp and Wilson (1996), for a given parameter θ_{true}. Then, treating our realization as data we approximate the posterior distribution for θ, $p(\theta|x) \propto p(x|\theta)p(\theta)$, by replacing the likelihood with our approximation for a given value of ν, $\tilde{p}_\nu(x|\theta)$ and using an improper uniform prior. We do this for $\theta_{\text{true}} = 0.4$, 0.6 and 0.8, and for two square lattices of dimensions 15×15 and 100×100, respectively. We let our algorithm parameter ν take values from 2 through to 13. For the 15×15 lattice we can calculate the exact posterior distribution and compare with our approximation. Note that performing an exact evaluation of the posterior for the 15×15 lattice is equivalent to $\nu \geqslant 15$. We calculate the posterior in a regularly spaced mesh of θ values and use interpolating splines to interpolate the results. This is done for both the exact algorithm and using our approximation. We then numerically evaluate the integral

$$D_{15}(\nu, x) = \int_0^\infty |p(\theta|x) - \tilde{p}_\nu(\theta|x)| d\theta, \tag{61}$$

	$\theta = 0.4$	$\theta = 0.6$	$\theta = 0.8$	t
$\epsilon = 0.001$	0.751368101	2.10248497	15.09297	
$\nu = 2$	14.409427206	56.58591888	253.06337	0.01124228
$\nu = 3$	1.108607538	20.62895721	185.87015	0.01387182
$\nu = 4$	0.450648842	10.02418249	159.21355	0.02204446
$\nu = 5$	0.390888291	9.03403984	134.23162	0.03710029
$\nu = 6$	0.539101884	3.09247293	103.32444	0.07013128
$\nu = 7$	0.256395124	2.88674328	70.23284	0.14840200
$\nu = 8$	0.173243954	0.59501500	67.70975	0.32226900
$\nu = 9$	0.079860112	0.25796856	44.78233	0.71145130
$\nu = 10$	0.030442210	0.39202281	41.06546	1.49160500
$\nu = 11$	0.026948158	0.14442855	36.72234	3.89972400
$\nu = 12$	0.021497266	0.01606258	19.77851	9.31453700
$\nu = 13$	0.009940589	0.02520535	12.31064	21.61825000

Table 1: Values of the integrated error $D_{15}(\epsilon = 0.001, x)$ for approximation of Tjelmeland and Austad (2012) and $D_{15}(\nu, x)$ for various values of ν. Associated run times (in seconds) for a single evaluation of the likelihood are given in the column on the right. Note that run time is not given for $D_{15}(\epsilon = 0.001, x)$ since this varies for different values of θ.

to measure how well our approximation works. For the 100×100 lattice an exact evaluation of the posterior is not available, so to evaluate how well the approximation does, we study how it changes as we increase ν by calculating,

$$D_{100}(\nu, x) = \int_0^\infty |\tilde{p}_{\nu+1}(\theta|x) - \tilde{p}_\nu(\theta|x)| d\theta. \tag{62}$$

If this value is sufficiently small we interpret it as a sign that our approximation is close to the exact solution. In Tjelmeland and Austad (2012) the authors demonstrated that their approximation performed well against methods such as pseudo-likelihood and block-pseudo likelihood and was competitive compared with the RDA approximation in Friel et al. (2009). We compare our approximation to the one in Tjelmeland and Austad (2012).

Results for the 15×15 lattice are visualized in Figure 4 and values for $D_{15}(\nu, x)$ presented in Table 1 along with associated run times. Figure 5 and Table 2 present the results for the 100×100 lattice. We also include the approximation from Tjelmeland and Austad (2012), seen as the red dotted curve as seen in Figures 4 and 5. As we would expect a larger value for ν tends to give a better approximation, although it is worth noting that exceptions exist. Also, getting a good approximation is harder for larger values of θ as this means the interactions in the model are stronger. We can see that for $\theta_{\text{true}} = 0.4$, we seem to get quite good approximations even for very small values of ν. The case of $\theta_{\text{true}} = 0.8$ is much harder though, and here only the larger values of ν seem to give good approximations. The run-time plots in the lower right of Figures 4 and 5 nicely illustrate an important difference between the approximation in Tjelmeland and Austad (2012) and

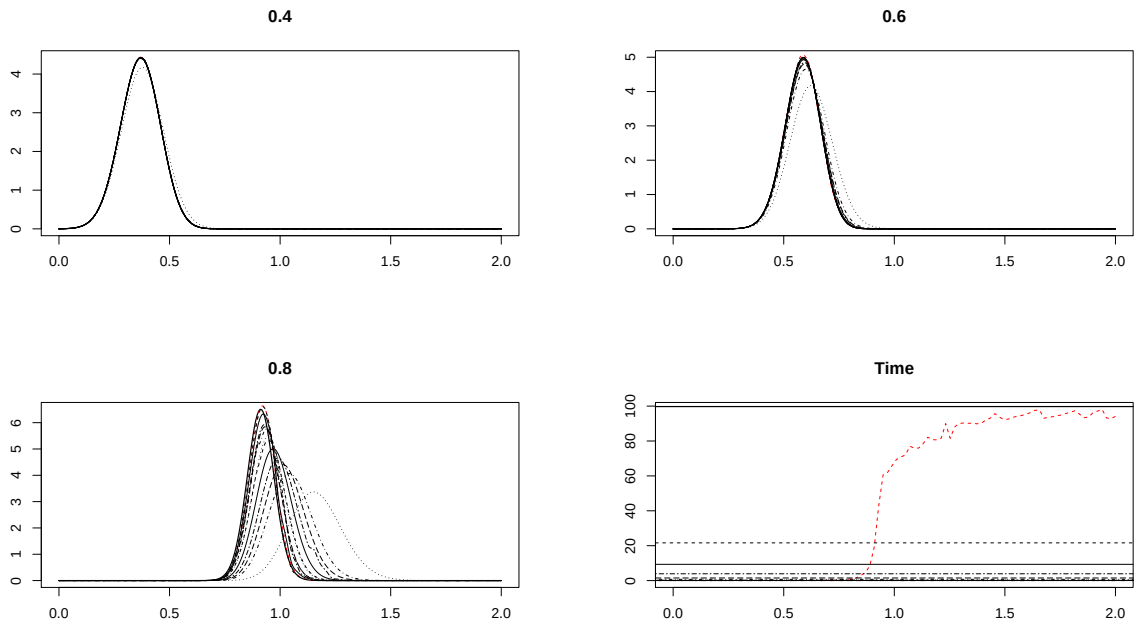

Figure 4: Exact posterior distributions $p(\theta|x)$ (thick continuous black curve), approximations with $\nu = 2$ up to $\nu = 13$ (dashed curves moving from right to left for increasing value of ν) and the approximation of Tjelmeland and Austad (2012)(dashed red curve). Results for $\theta_{\text{true}} = 0.4$ (upper left), $\theta_{\text{true}} = 0.6$ (upper right) and $\theta_{\text{true}} = 0.8$ (lower left) on the 15×15 lattice. Bottom right plot shows the run-times in seconds as a function of θ for the exact run (thick top line), the approximation of Tjelmeland and Austad (2012)(dashed red curve) and the various approximations, $\nu = 2$ to $\nu = 13$, black lines from bottom to top respectively. Note that for $\theta_{\text{true}} = 0.4$ and $\theta_{\text{true}} = 0.6$ the dashed red curve is visually indistinguishable from the other curves.

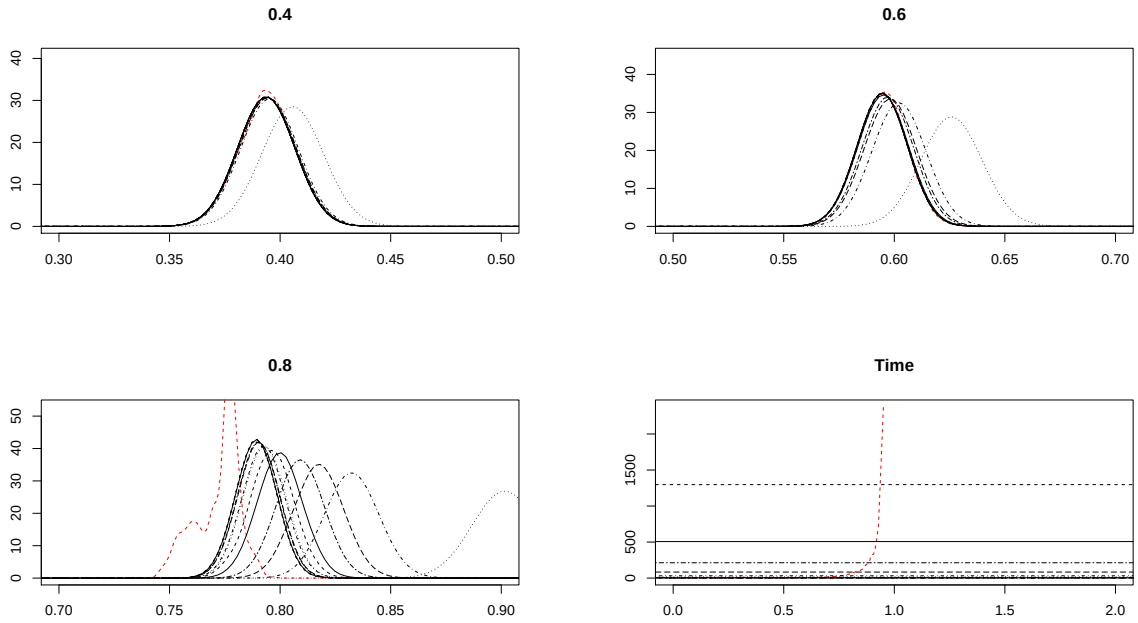

Figure 5: Approximations with $\nu = 2$ up to $\nu = 13$ (dashed curves moving from right to left for increasing value of ν) and the approximation of Tjelmeland and Austad (2012)(dashed red curve). Results for $\theta_{\text{true}} = 0.4$ (upper left), $\theta_{\text{true}} = 0.6$ (upper right) and $\theta_{\text{true}} = 0.8$ (lower left) on the 100×100 lattice. Bottom right plot shows the run-times in seconds as a function of θ for the approximation of Tjelmeland and Austad (2012)(dashed red curve) and the various approximations, $\nu = 2$ to $\nu = 13$, black lines from bottom to top respectively.

	$\theta = 0.4$	$\theta = 0.6$	$\theta = 0.8$	t
$\nu = 2$	88.6347	191.0028	295.8912	1.0001
$\nu = 3$	7.5723	38.5417	143.1385	1.0670
$\nu = 4$	4.2302	21.2625	101.4102	1.4402
$\nu = 5$	1.8341	11.7719	88.5960	2.1802
$\nu = 6$	0.1457	0.3696	44.1358	4.1627
$\nu = 7$	0.5383	5.6674	36.5903	8.5760
$\nu = 8$	0.0604	0.5028	27.2425	18.1769
$\nu = 9$	0.0707	0.2876	6.2459	32.1511
$\nu = 10$	0.03706	0.58005	12.1156	83.6642
$\nu = 11$	0.0031	0.3836	2.8092	198.9024
$\nu = 12$	0.0181	0.2239	1.2727	490.0662

Table 2: Values of the integrated error $D_{100}(\nu, x)$ for various values of ν. Associated run times (in seconds) for a single evaluation of the likelihood are given in the column on the right.

our approximation. As we can see the run-time of $\tilde{p}_\nu(\theta|x)$ is constant as a function of θ. This is because we through ν define the maximum computational cost of running the approximation. This does not change dynamically as θ changes, unlike the approximation in Tjelmeland and Austad (2012). As we can see from the plot, once θ becomes large enough, the run time for the approximation in Tjelmeland and Austad (2012) explodes. This is due to the interactions becoming so strong that the approximation is unable to remove any. Although this is a nice property in the sense that it allows the approximation to adapt dynamically to the model, it does mean that there are models for which the algorithm will not work. It also means that it can be hard to predict the run-time in advance. We note that the run-times for the new approximation roughly goes as 2^ν. This is to be expected since increasing ν by one doubles the size of our approximate pseudo-Boolean energy function. Our approximation also seems to do considerably better than the approximation in Tjelmeland and Austad (2012) for the case of $\theta_{\text{true}} = 0.8$ on the 100×100 lattice. Here the approximation in Tjelmeland and Austad (2012) breaks down, while our approximation seems to work satisfactorily.

We can also use our approximation to get estimates of the normalizing constant for the Ising model. Doing this on a lattice of θ values and using interpolating splines we can get a smooth approximation of c as a function of θ, $\tilde{c}_\nu(\theta)$. For the 15×15 lattice we can then compare our approximation to the truth. This can be particularly useful in illustrating how well the approximation works for increasing values of θ. The log ratio, $\log\left(\frac{c(\theta)}{\tilde{c}_\nu(\theta)}\right)$, as function of θ is plotted in Figure 6 for $\nu = 2$ up to $\nu = 13$ as well as for the approximation in Tjelmeland and Austad (2012) $\tilde{c}_\epsilon(\theta)$, again represented as a red dotted curve. This gives a decent indication of where the approximation works well. Note how the approximation in Tjelmeland and Austad (2012) converges to $0.6931 = \log(2)$ as θ increases. This happens

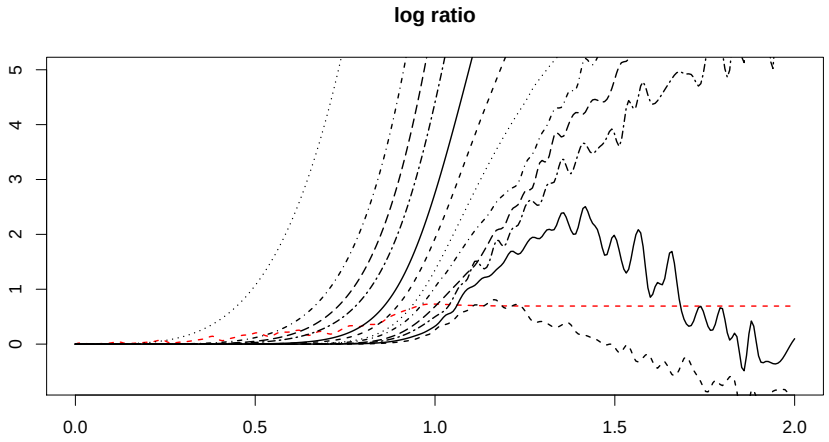

Figure 6: $\log(c(\theta)) - \log(\tilde{c}_\nu(\theta))$ for $\nu = 2$ up to $\nu = 13$) (black dotted curves moving from left to right for increasing value of ν) as well as $\log(c(\theta)) - \log(\tilde{c}_\epsilon(\theta))$ for $\epsilon = 0.001$ (dotted red curve). Results are shown for values of θ between 0 and 2 on a 15×15 lattice.

because a high θ means all the probability is focused on the configurations where all nodes are either 1 or 0, and the approximation puts all probability on the configuration where all nodes are 0. We also notice that it seems like our approximation usually supplies an underestimate of c.

Finally we have tested the upper and lower bounds derived in Section 2.4. As with the approximation we have generated a new realization from the model which we treat as data using $\theta_{\text{true}} = 0.4$, 0.6 and 0.8. We calculate upper and lower bounds for the normalizing constant $c_U(\theta)$ and $c_L(\theta)$ for a fine mesh of θ's between 0 and 2. Then for each of the values of θ_{true} we can calculate the un-normalized likelihood and combine with $c_U(\theta)$ and $c_L(\theta)$ to get upper and lower bounds for the likelihood $p_U(\theta|x)$ and $p_L(\theta|x)$. We have done this for $\nu = 6$, 10 and 13 and the results are plotted in Figure 7. The stapled curves show the upper and lower bounds for $\log(p(\theta|x))$ while the continuous curve represents the associated approximative log-likelihood, $\log(\tilde{p}_\nu(\theta|x))$. As we can see we get reasonably tight bounds for $\theta_{\text{true}} = 0.4$ and 0.6 while for the case of $\theta_{\text{true}} = 0.8$ the bounds are wider. Note that for larger values of θ in the $\theta_{\text{true}} = 0.8$ case, the approximation does not lie inside the upper and lower bound. We could imagine using these bounds to get intervals for a maximum likelihood estimator or the mode of the posterior distribution. Since we know that $\max_\theta\{p(\theta|x)\} \geqslant \max_\theta\{p_L(\theta|x)\}$ and since $p(\theta|x) \leqslant p_U(\theta|x)$ for all x we get an interval which we know must contain the true maximum. We could imagine a scheme where we start with a small value of ν to get a wide interval for the maximum. We then increase the value of ν and get upper and lower bounds within this interval and use this to shrink the interval. This is then repeated until the interval is sufficiently tight.

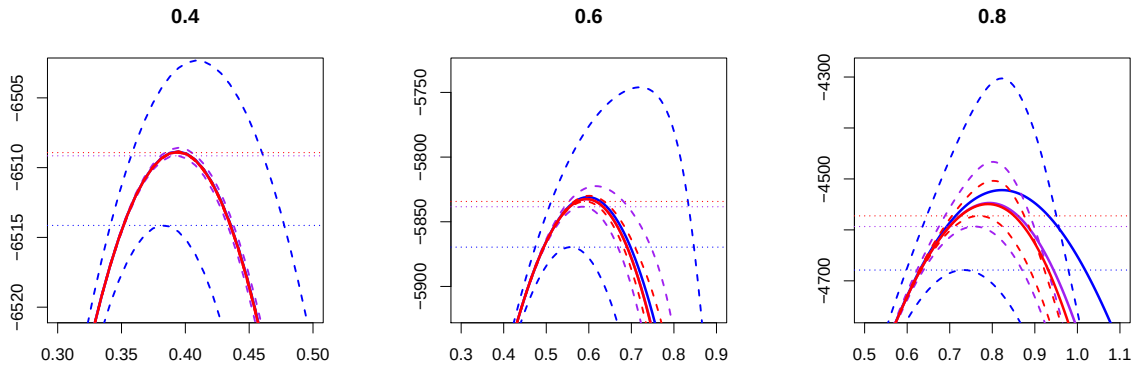

Figure 7: Approximate log-likelihood functions (continuous curves) for $\nu = 6$ (blue), $\nu = 10$ (purple) and $\nu = 13$ (red), for $\theta_{\text{true}} = 0.4$ (left), $\theta_{\text{true}} = 0.6$ (middle) and $\theta_{\text{true}} = 0.8$ (right) for the Ising model on a 100×100 lattice. Upper and lower bounds are given by the stapled curves.

6.2 Rejection sampling example

In Section 5.2 we showed how we can find upper and lower bounds for the mode of a function by combining our upper and lower bounds for pseudo-Boolean functions with the Viterbi algorithm. With this in hand we can construct a rejection sampling algorithm to generate perfect samples from a given distribution. We should point out that there are several perfect samplers out there, many of which perform better than the algorithm we are about to present, in particular for models like the Ising model. Still, however, we think the rejection sampling algorithm is an interesting application of the approximate Viterbi algorithm. In particular, our perfect sampler may be of interest for settings where a large number of samples are required. Once the initial precomputations are done, generating samples is done very quickly using our algorithm. For a detailed description of the rejection sampling algorithm we refer the reader to Ripley (1987). A short summary goes as follows. Assume we wish to sample from a distribution $p(x) = \frac{1}{c}\exp(U(x))$. We generate a proposal using a proposal distribution $q(x)$ and calculate the acceptance probability,

$$\alpha(x) = \frac{1}{k}\frac{p(x)}{q(x)} = \frac{1}{ck}\frac{\exp(U(x))}{q(x)} = \frac{1}{k^*}\frac{\exp(U(x))}{q(x)}, \tag{63}$$

where $k^* = ck$ is chosen such that $k^* \geqslant \frac{exp(U(x))}{q(x)}$ for all x. We then accept the proposal with probability $\alpha(x)$. This is repeated until we reach the desired number of samples. The difficulty is of course to find a sufficiently tight bound k^* to get an acceptable acceptance rate. Choosing our approximation as our proposal distribution, $q(x) = \tilde{p}_\nu(x)$, we need to find k^*, such that $k^* \geqslant r(x) = \frac{exp(U(x))}{\tilde{p}_\nu(x)}$. Using our algorithm for upper and lower bounds for the Viterbi algorithm in Section 5.2 we can find $k^* = \tilde{r}_U(\tilde{x}_{max}) \geqslant r(x_{max})$.

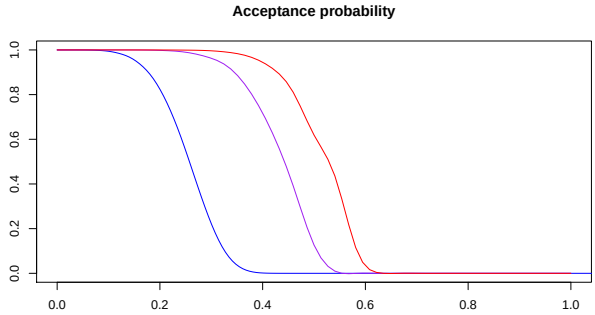

Figure 8: Estimated acceptance probabilities as a function of θ for the rejection sampling algorithm when sampling from an Ising model on a 100×100 lattice. Three values for ν were tested, $\nu = 6$ (blue), $\nu = 10$ (purple) and $\nu = 13$ (red).

Two things are important for us to achieve a high acceptance rate. Firstly the function $r(x)$ must be reasonably "flat", i.e $r(x_{max}) - r(x)$ should be reasonable small for all x, and second, our upper bound needs to be sufficiently tight. We would expect $r(x)$ to be reasonably constant as a function of x since, $r(x) = \frac{1}{\tilde{c}} exp(U(x) - \tilde{U}(x))$. We have pointed out how our approximation attempts to spread the error $U(x) - \tilde{U}(x)$ evenly among all states x, thus we would expect this function to be reasonably uniform. We have tested our perfect sampler by using it to generate samples from the Ising model on a 100×100 lattice for a fine mesh grid of θ values between 0 and 1. For each value of θ we generated 100 proposals and used this to estimate the acceptance rate. Once again we did this for $\nu = 6$, 10 and 13) and plotted the average acceptance rates as a function of θ, as seen in Figure 8. As expected we can see that the acceptance rate drops as θ increases. With $\nu = 13$ we get a high acceptance rate almost up to $\theta = 0.6$. We believe the reason the acceptance rate drops is primarily because the bounds for $r(x)$ become to weak.

6.3 Red Deer example part 1

In this section we present a Bayesian analysis of a data set of census counts of red deer in the Grampians Region of north-east Scotland. Our primary purpose for including this section is to demonstrate the flexibility and applicability of our approximation and as such this will not be a full analysis of the given data set. A full description of the data set can be found in Augistin et al. (1996) and Buckland and Elston (1993).

The data are presented in Figure 9 and represent presence or absence of red deer. A lattice has been laid over the region of interest and the data reduced to presence or absence in each of our n grid cells. We denote the data $y = \{y_1, \ldots, y_n\}$ and let $y_i = 1$ indicate presence and $y_i = 0$ indicate absence of deer. In each location i we have four covariates denoted z_{ij}, $j = 1, .., 4$. These are altitude and mires, as seen in Figure 9, and Cartesian coordinates easting and northing respectively. These have all been standardized to have

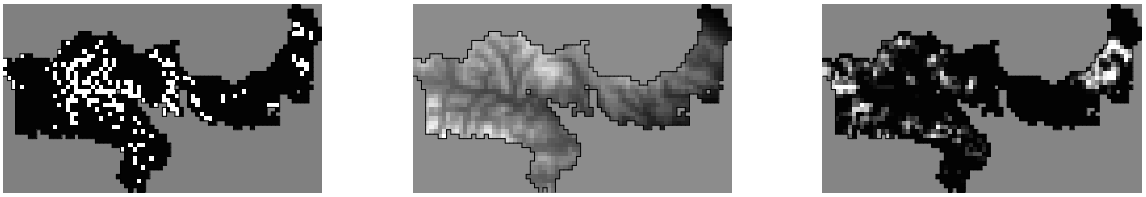

Figure 9: Red deer data set. Left plot shows presence/absence of reed deer, white indicating presence ($y_i = 1$) and black representing absence ($y_i = 0$). Middle plot shows altitude covariate, white indicating high altitude and black indicating low altitude. Right plot shows mires covariate, again with white indicating a high value and black indicating a low value.

zero mean and unit standard deviation.

Our goal is to do model choice as well as investigate posteriors for this data set. All four covariates can be either included or excluded from our model giving 2^4 possible combinations. We also consider 2 spatial models as well. The first model, denoted S_1 is an MRF with a 2×2 clique, giving us the following likelihood function,

$$p_1(y|\theta^{S_1}, \theta^C, z) = \frac{1}{c} \exp\left(\sum_{C \in \mathcal{C}} U_C(y_C, \theta^{S_1}) + \sum_{i=1}^{N} \sum_{j=1}^{4} z_{ij} \theta_j^C \right), \tag{64}$$

where θ^{S_1} are our spatial parameters, $\theta^C = (\theta_1^C, \theta_2^C, \theta_3^C, \theta_4^C)$ are our covariate parameters and $U_C(y_C, \theta^{S_1})$ assigns the associated clique potential to the configuration y_C. Assuming our clique potentials to be translation and rotation invariant we get 6 classes of clique configurations, see Figure 10. We define $\theta_6^S = 0$, this then leaves us with 5 spatial parameters, $\theta^{S_1} = (\theta_1^{S_1}, \theta_2^{S_1}, \theta_3^{S_1}, \theta_4^{S_1}, \theta_5^{S_1})$. The second spatial model S_2, is the Ising model with a trend term, giving us the following likelihood function,

$$p_2(y|\theta^{S_2}, \theta^C, z) = \frac{1}{c} \exp\left(\frac{\theta_1^{S_2}}{2} \sum_{i \sim j} I(y_i, y_j) + \theta_2^{S_2} \sum_{i=1}^{n} y_i + \sum_{i=1}^{N} \sum_{j=1}^{4} z_{ij} \theta_j^C \right). \tag{65}$$

Excluding a covariate from the model is obviously equivalent to setting $\theta_j^C = 0$. To explore the posterior distribution as well as decide which model best fits the data we use a reversible jump Markov chain Monte Carlo algorithm (RJMCMC), see Green (1995). Following Tjelmeland and Austad (2012), we adopt wide independent priors for our parameters. For the components of θ^{S_1} and θ^C and for $\theta_2^{S_2}$ we use independent normal priors with zero mean and variance 20. For $\theta_1^{S_2}$ we use a gamma prior with mean $\frac{2}{3}$ and variance $\frac{2}{9}$. Thus our posterior for S_1 becomes,

$$p_1(\theta^{S_1}, \theta^C|y, z) \propto p_1(y|\theta^{S_1}, \theta^C, z) \prod_{i=1}^{5} p(\theta_i^{S_1}) \prod_{i=1}^{4} p(\theta_i^C), \tag{66}$$

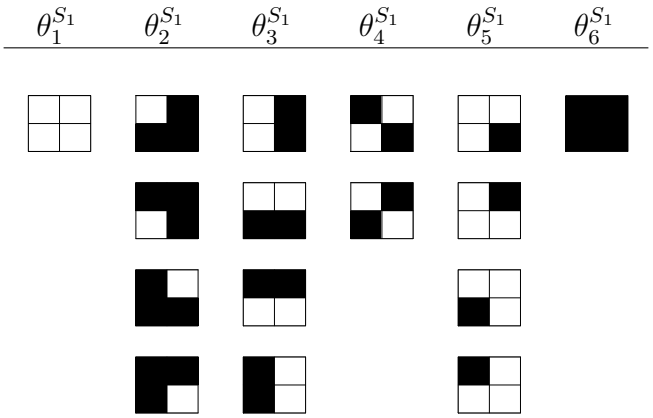

Figure 10: Classes of clique configurations for a translation and rotation invariant 2×2 clique. Each column includes the configurations of the clique represented by the associated parameter. Each gridcell is either 1 (white) or 0 (black).

where $p(\theta_i^{S_1})$ and $p(\theta_i^C)$ are the normal priors defined above. For S_2 our posterior becomes,

$$p_2(\theta^{S_2}, \theta^C | y, z) \propto p_2(y | \theta^{S_2}, \theta^C, z) p_g(\theta_1^{S_2}) p(\theta_2^{S_2}) \prod_{i=1}^{4} p(\theta_i^C), \tag{67}$$

where $p_g(\theta_1^{S_2})$ is the gamma prior defined above. To evaluate the likelihood we replace $p_1(y|\theta^{S_1}, \theta^C, z)$ and $p_2(y|\theta^{S_2}, \theta^C, z)$ by our approximations $\tilde{p}_1(y|\theta^{S_1}, \theta^C, z)$ and $\tilde{p}_2(y|\theta^{S_2}, \theta^C, z$). In each iteration of our sampler we perform one of four proposals. With probability 0.2, we remove one of the currently active covariates j, assuming all covariates are not off. The second proposal, again with probability 0.2, is to activate one of the currently inactive covariates j, assuming that all covariates are not on. The third proposal does not propose to change the model, but only change one of the parameters. With probability 0.5 we propose to change one of the spatial parameters or one of the covariate parameters (chosen uniform at random) by adding to the current value a value u drawn from a normal distribution with zero mean and variance 0.1^2. The last proposal, with probability 0.1, is to propose to switch spatial model from S_k to $S_{|k-1|}$. When we add a new covariate we propose a new value for the covariate by sampling from the prior. Likewise, when we switch spatial models we propose new values for the spatial parameters by sampling from the priors. We ran the algorithm for 200000 iterations, repeating the run for different starting values of parameters and different starting models. To ensure that our approximation was not influencing results too much we used different values of ν and compared the results. Testing values of ν up to 10, we found that for $nu \geqslant 6$ there was no discernable difference in the results. The results presented from here on are for $\nu = 6$. In our runs we found that the algorithm would never switch from the MRF model to Ising model, but when started with the Ising model would switch to the MRF model and not switch back. We thus concluded that the data clearly prefer spatial model S_1 and re-ran the algorithm using only spatial

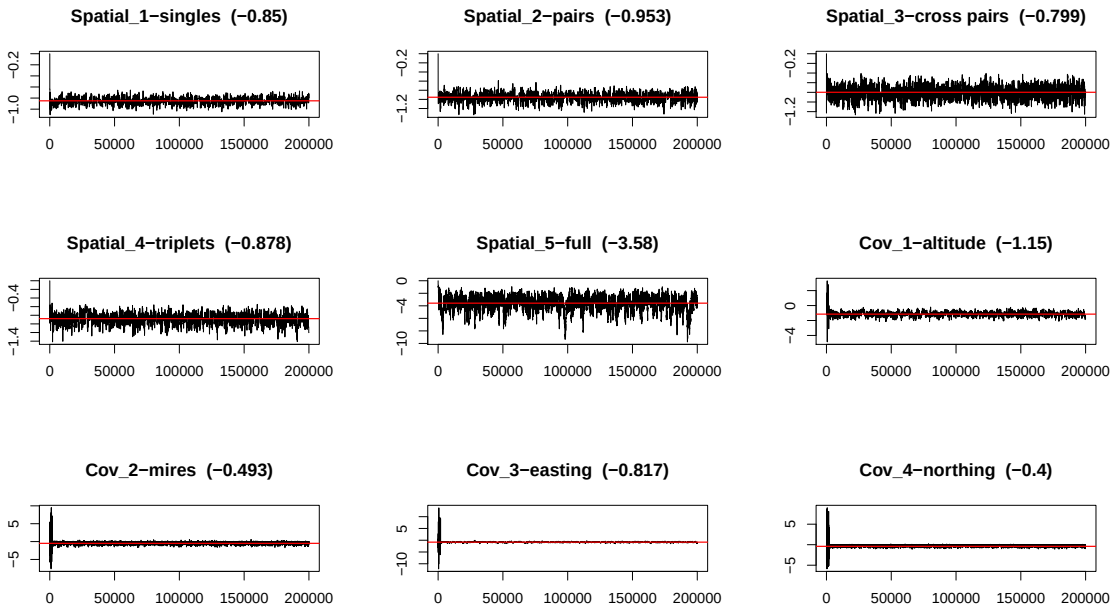

Figure 11: Trace plot for a RJMCMC run using spatial model S_1, from top to bottom, left to right, the first five plots show the spatial parameters $(\theta_1^{S_1}, \theta_2^{S_1}, \theta_3^{S_1}, \theta_4^{S_1}, \theta_5^{S_1})$ followed by covariate parameters $(\theta_1^C, \theta_2^C, \theta_3^C, \theta_4^C)$ for altitude, mires easting and northing respectively. Average value after a burn-in of 50000 iterations is represented by the red line and listed next to the parameter names.

model S_1. Trace plots from one such run can be seen in Figure 11. The acceptance rate for the run presented in Figure 11 was 0.193. Counting the number of occurrences of each covariate set we found that four different models made up 0.99 of the occurrences, they can be seen in Table 3. To see how well our model fits the data it can be interesting to sample sets of parameters from our RJMCMC run and use these parameters to simulate from the model in (64). We generated 9 such simulations, presented in Figure 12. As we can see, the model seems to capture the spatial patterns present in the data. The slight clumping as seen in the data seems to be present in the simulations as well. We also note, both from the simulations in Figure 12 and the trace plots in Figure 11 that the spatial terms of the model seem to catch much of the information in the data. If the covariates were more important we would expect the simulations to more closely resemble the data in where the deer occur. We ran the RJMCMC chain several times and these runs indicate a convergence after approximately 50000 iterations. The results presented in this report represent the last run that was performed.

Altitude	Mires	Easting	Northing	Frequency
×	×			0.440
×		×	×	0.359
×	×	×		0.099
×	×	×	×	0.091
Sum				0.990

Table 3: Frequency of occurrences for the four most occurring covariance sets. An × indicates that the associated parameter is present in the model.

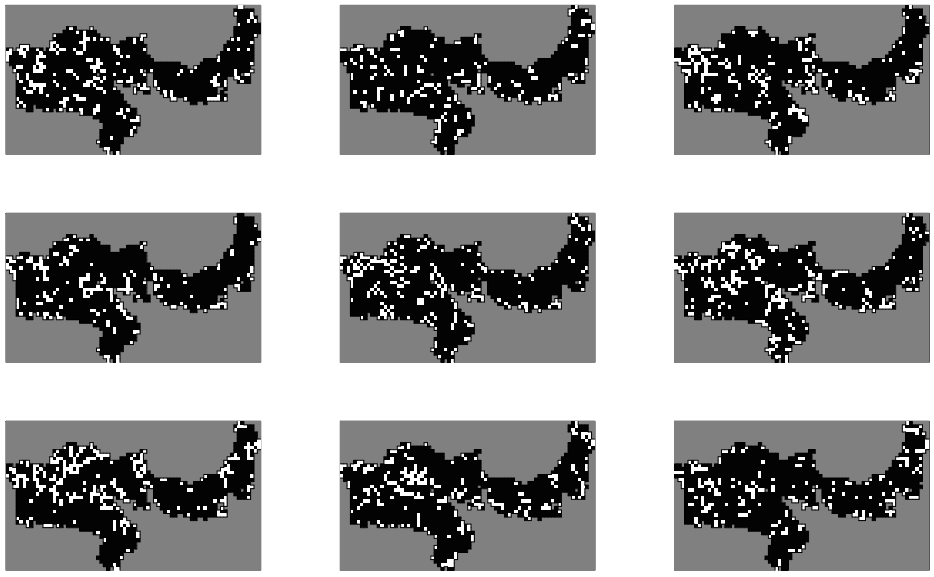

Figure 12: 9 realizations from $p_2(y|\theta^{S_2}, \theta^C, z)$ for values of θ^{S_2} and θ^C sampled from the RJMCMC run (discounting a burn-in of 50000 iterations).

6.4 Red Deer example part 2

As a further test of our approximation we also considered another approach to the data. We can view our data y, not as true locations of red deer, but merely observations of some latent field x representing the true presence or absence of red deer. Pursuing this line of thought we introduce a probability $\mathcal{P}$ of observing deer, given the presence of deer, so $p(y_i = 1|x_i = 1) = \mathcal{P}$ and subsequently $p(y_i = 0|x_i = 1) = 1 - \mathcal{P}$. Note that if deer is not present there is no chance that any will be observed, so $p(y_i = 1|x_i = 0) = 0$ and $p(y_i = 0|x_i = 0) = 1$. We model x as an MRF,

$$p(x|\theta^{S_1}, \theta^C, z) = \frac{1}{c} \exp\left(\sum_{C\in\mathcal{C}} U_C(x_C, \theta^{S_1}) + \sum_{i=1}^{N}\sum_{j=1}^{4} z_{ij}\theta_j^C\right), \tag{68}$$

while the distribution of y now conditional on x becomes,

$$p(y|x, \mathcal{P}) = \prod_{i=1}^{n} p(y_i|x_i, \mathcal{P}). \tag{69}$$

Our interest is still in the posterior distribution of the parameters given the data, were $\mathcal{P}$ is now a new parameter to which we assign a uniform prior. We can write the posterior as,

$$p(\theta^{S_1}, \theta^C, \mathcal{P}|y, z) \propto \frac{p(y|x, \mathcal{P})p(x|\theta^{S_1}, \theta^C, z)\prod_{i=1}^{5} p(\theta_i^{S_1})\prod_{i=1}^{4} p(\theta_i^C)}{p(x|y, \theta^{S_1}, \theta^C, \mathcal{P}, z)}. \tag{70}$$

The distribution $p(x|\theta^{S_1}, \theta^C, z)$ is the MRF in (68) while $p(x|y, \theta^{S_1}, \theta^C, \mathcal{P}, z)$ is essentially an MRF conditional on observations. Since we could not observe deer if no deer were actually present, if $y_i = 1$, then x_i must be 1 as well. This means a number of x_i's are no longer considered variables, but instead set equal to 1. Clearly,

$$\begin{aligned} p(x|y, \theta^{S_1}, \theta^C, \mathcal{P}, z) &\propto p(x|\theta^{S_1}, \theta^C, z)p(y|x, \mathcal{P}) \\ &= \exp\left(\sum_{C\in\mathcal{C}} U_C(x_C, \theta^{S_1}) + \sum_{i=1}^{N}\sum_{j=1}^{4} z_{ij}\theta_j^C + \sum_{i=1}^{n} \log(p(y_i|x_i, \mathcal{P}))\right), \end{aligned}$$

where we note that this includes a normalizing term dependent on y. As before we apply our approximation to get an approximate posterior,

$$\tilde{p}(\theta^S, \theta^C, \mathcal{P}|y, z) \propto \frac{p(y|x, \mathcal{P})\tilde{p}(x|\theta^{S_1}, \theta^C, z)\prod_{i=1}^{5} p(\theta_i^{S_1})\prod_{i=1}^{4} p(\theta_i^C)}{\tilde{p}(x|y, \theta^S, \theta^C, \mathcal{P}, z)}. \tag{71}$$

We should note the choice for x when evaluating this. Obviously if we did an exact evaluation the choice of x would not matter, however since the error of our approximation might be different for different values of x, the choice of x could influence the results. In our algorithm we have simply generated a realization of x from $\tilde{p}(x|y, \theta^S, \theta^C, z)$, using our approximation, and used this value when evaluating the posterior. Our experience is that

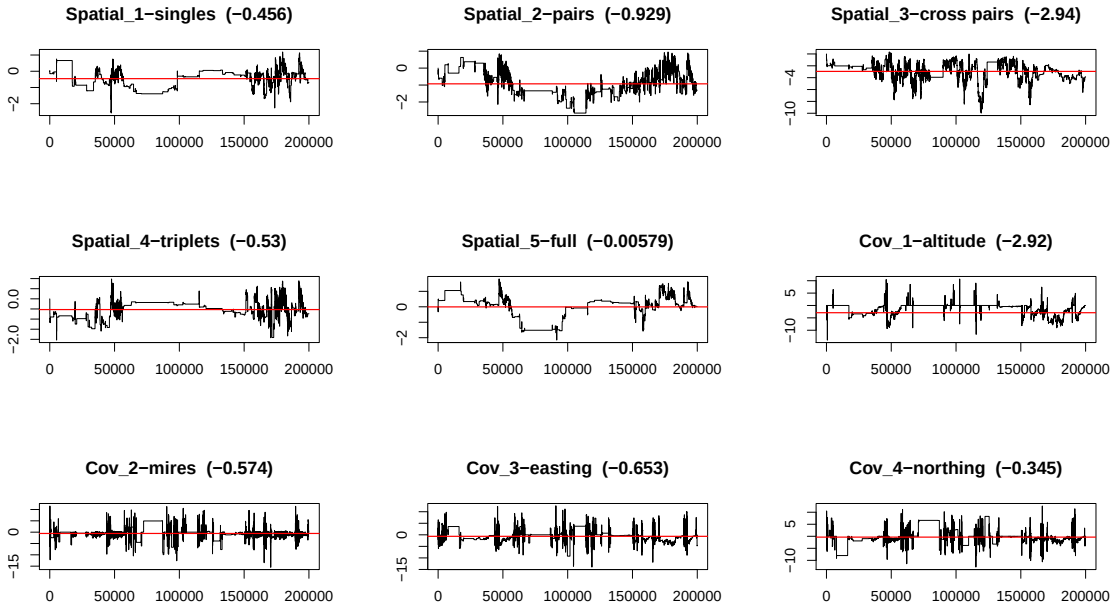

Figure 13: Trace plot for a RJMCMC run using spatial model S_1 and including the observation probability $\mathcal{P}$, from top to bottom, left to right, the first five plots show the spatial parameters $(\theta_1^{S_1}, \theta_2^{S_1}, \theta_3^{S_1}, \theta_4^{S_1}, \theta_5^{S_1})$ followed by covariate parameters $(\theta_1^C, \theta_2^C, \theta_3^C, \theta_4^C)$ for altitude, mires easting and northing respectively. Average value after a burn-in of 50000 iterations is represented by the red line and listed next to the parameter names.

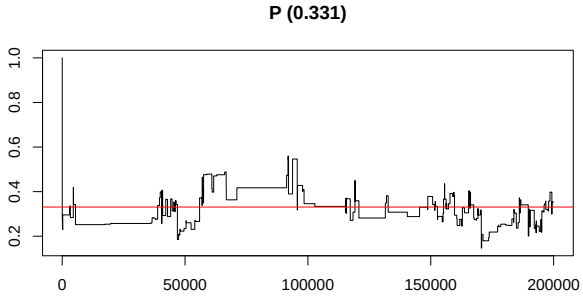

Figure 14: Trace plot for $\mathcal{P}$ for RJMCMC run using spatial model S_1 with probability of observing deer $\mathcal{P}$.

Altitude	Mires	Easting	Northing	Frequency
×	×	×	×	0.230
×		×	×	0.127
		×	×	0.121
×	×		×	0.120
Sum				0.598

Table 4: Frequency of occurrences for the four most occurring covariance sets using the observational probability $\mathcal{P}$. An × indicates that the associated parameter is present in the model.

this works quite well. As before we ran the algorithm for 200000 iterations, trace plots for the spatial and covariate parameters can be seen in Figure 13, while a trace plot for $\mathcal{P}$ can be seen in Figure 14 The acceptance rate for the run presented in Figures 13 and 14 was 0.079. A count of the top four models is presented in Table 4. The top four models made up a total of 0.598 of the model occurrences. As in the previous section we tested how well our model fits the data by sampling sets of parameters from our RJMCMC run and using these parameters to simulate from our model. This was done by first simulating x from $p(x|\theta^S, \theta^C, z)$ and then simulating y from $p(y|x, \mathcal{P})$. We generated nine such simulations, presented in Figure 15. As we can see from the trace plots and the lower acceptance rate, the introduction of $\mathcal{P}$ makes it harder for our algorithm to move around in the distribution. There also seems to be a greater variance in the parameters. This could be attributed to $\mathcal{P}$ settling around such a low value (0.331). A low value for $\mathcal{P}$ means more uncertainty around our observations and allows for a greater variation in the fitted model. This is also reflected in the model choice part of our algorithm as the four top models only made up about 60% of the occurrences. The realizations in Figure 15 do not seem entirely unreasonable, so the model is capturing some of the information in the data. The low value of $\mathcal{P}$ might seem a bit unrealistic, so one should perhaps reconsider the model. Again however, our primary purpose with this example is not to do a full analysis, but demonstrate applications.

7 Closing remarks

In this report we have shown how we can derive an approximate forward-backward algorithm by studying how to approximate the pseudo-Boolean energy function during the summation process. This approximation can then be used to work with statistical models such as MRFs. It allows us to produce approximations of the normalizing constant and likelihood as well as realizations, from models that would normally be too computationally heavy to work with directly. We have demonstrated the accuracy of the approximation through simple experiments with the Ising model, demonstrated some of its flexibility by applying our approximation to a real life data set as well as constructed a rejection sampling algorithm. We round off now with some possible future extensions as well as some

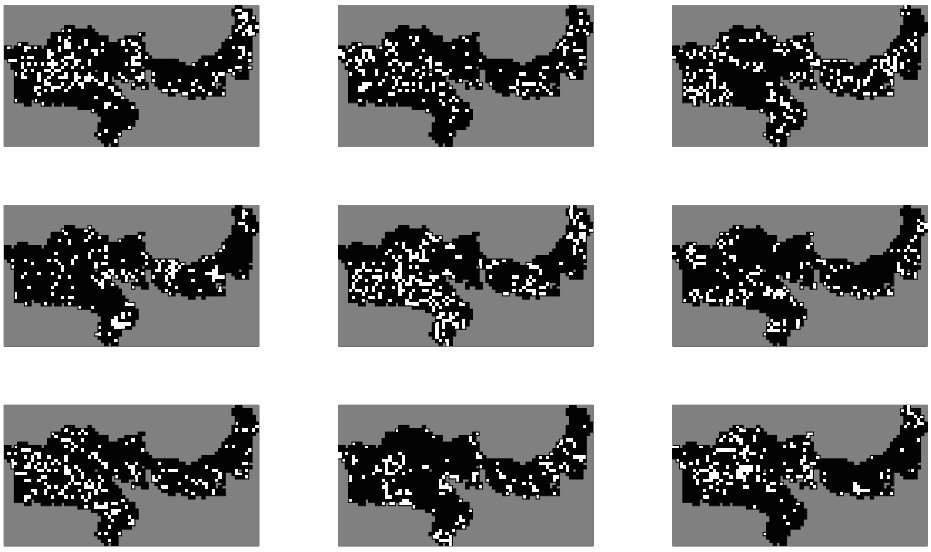

Figure 15: Nine realizations from $p(y|x, \mathcal{P})$ where x is simulated from $p(x|\theta^S, \theta^C, z)$ for values of θ^S, θ^C and $\mathcal{P}$ sampled from the RJMCMC run (discarding a burn-in of 50000 iterations).

closing remarks.

The approximation we have defined was inspired by the work in Tjelmeland and Austad (2012) and there are many parallels between the two. There the authors represented the energy function as a binary polynomial and dropped small interactions while running the forward-backward algorithm. This worked quite well, but had the drawback that for models with too strong interactions the approximation would either include too many terms and thus explode in run-time, or if the ϵ parameter was set low enough to run the algorithm, exclude so many of the interactions that the approximation became uninteresting. In a sense the work in this report has been an effort to deal with this issue. The construction of the algorithm allows a much more direct control over the run time. Also, by not just dropping small terms, but approximating the pseudo-Boolean function in such a way that we minimize the error sum of squares we manage to get better approximations of the models with stronger interactions. This also means that our approximation is even better suited for models with larger neighborhoods, although, there is definitely still room for improvement in these hard cases.

In our setting the sample space of our pseudo-Boolean function has a probability measure on it, however in our discussion of approximating pseudo-Boolean functions we have considered each state in the sample space as equally important. Assuming we are interested in approximating the normalizing constant, this is probably far from optimal and is reflected in our results. As we can see for the Ising model our approximation works better the smaller the interaction parameter θ. Our initial approximation attempts to spread the

error of removing interactions as evenly as possible among the states. Ideally, we would like to have small errors in states with a corresponding high probability, and larger errors in states with low probability. To get this approximation we could minimize a weighted error sum of squares function,

$$\text{WSSE}(f, \tilde{f}) = \sum_{x \in \Omega} \left[\left(f(x) - \tilde{f}(x) \right)^2 w(x) \right], \tag{72}$$

where for the weight function $w(x)$ we use the un-normalized probability distribution $\exp(f(x))$. Note that our current implementation is equivalent to letting $w(x)$ be a uniform distribution. This problem has also been studied in the literature, see for instance Ding et al. (2008, 2010). However, unlike the unweighted case an explicit solution is not readily available for a probability density like the MRF. The iterative method of removing interactions does not work here, nor can we group the equations like we do with the SOIR approximation. We can proceed as before and take partial derivatives with respect to $\tilde{\beta}^\lambda$ for all $\lambda \in \tilde{S}$ to get the system of equations,

$$\sum_{x \in \Omega_\lambda} w(x) \tilde{f}(x) = \sum_{x \in \Omega_\lambda} w(x) f(x) \ \forall \ \lambda \in \tilde{S}. \tag{73}$$

The computational cost of solving this linear system will be considerably higher than before, however this may be compensated for by a more accurate approximation. In particular in cases where the uniform assumption is very poor, we believe this trade off will be worth it. A second tactic available is to use an approximate distribution as weights instead of the full MRF. Ding et al. (2008) show how solutions exist for simple distributions and these might still capture where we want to minimize the error to get a good approximation. Depending on the distribution this might be a tactic worth pursuing as well.

We should note that the approximate forward-backward algorithm, defined in this report, applied to MRFs defines a probability distribution $\tilde{p}_\nu(x)$ and is an MRF in itself. In fact it is an example of a partially ordered Markov model (POMM), see Cressie and Davidson (1998). In this respect we can think of the approximation as a dynamic way of fitting POMMs to a general MRF. This is different from the treatment of POMMs in Cressie and Davidson (1998) where the partial ordering is specified by the user.

In our examples we have tested values for our algorithm parameter ν up to $\nu = 13$. As ν defines the size of $\check{\mathcal{N}}_i$ it should be perfectly plausible to run the algorithm for values up to $\nu = 20$, which should give some improved results over $\nu = 13$. We have neglected to demonstrate this here due to time constraints. It should also be mentioned that in our examples we have neglected to take advantage of a technique that can be used when summing out variables in a lexicographical order on a lattice. When the parameters β^λ are stationary we very quickly approach a stationary phase after summing out the first few columns. Thus we only really need to sum out the first few columns and the last, see Pettitt et al. (2003) for a demonstration of this technique. This could give substantial gains in run-time, particularly for the 100×100 lattice.

One of the nice properties of our approximation is its flexibility for handling many types of models. We could apply it to larger neighborhood structures or more special types of MRFs such as Bayesian networks.

In Section 2.3 we showed how we could go from removing just one interaction β^λ at the time in Section 2.2 to removing sets of interactions S_λ, simultaneously. This gave the advantage of faster computations, but more importantly a better understanding of how the error was distributed. That in itself was interesting, but it also allowed us to find better upper and lower bounds. The next step would be to consider removing all the interactions needed to reduce $|\tilde{\mathcal{N}}_i|$ to ν at once. This might lead to a better understanding of the approximation, which again might lead to better upper and lower bounds or ways of improving the approximation itself.

Acknowledgments

We acknowledge support from The Research Council of Norway, Statoil and ENI. We thank Nial Friel for providing us the data set used in Sections 6.3 and 6.4.

References

Augistin, N. H., Mugglestone, M. A. and Buckland, S. T. (1996). An autologistic model for the spatial distribution of wildlife, *Journal of Applied Ecology* **33**: 339–347.

Besag, J. E. (1974). Spatial interaction and the statistical analysis of lattice systems, *Journal of the Royal Statistical Society, Series B* **36**: 192–236.

Besag, J. E. (1986). On the statistical analysis of dirty pictures (with discussion), *Journal of the Royal Statistical Society, Series B* **48**: 259–302.

Buckland, S. T. and Elston, D. A. (1993). Empirical models for the spatial distribution of wildlife, *Journal of Applied Ecology* **30**: 478–495.

Clifford, P. (1990). Markov random fields in statistics, *in* G. R. Grimmett and D. J. A. Welsh (eds), *Disorder in Physical Systems*, Oxford University Press, pp. 19–31.

Cressie, N. A. C. (1993). *Statistics for Spatial Data*, 2 edn, John Wiley, New York.

Cressie, N. and Davidson, J. (1998). Image analysis with partially ordered Markov models, *Computational Statistics and Data Analysis* **29**: 1–26.

Ding, G., Lax, R., Chen, J. and Chen, P. P. (2008). Formulas for approximating pseudo-boolean random variables, *Discrete Applied Mathematics* **156**: 1581–1597.

Ding, G., Lax, R., Chen, J., Chen, P. P. and Marx, B. D. (2010). Transforms of pseudo-boolean random variables, *Discrete Applied Mathematics* **158**: 13–24.

Friel, N., Pettitt, A. N., Reeves, R. and Wit, E. (2009). Bayesian inference in hidden Markov random fields for binary data defined on large lattices, *Journal of Computational and Graphical Statistic* **18**: 243–261.

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

Gelman, A. and Meng, X. L. (1998). Simulating normalizing constants: from importance sampling to bridge sampling to path sampling, *Statistical Science* **13**: 163–185.

Geyer, C. J. and Thompson, E. A. (1992). Constrained Monte Carlo maximum likelihood for dependent data (with discussion), *Journal of the Royal Statistical Society, Series B* **54**: 657–699.

Grabisch, M., Marichal, J. L. and Roubens, M. (2000). Equivalent representations of set functions, *Mathematics of Operations Research* **25**: 157–178.

Green, P. J. (1995). Reversible jump MCMC computation and Bayesian model determination, *Biometrika* **82**: 711–732.

Gu, M. G. and Zhu, H. T. (2001). Maximum likelihood estimation for spatial models by Markov chain Monte Carlo stochastic approximation, *Journal of the Royal Statistical Society, Series B* **63**: 339–355.

Hammer, P. L. and Holzman, R. (1992). Approximations of pseudo-boolean functions; applications to game theory, *Methods and Models of Operations Research* **36**: 3–21.

Hammer, P. L. and Rudeanu, S. (1968). *Boolean methods in operations research and related areas*, Springer, Berlin.

Künsch, H. R. (2001). State space and hidden Markov models, *in* O. E. Barndorff-Nielsen, D. R. Cox and C. Klüppelberg (eds), *Complex Stochastic Systems*, Chapman & Hall/CRC.

Møller, J., Pettitt, A. N., Reeves, R. and Berthelsen, K. (2006). An efficient Markov chain Monte Carlo method for distributions with intractable normalizing constants, *Biometrika* **93**: 451–458.

Pettitt, A. N., Friel, N. and Reeves, R. (2003). Efficient calculation of the normalizing constant of the autologistic and related models on the cylinder and lattice, *Journal of the Royal Statistical Society, Series B* **65**: 235–247.

Propp, J. G. and Wilson, D. (1996). Exact sampling with coupled Markov chains and applications to statistical mechanics, *Random Structures and Algorithms* **9**: 223–252.

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

Ripley, B. D. (1987). *Stochastic simulation*, Wiley, New York.

Tjelmeland, H. and Austad, H. M. (2012). Exact and approximate recursive calculations for binary Markov random fields defined on graphs, *Journal of Computational and Graphical Statistic* . To appear.

A Optimal bounds for pseudo-Boolean functions

This section extends the work presented in Section 2.4. We show how a certain design of upper and lower bounds for pseudo-Boolean functions is optimal in the sense that it minimizes the error sum of squares. The term optimal must be used with some care however, as clearly our bounds are optimal with respect to the chosen representation set $\tilde{S}$. For this section we define $\tilde{S}$ by the following, it must be dense and is not allowed to contain any of the interactions in $S_{\{i,j\}}$, but otherwise can be chosen freely. So it may contain interactions not originally in S. As such it could be that $|\tilde{S}| > |S|$, making the bounds somewhat meaningless. None the less, we feel there is some insight to be gained from this discussion. Before we begin we present a theorem which we will need later in the discussion.

Theorem 6. Let $f(x)$ be a pseudo-Boolean function $f(x) = \sum_{\Lambda\in S}\beta^\Lambda\prod_{k\in\Lambda}x_k$ with the property that $\beta^\Lambda = 0$ for all $\Lambda \in S_{\{i,j\}}$. In other words there are no interactions involving both i and j. Then for any configuration of $x_{-\{i,j\}}$,

$$\begin{aligned} &f(x_i = 0, x_j = 0, x_{-\{i,j\}}) + f(x_1 = 1, x_2 = 1, x_{-\{i,j\}}) \\ &= f(x_1 = 1, x_2 = 0, x_{-\{i,j\}}) + f(x_1 = 0, x_2 = 1, x_{-\{i,j\}}). \end{aligned}$$

Proof. Since $\beta^\Lambda = 0$ for all $\Lambda \in S_{\{i,j\}}$ we can always rewrite $f(x)$ as follows,

$$f(x) = \sum_{\Lambda\in S_{\{i,j\}}}\left[(\beta^{\Lambda\setminus\{i,j\}} + \beta^{\Lambda\setminus\{j\}}x_i + \beta^{\Lambda\setminus\{i\}}x_j)\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right]. \tag{74}$$

Thus,

$$\begin{aligned} &f(x_i = 0, x_j = 0, x_{-\{i,j\}}) + f(x_1 = 1, x_2 = 1, x_{-\{i,j\}}) = \\ &= \sum_{\Lambda\in S_{\{i,j\}}}\left[(\beta^{\Lambda\setminus\{i,j\}})\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right] + \sum_{\Lambda\in S_{\{i,j\}}}\left[(\beta^{\Lambda\setminus\{i,j\}} + \beta^{\Lambda\setminus\{j\}} + \beta^{\Lambda\setminus\{i\}})\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right] \\ &= \sum_{\Lambda\in S_{\{i,j\}}}\left[(\beta^{\Lambda\setminus\{i,j\}} + (\beta^{\Lambda\setminus\{i,j\}} + \beta^{\Lambda\setminus\{j\}} + \beta^{\Lambda\setminus\{i\}})\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right] \\ &= \sum_{\Lambda\in S_{\{i,j\}}}\left[(\beta^{\Lambda\setminus\{i,j\}} + \beta^{\Lambda\setminus\{i\}})\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right] + \sum_{\Lambda\in S_{\{i,j\}}}\left[(\beta^{\Lambda\setminus\{i,j\}} + \beta^{\Lambda\setminus\{j\}})\prod_{k\in\Lambda\setminus\{i,j\}}x_k\right] \\ &= f(x_1 = 1, x_2 = 0, x_{-\{i,j\}}) + f(x_1 = 0, x_2 = 1, x_{-\{i,j\}}). \end{aligned}$$

This proves the theorem. □

Assume we have an approximation of $f(x)$, $A_{\tilde{S}}(f(x)) = \tilde{f}(x)$ with the requirement that $\tilde{S}$ should not contain any interactions involving both i and j. The goal is to construct optimal upper and lower bounds, i.e.

$$f_U(x) = \text{argmin}(\text{SSE}(f, f_U)), \text{ such that } f_U(x) \geqslant f(x) \ \forall \ x \in \Omega, \tag{75}$$

and

$$f_L(x) = \text{argmin}(\text{SSE}(f, f_L)), \text{ such that } f_L(x) \leqslant f(x) \ \forall \ x \in \Omega. \tag{76}$$

We define our upper and lower bounds as $f_U^*(x) = \tilde{f}(x) + g(x)$ and $f_L^*(x) = \tilde{f}(x) + h(x)$, where $g(x) = |f(x) - \tilde{f}(x)|$ and $h(x) = -|f(x) - \tilde{f}(x)|$. To prove that these bounds are optimal, we will show that for any pseudo-Boolean function $\dot{f}(x) = \sum_{\Lambda \in \tilde{S}} \dot{\beta}^\Lambda \prod_{k \in \Lambda} x_k$ such that $\dot{f}(x) \geqslant f(x) - f_U^*(x)$, $\text{SSE}(f, f_U^*) < \text{SSE}(f, f_U^* + \dot{f})$, except when $\dot{f}(x) = 0$, where the two are equal. Since $\text{SSE}(f, f_U^* + \dot{f}) = \sum_x (f(x) - f_U^*(x))^2 + \sum_x (\dot{f}(x))^2 + 2\sum_x \dot{f}(x)(f_U^*(x) - f(x))$, it is sufficient to show that $\sum_x \dot{f}(x)(f_U^*(x) - f(x)) \geqslant 0$. We start by studying $f_U^*(x) - f(x)$, inserting our expression for the error from (36) we get,

$$\begin{aligned} f_U^*(x) - f(x) &= |f(x) - \tilde{f}(x)| - (f(x) - \tilde{f}(x)) \\ &= \frac{1}{4}\left|\sum_{\Lambda \in S_{\{i,j\}}} \left[\beta^\Lambda \prod_{k \in \Lambda \setminus \{i,j\}} x_k\right]\right| - (x_i x_j + \frac{1}{4} - \frac{1}{2}x_j + \frac{1}{2}x_i) \sum_{\Lambda \in S_{\{i,j\}}} \left[\beta^\Lambda \prod_{k \in \Lambda \setminus \{i,j\}} x_k\right]. \end{aligned}$$

To study this expression more closely we first consider an $x_{-\{i,j\}}$ where $\sum_{\Lambda \in S_{\{i,j\}}} \left[\beta^\Lambda \prod_{k \in \Lambda \setminus \{i,j\}} x_k\right] > 0$. Then,

$$f_U^*(x) - f(x) = \begin{cases} 0 & x_i = 0, x_j = 0 \vee x_i = 1, x_j = 1, \\ 2|f(x) - \tilde{f}(x)| & x_i = 1, x_j = 0 \vee x_i = 0, x_j = 1. \end{cases} \tag{77}$$

Equivalently, for an $x_{-\{i,j\}}$, for which $\sum_{\Lambda \in S_{\{i,j\}}} \left[\beta^\Lambda \prod_{k \in \Lambda \setminus \{i,j\}} x_k\right] < 0$ we get,

$$f_U^*(x) - f(x) = \begin{cases} 0 & x_i = 1, x_j = 0 \vee x_i = 0, x_j = 1, \\ 2|f(x) - \tilde{f}(x)| & x_i = 0, x_j = 0 \vee x_i = 1, x_j = 1\}. \end{cases} \tag{78}$$

Note that this means that for half of the configurations of $x \in \Omega$, $f_U^*(x) - f(x)$ is zero. We define,

$$\Omega_0 = \{x : f_U^*(x) - f(x) = 0\}. \tag{79}$$

$$\Omega_1 = \{x : f_U^*(x) - f(x) = 2|f(x) - \tilde{f}(x)|\}. \tag{80}$$

Note that $\Omega_0 \cup \Omega_1 = \Omega$ and $|\Omega_0| = |\Omega_1|$. Using (79) and (80) we get,

$$\sum_{x \in \Omega} \dot{f}(x)(f_U^*(x) - f(x)) = 2 \sum_{x \in \Omega_1} \dot{f}(x)|f(x) - \tilde{f}(x)|, \tag{81}$$

since $f_U^*(x) - f(x) = 0$ for all $x \in \Omega_0$. The function $\dot{f}(x)|f(x) - \tilde{f}(x)|$ clearly has no interactions involving both i and j since $\dot{f}(x)$ is defined over the set $\tilde{S}$, which is designed not to include any interactions involving both i and j, and $|f(x) - \tilde{f}(x)|$, see (24), is independent of x_i and x_j and includes no interactions with either i or j. Thereby Theorem 6 applies to $\dot{f}(x)|f(x)-\tilde{f}(x)|$. All terms in the sum $\sum_{x\in\Omega_1} \dot{f}(x)|f(x)-\tilde{f}(x)|$ can be grouped into groups of two terms that are either

$$\begin{aligned}&\dot{f}(x_i = 0, x_j = 0, x_{-\{i,j\}})|f(x_i = 0, x_j = 0, x_{-\{i,j\}}) - \tilde{f}(x_i = 0, x_j = 0, x_{-\{i,j\}})| \\ &+\dot{f}(x_i = 1, x_j = 1, x_{-\{i,j\}})|f(x_i = 1, x_j = 1, x_{-\{i,j\}}) - \tilde{f}(x_i = 1, x_j = 1, x_{-\{i,j\}})|, \end{aligned} \tag{82}$$

or,

$$\begin{aligned}&\dot{f}(x_i = 1, x_j = 0, x_{-\{i,j\}})|f(x_i = 1, x_j = 0, x_{-\{i,j\}}) - \tilde{f}(x_i = 1, x_j = 0, x_{-\{i,j\}})| \\ &+\dot{f}(x_i = 0, x_j = 1, x_{-\{i,j\}})|f(x_i = 0, x_j = 1, x_{-\{i,j\}}) - \tilde{f}(x_i = 0, x_j = 1, x_{-\{i,j\}})|, \end{aligned} \tag{83}$$

depending on $x_{-\{i,j\}}$. Theorem 6 means that the sum in (82) is equal to the sum in (83). Thus since $\Omega_1 = \Omega_0^c$,

$$\sum_{x\in\Omega_1} \dot{f}(x)|f(x) - \tilde{f}(x)| = \sum_{x\in\Omega_0} \dot{f}(x)|f(x) - \tilde{f}(x)|. \tag{84}$$

Combining (81) and (84) we have that,

$$\sum_{x\in\Omega} \dot{f}(x)(f_U^*(x) - f(x)) = 2\sum_{x\in\Omega_1} \dot{f}(x)|f(x) - \tilde{f}(x)| = 2\sum_{x\in\Omega_0} \dot{f}(x)|f(x) - \tilde{f}(x)| \geqslant 0,$$

since $\dot{f}(x) \geqslant 0$ for all $x \in \Omega_0$. Therefore $f_U^*(x) = f_U(x)$, and through a similar argument $f_L^*(x) = f_L(x)$.