Haakon Michael Austad

Approximations of Binary Markov Random Elds

Thesis for the degree of Philosophiae Doctor

Trondheim, November 2011

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Mathematical Sciences

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Mathematical Sciences

ISBN 978-82-471-3156-5 (printed ver.)
ISBN 978-82-471-3157-2 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2011:292

Printed by NTNU-trykk

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The work has mainly been carried out at the Department of Mathematical Sciences at NTNU. The research has been funded via the project "Multipoint Methods for Improved Reservoir Models" at the Norwegian Computing Center.

First of all I would like to thank my supervisor, Professor Håkon Tjelmeland, for all his support and guidance throughout my work on this thesis. I have greatly enjoyed our many discussions and wish to thank him for sharing his insight and advice during these last four years.

During the fall of 2009 I was fortunate enough to get the opportunity to visit and collaborate with professor Nial Friel at the university college Dublin. I am indebted to professor Friel for his continuous help and advice both during and after my stay in 2009, as well as during my shorter return visit in September 2010. I would also like to thank everyone in the post-grad room at UCD for taking such good care of me and making me feel like a part of the group.

Thank you to all my colleagues at the Department of Mathematical Sciences, in particular members of the statistics group. A special thanks to Anne Kajander of the administrative staff and Per Kristian Hove in the computer support group for always being so helpful.

Space does not permit me to list all the friends, both at NTNU and elsewhere, that I would like to thank here, so I will have to make this out to all of you. Thank you for being so supportive and kind, and simply being the great friends that you are. A special thank you has to go to my girlfriend for always being there for me and putting up with me. Somehow you always know the right things to say and do to chase away the clouds. Thank you.

Finally, I would like to thank my parents, my brother and the rest of my family for all your love and support.

Trondheim, July 2011

Haakon Michael Austad

Thesis outline

Background

Consider the following setting. Assume we have some continuous region, either one, two or three dimensional, which we partition into a lattice of pixels. Each pixel may take one of a finite discrete set of values. These may represent colors or gray levels in an image, lithology fluid classes in sub-sea geology, crop types in a satellite image or any number of other examples. Common for them all is the notion that there is some true, unknown configuration of the pixel values, which we want to reconstruct. To do this, we have two sources of information. The first is our data, some noisy observation of the field. For all or possibly a subset of the pixels we have observed values which we usually assume follow some statistical distribution. The second source of information, we can think of as "past experience", an understanding of what this kind of region usually looks like. This is refered to as our prior knowledge of the system. Quantifying this information usually centers around the notion that pixels close to each other will tend to take the same value. A pixels nearest neighbors have the most influence over what value that pixel takes. Statistical models that quantify this second source of information is the chosen focus of study for this PhD.

The last decades explosive growth of computational power has led to a rapid expansion of the set of problems which are computationally manageable. This, coupled with an increase in the production of spatial data-sets, means spatial statistics and image processing is more relevant today than ever before. Spatial data can arise from a multitude of sources. Satellite images and geological data have already been mentioned, other sources could include medical diagnostics, X-ray or microscope images. The list of tasks is equally diverse, recent technological advances has stirred an interest in machine vision, features like object and facial detection or object tracking. Properly defined discrete statistical models for describing prior knowledge is of great use for many of these fields. Working with these models is unfortunately often computationally very hard. The reason for this is perhaps best illustrated through a small example. Consider a small two dimensional image consisting of 100×100 pixels. If each pixel takes one of two values, 0 or 1, we have $2^{100\times 100}$ possible configurations of our image. Evaluating all these configurations in reasonable time is well out of reach of even the most powerful supercomputer. In our statistical models this manifests itself in the form of an unknown normalizing constant. This means basic tasks like evaluating the probability of a configuration of pixels or simulating realizations from our model become non-trivial tasks.

The typical setting we will be studying in most of our applications is the following. Imagine we have some spatial field x for which we have some parametric model $p(x|\theta)$, where θ are our model parameters. Approaching the problem in a Bayesian manner we assign some prior to our parameters, $p(\theta)$, and do inference around the posterior distribution of the parameters, $p(\theta|x)$. Using Bayes law we can write,

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} \propto p(x|\theta)p(\theta). \tag{1}$$

It might also be the case that x is a latent unobserved field for which we have observations y. Usually we will make a conditional independence assumption on the likelihood of y, i.e. the distribution of a single pixel value y_i conditional on x is independent of the rest of the y_j's. Again we can use Bayes law to write the posterior distribution $p(\theta|y)$ as,

$$p(\theta|y) = \frac{p(x,\theta|y)}{p(x|\theta,y)} \propto \frac{p(y|x)p(x|\theta)p(\theta)}{p(x|\theta,y)}. \tag{2}$$

The problem is that $p(x|\theta)$ and $p(x|\theta, y)$ are typically unavailable to us due to unknown normalizing constants.

Solving this problem of the normalizing constants is of course a well studied problem and many different approaches exist. A number of simulation techniques have been developed, revolving around use of Markov chain Monte Carlo (MCMC) to estimate the unknown normalizing constant. See for instance Geyer and Thompson (1992), the path sampling algorithm in Gelman and Meng (1998) or Møller et al. (2006) where the authors construct an auxiliary variable Metropolis-Hastings algorithm using perfect sampling. The point of origin for this PhD is the deterministic approximations presented in Reeves and Pettitt (2004) and Friel and Rue (2007). Deterministic in this setting means that repeating the approximation algorithm, using the same algorithm parameters, yields the same approximation. This differs from most simulation based methods.

In the remainder of this introduction we briefly discuss the central elements of the three parts of this thesis. We first define Markov random fields and exponential random graph models. The forward-backward algorithm is then described and pseudo-Boolean functions are defined. We finish off with a summary of the three parts.

Markov random fields

For a general introduction to Markov random fields (MRF) we refer the reader to Besag (1974), Kindermann and Snell (1980) or Cressie (1993). In this section we will define neighborhoods and cliques and explain the importance of the Hammersley-Clifford theorem as well as define the Ising model.

The class of models known as Markov random fields encapsulates a wide range of different models. Let N denote a list of pixels or nodes, $N = \{1, 2, \ldots, n\}$. To each node i we associate a discrete stochastic variable $x_i \in \mathcal{X}$. We let $x = (x_1, \ldots, x_n) \in \mathcal{X}^n$ and use the notation $x_\Lambda = \{x_i : i \in \Lambda\}$, for some subset $\Lambda \subseteq N$, and $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. Constructing a probability distribution for this quite general situation can be difficult. An MRF is essentially a way of designing this distribution by using the concept of neighborhoods. A formal definition of this concept is required. We associate a neighborhood system $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$ to our set of nodes where $\mathcal{N}_i$ denotes the list of neighbors of node i. We require a symmetrical system, so if j is a neighbor of i we must also have that i is a neighbor of j. Thus if $j \in \mathcal{N}_i$, then $i \in \mathcal{N}_j$. Also, by convention a node is not a neighbor of itself, so $i \notin \mathcal{N}_i$. We are now ready to give a formal definition of an MRF. We say that x is an MRF with respect to a neighborhood system $\mathcal{N}$ if the conditional distribution of a node given the rest is equal to the distribution of a node given its neighbors,

$$p(x_i|x_{-i}) = p(x_i|x_{\mathcal{N}_i}). \tag{3}$$

This is the so called Markov property of the distribution. It is useful to approach the modeling issue in this manner as it is often easier to define how a single node is influenced by its neighbors, rather than considering the entire field at once. Before we can take advantage of this however we need to show that defining the distribution through its full conditionals yields a valid distribution. To introduce the theorem that proves this, we first need to define a clique. A set $\Lambda \subseteq N$ is said to be a clique if $i \in \mathcal{N}_j$ for all pairs i and j in Λ. In other words, a clique is a set of nodes where all the nodes in the set are neighbors of all the other nodes in the set. We say that a clique is a maximal clique if it is not a subset of another clique and let $\mathcal{C}$ denote the set of all the maximal cliques over N. We can now state the Hammersley-Clifford theorem, see Besag (1974) and Clifford (1990),

Theorem 1. A distribution $p(x|\theta)$, satisfying $p(x|\theta) > 0$ for all $x \in \mathcal{X}^n$ is an MRF if and only if we can write the distribution as,

$$p(x|\theta) = \frac{1}{c(\theta)} \exp\left(\sum_{\Lambda \in \mathcal{C}} U_\Lambda(x_\Lambda, \theta)\right), \tag{4}$$

where $U_\Lambda(x_\Lambda, \theta)$ are some functions, often termed clique potential functions, $c(\theta)$ is a normalizing constant and θ are our model parameters.

This theorem is important as it tells us that a model specified by its full conditionals is a legitimate probability distribution assuming we can find a distribution of the form (4) that matches the full conditionals. We can specify our model through the full conditionals or through the clique potential functions and there is a one to one correspondence between the two. The difficulty when working with MRFs is that the normalizing constant $c(\theta)$ is generally unknown and calculation of $c(\theta)$ involves a sum over $|\mathcal{X}|^n$ terms. Performing this calculation directly is obviously impossible even for reasonably small values of $\mathcal{X}$ and n and certainly for most applications of practical interest. Since $c(\theta)$ is a function of θ it will also be necessary to re-calculate the normalizing constant every time we change the model parameters.

Perhaps the most common example when discussing MRFs is the Ising model, named after Ernst Ising and presented for the first time in 1925. Let each x_i represent a node on a 2 dimensional lattice and let $\mathcal{X} = \{0, 1\}$, so each x_i is a binary variable. Let $\mathcal{N}$ be a first order neighborhood system, i.e the neighbors of each node x_i are the four nodes immediately adjacent to it. $\mathcal{C}$ then becomes the set of all pairs of nodes adjacent to each other. Denoting the neighboring pairs as $i \sim j$, we can write the distribution of Ising model as,

$$p(x|\theta) = \frac{1}{c(\theta)} \exp\left(\theta \sum_{i \sim j} I(x_i = x_j)\right), \tag{5}$$

where $I(x_i = x_j)$ is the indicator function equal to 1 if $x_i = x_j$ and 0 otherwise. Thus, if all neighbors of a node are 1, that node will tend to take the value 1 as well. How influential the neighbors are is determined by our parameter θ. A value for θ close to zero will give more noisy realizations while a high value for θ will give realizations with large areas dominated by one value. An interesting feature of the Ising model is the existence of a phase transition phenomenon. Assume the Ising model is defined on a lattice where the values at the boundary are given. If the lattice were large enough one would intuitively expect a node at the center of the lattice to be unaffected by the values at the boundary. More precisely, if the lattice were infinitely large, the marginal distribution of a node at the center should be unaffected by the values at the boundary. As it turns out, this is true only for values of θ bellow a so called "critical value", θ_{critical}. It can be shown that for the Ising model defined as in (5), $\theta_{\text{critical}} = \log(1 + \sqrt{2}) \approx 0.88$. Although simple, the Ising model is frequently used in a number of fields.

Exponential random graph models

Exponential random graph models (ERGM), see Wasserman and Pattison (1996) and Robins et al. (2007), often refered to as the p^* model, is a model for describing the structure of networks and is much used in social network analysis. Consider the following setting. We have an undirected graph consisting of m nodes. Each of the edges between the nodes are either on or off, these make up our binary variables. We can represent a graph through an $m \times m$ adjacency matrix y. Each entry of the matrix y_{ij} represents an edge and takes value $y_{ij} = 1$ if the edge is on and $y_{ij} = 0$ if it is not. By definition a node is never connected by an edge to itself, so the diagonal of the adjacency matrix is 0, $y_{ii} = 0$ for all $i \in \{1, \dots, m\}$. Note that y will be symmetric and that we thereby have $n = m(m-1)/2$ binary variables. The ERGM writes the distribution of y as,

$$p(y|\theta) = \frac{1}{c(\theta)} \exp\left(\sum_{i=1}^{k} \theta_i s_i(y)\right), \tag{6}$$

where $\theta = (\theta_1, \dots, \theta_k)$ are our model parameters, $c(\theta)$ is the normalizing constant and $s_i(y)$ is some sufficient statistic. The simplest example of such a sufficient statistic is probably the number of active edges, $s(y) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} y_{i,j}$, but more complicated statistics are also frequently used. Two examples are the number of two-star configurations, $s(y) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} \sum_{k=j+1}^{m} y_{i,k} y_{j,k}$, or the number of triangle configurations, $s(y) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} \sum_{k=j+1}^{m} y_{i,k} y_{j,k} y_{i,j}$. As before the computational problem arises due to the unknown normalizing constant, whose calculation requires summing over $2^{m(m-1)/2}$ terms. It should be noted that ERGMs are a subclass of MRFs, where two edges are neighbors if they share a common node.

The forward-backward algorithm

At its core, the forward-backward algorithm is simply an efficient way of performing summation over a large set of variables. In Reeves and Pettitt (2004) the authors presented the algorithm for general factorisable models and in Friel and Rue (2007) the algorithm was further explored and a number of properties and examples illustrated. In this section we give a brief summary and attempt to convey the central aspects of the algorithm.

Let $x = (x_1, \dots, x_n)$, $x_i \in \mathcal{X}$, be a vector of discrete variables, $N = \{1, \dots, n\}$ the set of indices and $q(x)$ the un-normalized distribution of x. We will use the notation $x_\Lambda = (x_i : i \in \Lambda)$. For a distribution $p(x) = \frac{1}{c} q(x)$, our interest lies in calculating the sum,

$$c = \sum_{x \in \mathcal{X}^n} q(x). \tag{7}$$

If this were to be done in a straightforward manner, we would have to evaluate $|\mathcal{X}|^n$ terms to calculate the sum, which can be very time consuming. Assume we can factorize $q(x)$ in the following manner,

$$q(x) = \prod_{i=1}^{n} q_i(x_{\mathcal{L}_i}), \tag{8}$$

where the sets $\mathcal{L}_i$ are defined as $\mathcal{L}_i \subseteq \{i, \dots, n\}$ such that $i \in \mathcal{L}_i$ and $\mathcal{L}_i \backslash \{i\} \subseteq \mathcal{L}_{i+1}$. Defining $r = \max_{i \in N} |\mathcal{L}_i|$, we then say that $q(x)$ is a lag-r general factorisable model. The fact that we can factorize $q(x)$ in this manner means we can rewrite the sum in (7) as,

$$\sum_{x \in \mathcal{X}^n} q(x) = \sum_{x_n} \left[q_n(x_{\mathcal{L}_n}) \cdots \sum_{x_i} \left[q_i(x_{\mathcal{L}_i}) \cdots \sum_{x_1} q_1(x_{\mathcal{L}_1}) \right] \right]. \tag{9}$$

The forward-backward algorithm takes advantage of this factorization by constructing the following recursive algorithm,

$$\begin{aligned} c_1(x_{\mathcal{L}_1 \backslash \{1\}}) &= \sum_{x_1} q_1(x_{\mathcal{L}_1}), \\ &\vdots \\ c_i(x_{\mathcal{L}_i \backslash \{i\}}) &= \sum_{x_i} \left[q_i(x_{\mathcal{L}_i}) c_{i-1}(x_{\mathcal{L}_{i-1} \backslash \{i-1\}}) \right], \\ &\vdots \\ c_n &= \sum_{x_n} \left[q_n(x_{\mathcal{L}_n}) c_{n-1}(x_{\mathcal{L}_{n-1} \backslash \{n-1\}}) \right], \end{aligned} \tag{10}$$

where obviously $c_n = c$. Calculating $c_1, \dots, c_n$ and thereby the sum in (7) thus requires $\sum_{i=1}^n |\mathcal{X}|^{|\mathcal{L}_i|-1}$ calculations. This calculation also gives us access to the conditional probabilities $p(x_i|x_{i+1:n})$ since,

$$p(x_i|x_{i+1:n}) = \frac{q_i(x_{\mathcal{L}_i}) c_{i-1}(x_{\mathcal{L}_{i-1} \backslash \{i-1\}})}{c_i(x_{\mathcal{L}_i \backslash \{i\}})}. \tag{11}$$

Through this we can calculate the full likelihood,

$$p(x) = p(x_n) \prod_{i=1}^{n-1} p(x_i|x_{i+1:n}). \tag{12}$$

This can also be used to generate samples from the distribution.

The efficiency of the algorithm obviously depends on the value of r. Consider the Ising model on a $u \times v$ lattice and assume a lexicographical ordering of the nodes and $u \leqslant v$. In this case we could choose, $\mathcal{L}_1 = \{1, 2, u+1\}$, $\mathcal{L}_2 = \{2, 3, u+1, u+2\}$, $\mathcal{L}_3 = \{3, 4, u+1, u+2, u+3\}$ and so on. The size of $\mathcal{L}_i$ grows up to $u+1$ and thus, $r = u+1$. In practice this restricts the algorithm to lattices where $u \leqslant 20$ for the Ising model. If the MRF has a neighborhood system with larger neighborhoods this will further restrict the size of the lattice we can handle.

The ERGM model is an example of a class of MRFs where the forward-backward algorithm is poorly suited. Assume we have a graph with m nodes and $n = m(m-1)/2$ edges represented by $y = (y_1, \ldots, y_n)$. Since two edges are neighbors if they share a common node we could choose $\mathcal{L}_1$ to contain 1 and the indices of all edges that share a node with y_1, thus $|\mathcal{L}_1| = 1 + 2(m-2)$. If the ordering is chosen such that $2 \in \mathcal{L}_1$, we then get $|\mathcal{L}_2| = 2(m-2) + (m-3)$. How this continues depends heavily on the ordering of the edges. There is no way to factorize the model, the way we can with for instance the Ising model on a lattice, that gives any substantial saving in calculating c. This means ERGMs have to be approached differently from MRFs defined on lattices.

Pseudo-Boolean functions

A pseudo-Boolean function is a function $f : \{0,1\}^n \to \mathbb{R}$. So given a vector of n binary variables $x = (x_1, \ldots, x_n)$, a pseudo-Boolean function associates a real value to each configuration of x. An example of such a function could be statistical data when the explanatory variables are binary. We could express a pseudo-Boolean function as a list of 2^n elements, using some ordering. Letting $N = \{1, \ldots, n\}$ be the list of indices, Hammer and Rudeanu (1968) showed that any pseudo-Boolean function can be expressed uniquely as a binary polynomial of n variables,

$$f(x) = \sum_{\Lambda \subseteq N} \beta^{\Lambda} \prod_{k \in \Lambda} x_k, \tag{13}$$

where β^{Λ} are real valued coefficients which we refer to as interactions. Note that in general, representing a pseudo-Boolean function in this manner still requires 2^n terms. Given a pseudo-Boolean function $f(x)$ we can recursively calculate the interactions β^{Λ} by evaluating $f(x)$ for different configurations of x. Letting $x^{\Lambda} = (x_1, \ldots, x_n)$, where $x_k = 1$ if $k \in \Lambda$ and $x_k = 0$ if $k \notin \Lambda$, we first calculate the zero order interaction $\beta^{\varnothing}$,

$$\beta^{\varnothing} = f(x^{\varnothing}). \tag{14}$$

Next we can calculate all the first order interactions,

$$\beta^{\{i\}} = f(x^{\{i\}}) - \beta^{\varnothing} \;\forall\; i \in N. \tag{15}$$

Continuing in this recursive way we get the general expression for calculating the interactions,

$$\beta^{\Lambda} = f(x^{\Lambda}) - \sum_{\lambda \subset \Lambda} \beta^{\lambda} \;\forall\; \Lambda \subseteq N. \tag{16}$$

Summary

In this section we give a brief summary of the thesis. The three parts of this thesis can be read in any order, although we recommend reading part I before part III. The two are closely related and both deal primarily with binary Markov random fields defined on lattices. Part II focuses on the ERGM class of models and applies different techniques than those in parts I and III.

All the applications and examples presented in this thesis are binary and this has also been the focus when developing the algorithms and techniques we use. It is possible to extend and modify the algorithms to handle more general discrete distributions, this is briefly discussed in part I, and some small experiments have been run to test this. In general the computational load grows considerably when one increases the number of colors in the model and how to extend the algorithms to handle this is not a trivial task. This could be an interesting area for further study however.

Part I proposes an approximation to binary Markov random fields. The basis for the approximation is the forward-backward algorithm. As we have pointed out, whether this algorithm is computationally viable depends on the lag, r, of the model it is applied to. For MRFs on lattices this restricts the algorithm to quite small lattices, typically consisting of $\leqslant 20$ rows. The idea behind our approximation is the following. The sums defining c_1 to c_n in (10) and the clique potential functions of the MRF are pseudo-Boolean functions and as such can be represented as in (13). Assuming we do this, we can then drop terms in the sum in (13), that are smaller than some parameter ϵ, during the summation procedure in the forward-backward algorithm. If this is done appropriately, it reduces the amount of computation and storage required for the algorithm and allows the forward-backward algorithm to be applied to MRFs on larger grids. This in turn defines an approximate MRF. To test the algorithm a number of different numerical experiments were run. This included tests with the Ising model as well as MRFs with larger neighborhoods. The paper in Part I represents our first proper attempt at constructing an approximate MRF, following some work with multigrid approaches, as well as our first meeting with the representation in (13). At the time we were not familiar with the term "pseudo-Boolean function" and as such the reader will not find this term mentioned anywhere in the paper. The representation in (13) is refered to as a sum of interaction parameters, throughout part I. Although the approximation constructed in part III is significantly different from the approximation in part I, the lessons learned from this first part inspired much of the work in part III.

Most of the work in the paper in part II was done while visiting and collaborating with professor Nial Friel at the University College of Dublin. This part focuses on the ERGM class of models where, as mentioned, the forward-backward algorithm is not well suited. The approximation constructed in part I, while applicable, performs poorly for this class of models and as such other approaches were required. The reduced dependency approximation (RDA) in Friel et al. (2009) uses the forward-backward algorithm to calculate the normalizing constant for small lattices and combines these to get an approximation of the normalizing constant for MRFs defined on large lattices. This has been shown to work well for MRFs defined on lattices. However, while choosing which sublattices to calculate for the RDA is not an issue for MRFs defined on lattices it becomes a difficult question when working with ERGMs. The paper in part II focuses on modifying and adapting the RDA approximation for use on ERGMs. The method was tested on a number of examples using Bayesian analysis and Bayesian model choice and compared to pseudo likelihood and block-pseudo likelihood methods.

Part III represents an attempt at improving the approximation defined in part I. Improving on the initial approximation not only involved finding an approximation that gave a higher degree of accuracy, but also finding a more formal criteria for how the approximation was designed, rather than simply dropping small terms in the sum in (13). To quantify the error of the approximation it was also of interest to construct upper and lower bounds for the normalizing constant. Building on the work in part I we initially attempted to construct these bounds using our approximate MRFs as a starting point. Although our initial attempts at this resulted in bounds that were too wide to be of any real interest it did lead to a closer investigation of the literature available on pseudo-Boolean functions, which in turn led to the ideas behind the approximation presented in part III. In this part we once again, as in part I, focus on approximating binary MRFs through an approximate forward-backward algorithm. The approximation in part I dropped terms in the sum in (13) smaller than a given parameter ϵ, and as such the algorithm, in a sense, dynamically adapted to the model it was applied to. This meant that for a small enough value of ϵ we could be reasonably confident that the algorithm would give us a good approximation for a large number of different models. Unfortunately, this also meant that if the correlations between the variables in the model were too strong, for a small ϵ, the algorithm would be unable to remove enough interactions to make the algorithm computationally viable. One could increase ϵ, but this often resulted in so many large interactions being dropped that the approximation became poor. The work in part III was partly motivated by a desire to construct an algorithm where the user maintained a more direct control over the computational load, while also improving the accuracy of our first attempt. The approximation in part III does this by choosing which interactions to include in the model directly and instead of simply dropping terms, approaches the problem as a general function approximation problem where one seeks to minimize the error sum of squares given a chosen representation. We devised a fast technique for solving the resulting linear system of equations which arose from this minimization problem. With this new approximation we were also able to define reasonably tight upper and lower bounds for the normalizing constant. As in the paper in part I the report in part III tests the new approximation on a number of numerical examples using different binary MRFs defined on lattices. The results indicate a significant improvement over the approximation in part I, in particular for the difficult cases where the interactions in the field are strong.

References

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

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.

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, *Statstical 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.

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

Kindermann, R. and Snell, J. L. (1980). *Markov random fields and their applications*, American Mathematical Society, Providence, R.I.

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

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

Robins, G., Snijders, T., Wang, P., Handcock, M. and Pattison, P. (2007). Recent developments in exponential random graph (p^*) models for social networks, *Social networks* **29**: 192–215.

Wasserman, S. and Pattison, P. (1996). Logit models and logistic regressions for social networks: I. an introduction to markov graphs and p^*, *psychometrika* **61**: 401–425.

Part I:

Exact and approximate recursive calculations for binary Markov random fields defined on graphs.

Håkon Tjelmeland and Haakon Michael Austad.

Paper accepted for publication in *Journal of Computational and Graphical Statistics.*

Exact and approximate recursive calculations for binary Markov random fields defined on graphs

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

Abstract

In this paper we propose computationally feasible approximations to binary Markov random fields. The basis of the approximation is the forward-backward algorithm. The exact forward-backward algorithm is computationally feasible only for fields defined on small lattices. The forward part of the algorithm computes a series of joint marginal distributions by summing out each variable in turn. We represent these joint marginal distributions by interaction parameters of different orders. The approximation is defined by approximating to zero all interaction parameters that are sufficiently close to zero. In addition, an interaction parameter is approximated to zero whenever all associated lower level interactions are (approximated to) zero. If sufficiently many interaction parameters are set to zero, the algorithm is computationally feasible both in terms of computation time and memory requirements. The resulting approximate forward part of the forward-backward algorithm defines an approximation to the intractable normalizing constant and the corresponding backward part of the algorithm defines a computationally feasible approximation to the Markov random field. We present numerical examples demonstrating the quality of the approximation.

The Supplemental Material for this article, which is available online, includes R and C code for the proposed recursive algorithms.

Key words: approximate inference, autologistic model, forward-backward algorithm, Ising model, Markov random field, recursive computation.

1 Introduction

In this paper we consider computational problems related to inference in binary Markov random fields (MRF). An example is for an observed binary image x to find the maximum likelihood estimator $\widehat{\theta}$ for a parameter vector θ in a specified parametric family $p(x|\theta)$ of binary MRFs. Alternatively, x is a latent unobserved variable and a y is observed with assumed distribution $p(y|x,\theta)$. Again the maximum likelihood estimator for θ and potentially also for x is of interest. The latter problem can also be formulated in a Bayesian setting. For this let $p(y|x,\theta)$ be as above, let $p(x|\theta)$ be an MRF prior for x and assume a prior $p(\theta)$ for θ. The interest is typically in the posterior distribution for θ or x. The problem in these situations is that the MRF $p(x|\theta)$ includes a computationally intractable normalizing constant. This makes both numerical optimization and standard Markov chain Monte Carlo (MCMC) algorithms infeasible. See Møller et al. (2006) and Murray (2007) for similar problems for other model classes.

Strategies for dealing with intractable normalizing constants proposed in the literature can be categorized into three groups. The first is to replace the intractable constant with an approximation. For MRFs Besag (1974) defines a pseudo-likelihood function as a product of full conditionals and estimates θ when x is observed by maximizing this function. In Rydén and Titterington (1998) the same pseudo-likelihood function is used as an approximation to the exact likelihood in a Bayesian setting. Heikkinen and Högmander (1994) and Huang and Ogata (2002) define alternative pseudo-likelihood functions. Friel and Rue (2007) and Friel et al. (2009) define approximations based on exact calculations for smaller lattices by the so called forward-backward algorithm (Künsch, 2001; Scott, 2002; Pettitt et al., 2003). Results for smaller lattices are glued together to give approximative results for larger lattices. However, this is only feasible for MRFs with very small neighborhoods, as otherwise exact computations are infeasible even for small lattices. The second approach is to estimate the intractable normalizing constant using MCMC samples. Geyer and Thompson (1995) run an MCMC chain for a specific value of θ, θ_0, use this to estimate the normalizing constant for θ close to θ_0, and perform numerical optimization on the resulting estimated likelihood function. Tjelmeland and Besag (1998) use this strategy to find the maximum likelihood estimator for a binary MRF. Gelman and Meng (1998) run independent MCMC chains for several θ values, use this to estimate the normalizing constant as a function of θ, and run MCMC for $p(x,\theta|y)$ with the normalizing constant replaced by this estimate. Møller et al. (2006) use a third strategy to cope with an intractable normalizing constant in a Bayesian problem. An auxiliary variable Metropolis–Hastings algorithm is defined where the intractable normalizing constant cancels from the Metropolis–Hastings ratio. However, the algorithm requires exact sampling from $p(x|\theta)$ and to obtain good mixing an approximation to $p(x|\theta)$ without an intractable normalizing constant must be available, see also Murray (2007).

In the present article we define a deterministic approximation to $p(x|\theta)$ where the normalizing constant is easily computable. Our solution is thereby within the first class discussed above, but it can also be applied in the construction of Møller et al. (2006). As in Friel and Rue (2007) and Friel et al. (2009), our starting point is the exact forward-backward algorithm. The forward part of this algorithm computes a series of joint marginal distributions by summing out each variable in turn. We represent the joint marginal distributions by interaction parameters of different orders. We approximate to zero interaction parameters that are close to zero. In addition, an interaction parameter is approximated to zero if all associated lower level interactions are (approximated to) zero. If sufficiently many interactions are set to zero, the algorithm is feasible both in terms of computation time and memory requirements. The approach does not require exact computations on smaller lattices, so it can be used also for MRFs with somewhat larger neighborhoods.

The paper is organized as follows. In Section 2 we discuss how functions of binary variables can be represented by interaction parameters, and define a canonical representation of functions with a Markov property. In Section 3 we relate this to binary MRFs. The exact forward-backward algorithm for MRFs is specified in Section 4.1, and in Section 4.2 we introduce our approximation scheme. In Section 5 we briefly discuss how our strategy can be generalized to fields with more than two classes. In Section 6 we present results for two example MRFs, and in Section 7 we discuss a real data Bayesian example. Finally, Section 8 provides conclusions.

2 Canonical representation

In this section we first define how a function of binary variables can be represented by interaction parameters. Next, we limit the attention to functions with a Markov property and introduce a canonical representation for such functions.

2.1 Interaction parameters

Let S be a finite set of $n = |S|$ elements. To each $k \in S$ we associate a binary variable $x_k \in \{0, 1\}$ and let $x = (x_k, k \in S) \in \Omega = \{0, 1\}^n$. We also use the standard notations $x_\Lambda = (x_k, k \in \Lambda)$ and $x_{-\Lambda} = x_{S\setminus\Lambda}$ for $\Lambda \subseteq S$, and $x_{-k} = x_{S\setminus\{k\}}$ for $k \in S$. There is then a one-to-one relation between the sample space Ω and the power set of S, $\mathcal{P}(S) = \{\Lambda | \Lambda \subseteq S\}$. Corresponding to an $x \in \Omega$ we have the set $\Lambda = \{k \in S | x_k = 1\} \in \mathcal{P}(S)$ and corresponding to a set $\Lambda \in \mathcal{P}(S)$ we have the vector $\chi(\Lambda) = (I(k \in \Lambda), k \in S)$, where $I(\cdot)$ is the indicator function. Any function $U(x), x \in \Omega$ can then be represented by a set of interaction parameters $\{\beta_U(\Lambda), \Lambda \in \mathcal{P}(S)\}$ defined by

$$U(x) = \sum_{\Lambda \in \mathcal{P}(S)} \beta_U(\Lambda) \prod_{k \in \Lambda} x_k. \tag{1}$$

Clearly $U(x)$ is uniquely given by $\{\beta_U(\Lambda), \Lambda \in \mathcal{P}(S)\}$. To see how $\{\beta_U(\Lambda), \Lambda \in \mathcal{P}(S)\}$ is uniquely given by $U(x)$, insert $x = \chi(\Lambda)$ in (1) for a $\Lambda \in \mathcal{P}(S)$ and solve for $\beta_U(\Lambda)$ to get

$$\beta_U(\Lambda) = U(\chi(\Lambda)) - \sum_{A \subset \Lambda} \beta_U(A). \tag{2}$$

Thereby $\beta_U(\Lambda)$ can be computed recursively, first computing $\beta_U(\emptyset) = U(\chi(\emptyset))$, then $\beta_U(\{k\})$ for all $k \in S$, then $\beta_U(\Lambda)$ for all $\Lambda \subseteq S$ with $|\Lambda| = 2$ and so on. A direct implementation of (2) is, however, computationally inefficient as it requires repeated calculations of the the same sum, see Appendix A for a computationally more efficient variant. The number of elements in $\{\beta_U(\Lambda), \Lambda \in \mathcal{P}(S)\}$ is 2^n, so to represent a function as described here is in practice only feasible for small values of n. In the next section we consider functions with a Markov property, where fewer interaction parameters are necessary to represent a function.

2.2 Markov property for functions of binary variables

Let S be as above. Following Besag (1974) we define neighborhood and clique systems for S.

Definition 1. *A collection $N = \{N_1, \dots, N_n\}$ is a neighborhood system for the set S if $N_k \subseteq S \backslash \{k\}$ for all $k \in S$, and $k \in N_l \Leftrightarrow l \in N_k$ for all distinct pairs $k, l \in S$. If $k \in N_l$ for $k, l \in S$ we say that k and l are neighbors.*

Definition 2. *A set $\Lambda \subseteq S$ is said to be a clique if $k \in N_l$ for all distinct pairs $k, l \in \Lambda$. We let $\mathcal{C}$ denote the set of all cliques.*

Toy example: *If $S = \{1,2,3,4\}$ one may have $N_1 = \{2,3\}$, $N_2 = \{1,3\}$, $N_3 = \{1,2,4\}$ and $N_4 = \{3\}$, in which case we get $\mathcal{C} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{2,3\}, \{3,4\}, \{1,2,3\}\}$.*

Definition 3. *A function of binary variables, $U(x), x \in \{0,1\}^n$, is said to have a Markov property with respect to a neighborhood system N if it can be written in the form*

$$U(x) = \sum_{\Lambda \in \mathcal{C}} V_\Lambda(x_\Lambda), \tag{3}$$

where $\mathcal{C}$ is the set of all cliques and $V_\Lambda(x_\Lambda)$ is an arbitrary function of x_Λ. In particular $V_\emptyset$ equals a constant.

In the following we assume $U(x)$ to have a Markov property with respect to a given neighborhood system N. One should note that $U(x)$ then corresponds to an energy function of a binary MRF with respect to N, see for example Besag (1974), Kindermann and Snell (1980) or Cressie (1993). We return to this correspondence in Section 3, but first we consider how the Markov property induces vanishing interaction parameters.

Theorem 1. *Let $U(x)$ have a Markov property with respect to a neighborhood system N, let $\mathcal{C}$ be the set of all cliques corresponding to N, and let $\{\beta_U(\Lambda), \Lambda \in \mathcal{P}(S)\}$ be given by (1). Then $\beta_U(\Lambda) = 0$ for all $\Lambda \notin \mathcal{C}$.*

A proof of the theorem is given in Appendix B. To find the interaction parameters of $U(x)$ we thereby only need to compute $\beta_U(\Lambda)$ for $\Lambda \in \mathcal{C}$. Moreover, the number of terms in the sum in (1) can clearly be reduced by excluding terms that correspond to interaction parameters equal to zero. In the following we replace (1) by what will be our canonical representation of $U(x)$,

$$U(x) = \sum_{\Lambda \in \mathcal{B}_U} \beta_U(\Lambda) \prod_{k \in \Lambda} x_k, \quad \text{where} \quad \mathcal{B}_U = \bigcup_{\Lambda \in \mathcal{P}(S) : \beta_U(\Lambda) \neq 0} \mathcal{P}(\Lambda). \tag{4}$$

By Theorem 1 we have $\mathcal{B}_U \subseteq \mathcal{C}$. Thus, to find our canonical representation of a given $U(x)$ we must first recursively compute $\beta_U(\Lambda)$ for all cliques $\Lambda \in \mathcal{C}$, and use these values to find $\mathcal{B}_U$ and form $\beta_U = (\beta_U(\Lambda), \Lambda \in \mathcal{B}_U)$.

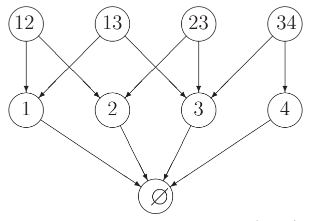


Figure 1: The unweighted graph $\mathcal{G}(\mathcal{B}_U)$ for the toy example.

Toy example (cont.): *Let S, N and $\mathcal{C}$ be as in the toy example above. Moreover let $\beta_U(\Lambda) \neq 0$ for $\Lambda \in \{\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$ and let $\beta_U(\{1,2,3\}) = 0$. Then $\mathcal{B}_U = \{\emptyset,$ $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{1,2\}$, $\{1,3\}$, $\{2,3\},\{3,4\}\}$. Note that here $\mathcal{B}_U$ does not depend on the value of $\beta_U(\Lambda)$ for $\Lambda \in \{\emptyset,\{1\},\{2\},\{3\},\{4\}\}$.*

As also seen in the toy example, we clearly may have $\beta_U(\Lambda) = 0$ for some $\Lambda \in \mathcal{B}_U$. We include these 'zero terms' in our representation of $U(x)$ because having $\mathcal{B}_U$ as a union of power sets simplifies our recursive algorithms discussed in Sections 4.1 and 4.2.

As illustrated for the toy example in Figure 1, the elements in $\mathcal{B}_U$ can be organized into a directed acyclic graph with one vertex for each set $\Lambda \in \mathcal{B}_U$ and where the descendents of a vertex Λ are all proper subsets of Λ. A corresponding vertex-weighted graph can be defined by storing the elements of $\{\beta_U(\Lambda), \Lambda \in \mathcal{B}_U\}$ in their respective vertices. We denote these unweighted and weighted graphs by $\mathcal{G}(\mathcal{B}_U)$ and $\mathcal{G}(\mathcal{B}_U, \beta_U)$, respectively. In the following we use vertex-weighted graphs to define and implement our recursive algorithms, whereas we use the corresponding unweighted graphs to illustrate the procedures.

One should note that if we have two functions of binary variables represented by vertex-weighted graphs, the vertex-weighted graph representation of the sum of the two functions is easy to find. For example, let the vertex-weighted graph representations of $U_1(x), x = (x_k, k \in S)$ and $U_2(x_A), x_A = (x_k, k \in A)$ where $A \subseteq S$ be $\mathcal{G}(\mathcal{B}_{U_1}, \beta_{U_1})$ and $\mathcal{G}(\mathcal{B}_{U_2}, \beta_{U_2})$, respectively. Assume we want to "update" the graph $G = \mathcal{G}(\mathcal{B}_{U_1}, \beta_{U_1})$ to become a representation of $U(x) = U_1(x) + U_2(x_A)$. One should then first visit all the vertices of $\mathcal{G}(\mathcal{B}_{U_2}, \beta_{U_2})$, from the bottom to the top. When visiting $\Lambda \in \mathcal{B}_{U_2}$ there are two possibilities, either there is a vertex Λ also in the graph G, or there is not. If there is a vertex Λ also in G, one should add $\beta_{U_2}(\Lambda)$ to the current weight of this vertex. If there is not a vertex Λ in G, one should insert a vertex Λ in G, with associated edges, and initialize the weight of this new vertex to $\beta_{U_2}(\Lambda)$. When all vertices of $\mathcal{G}(\mathcal{B}_{U_2}, \beta_{U_2})$ have been visited, the resulting graph G is a representation of $U(x)$. However, it may include vertices with zero weight that should not be included according to (4). To get the canonical representation one must therefore once more traverse the vertices in G that have a corresponding vertex in $\mathcal{G}(\mathcal{B}_{U_2}, \beta_{U_2})$, now from top to bottom, to remove any such superfluous vertices (and associated edges). The computational complexity of the whole updating procedure for G is clearly proportional to the number of vertices in $\mathcal{G}(\mathcal{B}_{U_2}, \beta_{U_2})$.

We end this section by noting that if we have a vertex-weighted graph $\mathcal{G}(\mathcal{B}_U, \beta_U)$ representing a function $U(x)$, for example obtained as a sum of two functions as discussed above, it follows directly that $U(x)$ has a Markov property with respect to the neighborhood system $N = \{N_1, \ldots, N_n\}$ where $N_k = \{l \in S \backslash \{k\} | \{k, l\} \in \mathcal{B}_U\}$. Moreover, this neighborhood system is minimal for $U(x)$ in the sense that $U(x)$ has a Markov property with respect to a neighborhood system $N = \{N_1, \ldots, N_n\}$ if and only if $N_k \supseteq \{l \in S \backslash \{k\} | \{k, l\} \in \mathcal{B}_U\}$ for all $k \in S$.

3 Binary Markov random fields

For a general introduction to MRFs, see for example Besag (1974), Kindermann and Snell (1980) or Cressie (1993). Here we give just a brief introduction to binary MRFs to facilitate our development of exact and approximate recursive algorithms in the next section.

Let $x = (x_k, k \in S) \in \Omega = \{0, 1\}^n$ be a vector of binary variables. Then x is a binary MRF with respect to a neighborhood system N if $p(x) > 0$ for all $x \in \Omega$, and the full conditionals, $p(x_k | x_{-k})$, fulfil the Markov property

$$p(x_k | x_{-k}) = p(x_k | x_{N_k}) \quad \text{for all } k \in S. \tag{5}$$

The positivity condition $p(x) > 0$ for all $x \in \Omega$ clearly gives that $p(x)$ can be expressed as

$$p(x) = c \exp\{-U(x)\}, \tag{6}$$

where c is a normalizing constant and $U(x)$ is called the energy function. The Hammersley–Clifford theorem (Besag, 1974; Clifford, 1990) gives that the most general form for $U(x)$ is given by (3). Thus, a vertex-weighted graph $\mathcal{G}(\mathcal{B}_U, \beta_U)$ representing $U(x)$ can also be used to represent the corresponding $p(x)$.

4 Recursive computations

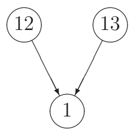
Let $S = \{1, \ldots, n\}$, let $x = (x_1, \ldots, x_n)$ be a binary MRF with respect to a neighborhood system N, and let $p(x) = c \exp\{-U(x)\}$ be the corresponding joint distribution. Assuming $U(x)$, and thereby also $p(x)$, to be represented by a vertex-weighted graph $\mathcal{G}(\mathcal{B}_U, \beta_U)$, we discuss in the next section how to decompose $p(x)$ into the product

$$p(x) = p(x_1 | x_{-1}) p(x_{-1}) \tag{7}$$

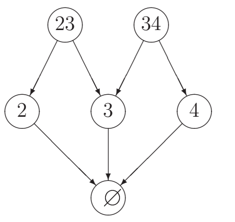
with a vertex-weighted graph representation for each of the two factors. This process can be iterated, next decomposing $p(x_{-1}) = p(x_2 | x_{-\{1,2\}}) p(x_{-\{1,2\}})$, and so on. Finally this gives

$$p(x) = \left[\prod_{k=1}^{n-1} p(x_k | x_{k+1}, \ldots, x_n) \right] p(x_n), \tag{8}$$

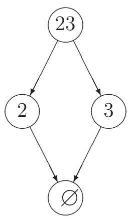
where each factor is represented by a vertex-weighted graph. In particular, the factor $p(x_n)$ is represented by a graph with only two vertices, $\emptyset$ and $\{n\}$. The normalizing constant c of $p(x)$ is included also in $p(x_n)$ and thereby this constant can be evaluated from the obvious restriction $p(x_n = 0) + p(x_n = 1) = 1$. Moreover, simulation from $p(x)$ is straightforward by a backward pass, first simulating x_n from $p(x_n)$, next x_{n-1} from $p(x_{n-1} | x_n)$ and so on.



(a) $\mathcal{G}_1(\mathcal{B}_U)$



(b) $\mathcal{G}_{-1}(\mathcal{B}_U)$



(c) $\mathcal{G}(\mathcal{P}(N_1))$

Figure 2: Unweighted graphs derived from $\mathcal{G}(\mathcal{B}_U)$ shown in Figure 1.

4.1 Exact computations

To obtain the vertex-weighted graph representations of $p(x_1|x_{-1})$ and $p(x_{-1})$ from the corresponding representation of $p(x)$, one should first split the graph $\mathcal{G}(\mathcal{B}_U, \beta_U)$ into two subgraphs $\mathcal{G}_1(\mathcal{B}_U, \beta_U)$ and $\mathcal{G}_{-1}(\mathcal{B}_U, \beta_U)$, where the first contains all vertices $\Lambda \in \mathcal{B}_U$ with $1 \in \Lambda$, and the remaining vertices are included in $\mathcal{G}_{-1}(\mathcal{B}_U, \beta_U)$. Figures 2(a) and (b) show the corresponding unweighted graphs $\mathcal{G}_1(\mathcal{B}_U)$ and $\mathcal{G}_{-1}(\mathcal{B}_U)$ for the toy example. Note that the split can be done by starting at the vertex $\{1\}$ in $\mathcal{G}(\mathcal{B}_U, \beta_U)$ and successively traverse nodes upwards while cutting loose the graph $\mathcal{G}_1(\mathcal{B}_U, \beta_U)$. The computational complexity of the operation is proportional to the number of vertices in the resulting $\mathcal{G}_1(\mathcal{B}_U, \beta_U)$. As discussed above, the vertices of $\mathcal{G}(\mathcal{B}_U, \beta_U)$ is a union of power sets, and this property is inherited by $\mathcal{G}_{-1}(\mathcal{B}_U, \beta_U)$, but not by $\mathcal{G}_1(\mathcal{B}_U, \beta_U)$. Still, however, we can, and will, use $\mathcal{G}_1(\mathcal{B}_U, \beta_U)$ to represent a probability distribution.

From (4) and the assumed Markov property in (5) we readily get

$$p(x_1|x_{-1}) = p(x_1|x_{N_1}) \propto \exp\left\{-\sum_{\Lambda\in\mathcal{B}_U:1\in\Lambda} \beta_U(\Lambda)\prod_{k\in\Lambda} x_k\right\} \tag{9}$$

and recognize the vertex-weighted graph $\mathcal{G}_1(\mathcal{B}_U, \beta_U)$ as a representation of this full conditional. To get the graph representation of $p(x_{-1})$ is less immediate. Let $U_{-1}(x_{-1})$ denote the energy function of $p(x_{-1})$, i.e. $p(x_{-1}) = c\exp\{-U_{-1}(x_{-1})\}$. Thus we have

$$U_{-1}(x_{-1}) = -\ln\left[\sum_{x_1\in\{0,1\}} \exp\left\{-\sum_{\Lambda\in\mathcal{B}_U} \beta_U(\Lambda)\prod_{k\in\Lambda} x_k\right\}\right]. \tag{10}$$

Splitting the inner sum into a sum of two sums, one sum over the vertices in $\mathcal{G}_1(\mathcal{B}_U, \beta_U)$ and one sum over the vertices in $\mathcal{G}_{-1}(\mathcal{B}_U, \beta_U)$, and using that the latter sum is not a function of x_1, we get

$$U_{-1}(x_{-1}) = \left[\sum_{\Lambda\in\mathcal{B}_U:1\notin\Lambda} \beta_U(\Lambda)\prod_{k\in\Lambda} x_k\right] + \left[-\ln\left(1+\exp\left\{-\sum_{\Lambda\in\mathcal{B}_U:1\in\Lambda} \beta_U(\Lambda)\prod_{k\in\Lambda} x_k\right\}\right)\right]. \tag{11}$$

1. Split the graph $\mathcal{G}(\mathcal{B}_U, \beta_U)$ into the two subgraphs $\mathcal{G}_1(\mathcal{B}_U, \beta_U)$ and $\mathcal{G}_{-1}(\mathcal{B}_U, \beta_U)$. Store the first of these as a representation of the full conditional $p(x_1|x_{-1})$.
2. Use the algorithm i Appendix A to compute recursively the interaction parameters for all $\Lambda \in \mathcal{P}(N_1)$ of the second term in (11) and form the corresponding canonical representation.
3. Using the procedure discussed in Section 2.2, form the canonical representation of $U_{-1}(x_{-1})$, and thereby also of $p(x_{-1})$, by adding the canonical representations of the two terms in (11).

Figure 3: Algorithm for generating vertex-weighted graph representations of $p(x_1|x_{-1})$ and $p(x_{-1})$ from the corresponding representation of $p(x)$.

We see that here $U_{-1}(x_{-1})$ is given as a sum of two functions, a situation discussed in Section 2.2. The first term is a function of x_{-1} and is in the canonical form. The second term is a function of x_{N_1} and is not yet in the canonical form. To get the canonical form for $U_{-1}(x_{-1})$ as discussed in Section 2.2 one must thereby first find the canonical form for the second term in (11). As this function has no known Markov property, interaction parameters must be computed for all $\Lambda \in \mathcal{P}(N_1)$. The computational complexity of this operation is clearly given by the number of elements in $\mathcal{P}(N_1)$. We denote the resulting vertex-weighted graph representation of $U_{-1}(x_{-1})$ by $\mathcal{G}(\mathcal{B}_{U_{-1}}, \beta_{U_{-1}})$. The total algorithm is summarized in Figure 3. One should note that the resulting $p(x_{-1})$ is a binary MRF with $S = \{2, \ldots, n\}$ and a new neighborhood system. From the vertex-weighted graph representation of $p(x_{-1})$, the (minimal) neighborhood system for $p(x_{-1})$ can be found as discussed in Section 2.2.

As the computational complexity of finding the decomposition (7) is given by the size of $\mathcal{P}(N_1)$, i.e. exponential in $|N_1|$, this is only feasible when the number of neighbors is reasonably low. When iterating the procedure we do not have general results for the cost of the algorithm. Next we propose a computationally cheaper, but approximate version of the algorithm.

4.2 Approximate recursive computations

The exact algorithm described above is not computationally feasible when the number of neighbors is large. There are two problems. First, in Item 2 of Figure 3, it requires too much computation time to compute the necessary interaction parameters. Second, even if it had been feasible to compute all interaction parameters, it would require too much computer memory to store them all.

When computing interaction parameters of frequently used MRFs (small enough to

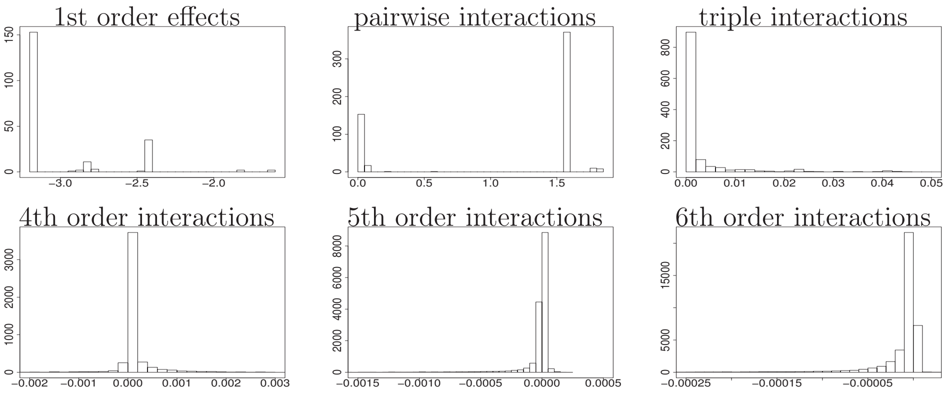


Figure 4: The interaction parameter values in a typical iteration of the recursive algorithm for an Ising model. In lexicographical order the six plots show histograms of the interaction parameters for sets Λ with $|\Lambda| = 1, 2, 3, 4, 5$ and 6, respectively. Note the difference in horizontal scale.

allow exact computations) we observe that most values are very close to zero. For example, for an Ising model Figure 4 shows the computed interaction parameters in a typical iteration of the recursive algorithm. For each $r = 1, 2, 3, 4, 5$ and 6 the figure shows one histogram for interaction parameters for sets Λ with $|\Lambda| = r$. In Item 2 of Figure 3, a natural approximation strategy is therefore, for some threshold ε, to approximate an interaction parameter to zero whenever its absolute value is less than of equal to ε. This solves the memory problem, but not the computation time problem as still all interaction parameters have to be computed in order to decide which to store and which to ignore. To cope also with the computation time problem we assume that the interaction parameter for a set Λ is close to zero whenever the interaction parameters for all $A \subset \Lambda$ with $|A| = |\Lambda| - 1$ are close to zero. Note that this is a frequently used assumption in statistics, a higher order effect can be present only if at least one corresponding lower order effect is present. We have checked this assumption in several frequently used MRFs without finding cases where it is violated, but one can clearly construct cases where the assumption is incorrect. Using this assumption in Item 2 of Figure 3 we approximate the interaction parameter of a Λ to zero, without computing its value, whenever the interaction parameters of all children vertices of Λ are (approximated to) zero.

The two approximation rules given above define our approximation. The corresponding algorithm is equal to the exact version in Figure 3 except that in Item 2 only some of the interaction parameters are computed and stored. Thus, the joint distribution $p(x)$ is decomposed into the exact full conditional $p(x_1|x_{-1})$ and an approximation $\widetilde{p}_\varepsilon(x_{-1})$ of $p(x_{-1})$. In the next step, $\widetilde{p}_\varepsilon(x_{-1})$ is decomposed into $\widetilde{p}_\varepsilon(x_2|x_{-\{1,2\}})$, which of course is only an approximation to $p(x_2|x_{-\{1,2\}})$, and after new approximations $\widetilde{p}_\varepsilon(x_{-\{1,2\}})$. Ultimately we end up with an approximate version of (8),

$$p(x) \approx \widetilde{p}_\varepsilon(x) = p(x_1|x_l, l = 2, \ldots, n) \left[\prod_{k=2}^{n-1} \widetilde{p}_\varepsilon(x_k|x_l, l = k+1, \ldots, n) \right] \widetilde{p}_\varepsilon(x_n). \tag{12}$$

The normalizing constant of $p(x)$ can be approximated by the normalizing constant of $\widetilde{p}_\varepsilon(x)$ and approximate samples from $p(x)$ can be generated by sampling from $\widetilde{p}_\varepsilon(x)$. We end this section by three remarks concerning the approximative algorithm just defined.

Remark 1. *The focus when defining the approximation is to make it computationally feasible to decompose $p(x)$ into $p(x_1|x_{-1})$ and $\widetilde{p}_\varepsilon(x_{-1})$. However, approximating interaction parameters to zero also introduces conditional independence to $\widetilde{p}_\varepsilon(x_{-1})$ that is not present in the corresponding exact $p(x_{-1})$. This becomes computationally beneficial in later iterations of the recursive algorithm.*

Remark 2. *The approximate recursive algorithm is effectively dividing the elements of the set $\mathcal{P}(N_1)$ into three groups. The first consists of sets Λ for which we compute the interaction parameters, find their values to be large and thereby store them in memory. The second group contains the sets Λ for which we compute interaction parameters and find their values to be small. The third group consists of the remaining sets Λ, for which we do not compute the interaction parameters. When the number of elements in N_1 is large it is essential for the algorithm to be feasible that most interaction parameters end up in the third group.*

Remark 3. *The quality of the approximation and the computation time and memory requirements for the approximate algorithm clearly depend on the threshold value ε. How small ε needs to be to give reasonable approximations must be explored empirically. In Section 6 we report our experience with both this and the associated computer resources required for some frequently used binary MRFs. One should note that we have defined the approximation so that the algorithm is exact for $\varepsilon = 0$.*

5 Beyond binary fields

The focus of this paper is binary MRFs. In this section, however, we briefly discuss how the above algorithms can be generalized to handle discrete MRFs with more than two possible values. Thus, in this section we let the distribution of interest be $p_z(z) = c\exp\{-U_z(z)\}$ where $z = (z_1, \ldots, z_n) \in \{0, 1, \ldots, K-1\}^n$. There are two natural strategies for generalizing the above algorithm to this situation. The first is to start by representing $U_z(z)$ by a generalized version of (1),

$$U_z(z) = \sum_{\Lambda \subseteq S} \beta_{U_z}(\Lambda, z_\Lambda) \prod_{k \in \Lambda} z_k. \tag{13}$$

Thus, instead of having only one interaction parameter for a set Λ we now have $(K-1)^{|\Lambda|}$ interaction parameters associated to Λ. Again the interaction parameters can be computed

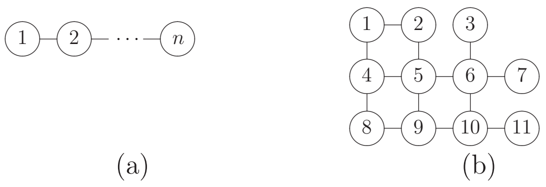


Figure 5: Two undirected graphs defining neighborhood systems. Two nodes are neighbors if and only if an edge is connecting them. A pairwise interaction, binary field which is Markov with respect to one of these neighborhood systems can be very efficiently handled by the exact algorithm described in Section 4.1. (a) A chain graph and (b) a somewhat more complex graph.

recursively and approximations following the same ideas as discussed in Section 4.2 can be defined.

The second strategy to cope with more than two possible values is to map $p_z(z)$ over to a corresponding binary problem $p_x(x)$. For example, if we have $K = 3$ or 4 classes, each variable z_k can be represented by two binary variables x_{k1} and x_{k2}.

6 Simulation examples

In this section we consider a number of binary MRFs and discuss the feasibility of our algorithms for these models. We first consider models where the exact algorithm is especially efficient. In particular we discuss how the exact algorithm reduces to the famous forward-backward algorithm when the field is Markov with respect to a neighborhood system defined by a chain graph. Finally we consider the autologistic model on a two dimensional rectangular lattice, where the exact algorithm is feasible only for small lattices.

6.1 Models where the exact algorithm is particularly efficient

Let $S = \{1, \ldots, n\}$ and assume x to be a binary MRF with respect to the neighborhood system defined by the chain graph in Figure 5(a), i.e. $N_1 = \{2\}, N_k = \{k-1, k+1\}$ for $k = 2, \ldots, n-1$, and $N_k = \{k-1\}$. The most general form of the energy function is then

$$U(x) = \alpha_0 + \sum_{k=1}^{n} \alpha_k x_k + \sum_{k=1}^{n-1} \beta_{k,k+1} x_k x_{k+1}, \qquad (14)$$

where $\alpha_0, \alpha_1, \ldots, \alpha_n$ and $\beta_{1,2}, \ldots, \beta_{n-1,n}$ are model parameters. Note that in particular the hidden (binary) Markov chain, see for example Künsch (2001), can be formulated in this form. The pairwise interaction parameters $\beta_{1,2}, \ldots, \beta_{n-1,n}$ are then related to the Markov prior, whereas the α_i's are functions of both the prior and observed data. It is

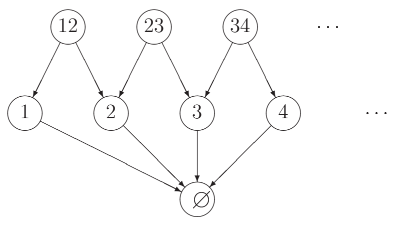


Figure 6: The graph $\mathcal{G}(\mathcal{B}_U)$ for the energy function (14) if $\beta_{k,k+1} \neq 0$ for all $k = 1, \ldots, n-1$.

easy to see that for this model the exact algorithm is equivalent to the efficient forward-backward algorithm, again see Künsch (2001). To see what makes the algorithm so efficient for this model we have to look at the graph $\mathcal{G}(\mathcal{B}_U)$ and how this develops when running the iterations. The function (14) is already in the canonical form (4) so $\beta_U(\emptyset) = \alpha_0$, $\beta_U(\{k\}) = \alpha_k$ for $k = 1, \ldots, n$ and $\beta_U(\{k, k+1\}) = \beta_{k,k+1}$ for $k = 1, \ldots, n-1$. If none of the $\beta_{k,k+1}$'s are equal to zero the graph $\mathcal{G}(\mathcal{B}_U)$ becomes as shown in Figure 6. In the first iteration of the recursive algorithm, when summing out x_1, we get that $\mathcal{G}_1(\mathcal{B}_U)$ includes only the two vertices $\{1\}$ and $\{1, 2\}$ and thereby $N_1 = \{2\}$ and $\mathcal{P}(N_1) = \{\emptyset, \{2\}\}$. In turn this gives that $\mathcal{B}_{U_{-1}}$ is of the same form as the original $\mathcal{B}_U$. Thereby, the situation repeats when summing out x_2, x_3 and so on. What is important for the efficiency is that all distributions $p(x_{-1}), p(x_{-\{1,2\}}), \ldots$ are pairwise interaction models. For this to be the case the neighbor set N_1, and corresponding sets in later iterations, must never contain more than two elements. In fact, this implies that for the current model the exact algorithm is equally efficient if marginalizing out the variables in a different order.

The requirement that N_1, and corresponding sets in later iterations, contains at most two elements is fulfilled also for other models than just (14). By drawing up the relevant graphs it is easy to check that for a suitable ordering of the variables, any pairwise interaction, binary field that are Markov with respect to a neighborhood system defined by a graph without loops, i.e. a tree, fulfils the requirement. Moreover, even for graphs that contain loops the corresponding pairwise interaction MRF may fulfil the formulated condition. An example of this with $n = 11$ is shown in Figure 5(b). Drawing up the relevant graphs, one can again check that it works in this case. However, if we add an edge between 2 and 3 the set requirement is no longer satisfied, not even after a reordering of the variables. We do not have an easy to check criterion for undirected graphs that can characterize the set of MRFs that fulfil the requirements. However, our discussion here demonstrates that the exact recursive algorithm is highly efficient for a larger class of models than where the forward-backward algorithm is typically used today. Moreover the algorithm may of course be highly efficient even if the neighborhood sets contain slightly more than two elements.

6.2 The autologistic model

The autologistic model was first introduced in Besag (1974). A binary field $x = (x_1, \ldots, x_n)$ is an autologistic model with respect to a given neighborhood system $N = (N_1, \ldots, N_n)$ if the energy function can be expressed as (4) with $\mathcal{B}_U = \{\emptyset, \{1\}, \ldots, \{n\}\} \cup \{(k, l) | l \in N_k, k, l = 1, \ldots, n\}$, i.e.

$$U(x) = \beta_U(\emptyset) + \sum_{k=1}^{n} \beta_U(\{k\}) x_k + \frac{1}{2} \sum_{k=1}^{n} \sum_{l \in N_k} \beta_U(\{k, l\}) x_k x_l. \tag{15}$$

Strictly speaking the models in Section 6.1 are thereby autologistic models. However, the term is usually reserved for models defined on a rectangular lattice, and from now on we limit the attention to such models. Thus, consider a $u \times v$ rectangular lattice and number the lattice nodes from one to $n = uv$ in lexicographical order. Except when many of the $\beta_U(\{k, l\})$'s are equal to zero, the autologistic model does not fall into the set of models that can be handled efficiently by the exact algorithm as discussed above. The algorithms in Pettitt et al. (2003), Reeves and Pettitt (2004) and Friel and Rue (2007) are essentially the same as our exact algorithm. They conclude that for a model with a first order neighborhood system the algorithm is computationally feasible only when the number of columns, v, is less than or equal to 20. The number of rows, u, can be large. For larger neighborhoods the lattice size that is feasible by the exact algorithm is even smaller. In the following we evaluate empirically the approximation quality and computation time of the approximate algorithm of Section 4.2 for some autologistic models.

6.3 The Ising model

We first consider the Ising model, i.e. an autologistic model with a first order neighborhood system (Besag, 1986). The energy function can be defined as

$$U(x) = -\frac{\theta}{2} \sum_{k=1}^{n} \sum_{l \in N_k} I(x_k = x_l). \tag{16}$$

Using that for binary variables we have $I(x_k = x_l) = x_k x_l + (1 - x_k)(1 - x_l)$, this can easily be rewritten to the form in (15) and gives

$$\begin{aligned} \beta_U(\emptyset) &= -\frac{\theta}{2} \sum_{k=1}^{n} |N_k|, \\ \beta_U(\{k\}) &= |N_k| \theta, \text{ for } k = 1, \ldots, n, \\ \beta_U(\{k, l\}) &= -2\theta, \text{ for } l \in N_k, k, l = 1, \ldots, n. \end{aligned} \tag{17}$$

We first consider the approximation quality for a small 15×15 lattice, for which exact computations are feasible. For each $\theta = \theta_{\text{true}} = 0.4$, 0.6 and 0.8 we generate an exact sample x from the Ising model and thereafter, separately for each of the three realizations,

Table 1: Results for the Ising model on 15×15 lattice: For realizations x generated from the Ising model for each of the parameter values $\theta_{\text{true}} = 0.4$, 0.6 and 0.8, values of $d_0(\varepsilon, x)$ for the 15×15 lattice. Results are given for different values of ε, for the pseudo likelihood and block pseudo likelihood approximations, and for two variants of RDA (Friel et al., 2009).

	15×15 lattice		
$\varepsilon \backslash \theta_{\text{true}}$	0.4	0.6	0.8
10^{-1}	$1.49 \cdot 10^{-1}$	$1.87 \cdot 10^{-1}$	$6.70 \cdot 10^{-1}$
10^{-2}	$8.12 \cdot 10^{-3}$	$1.07 \cdot 10^{-1}$	$1.15 \cdot 10^{-1}$
10^{-3}	$3.20 \cdot 10^{-3}$	$2.61 \cdot 10^{-2}$	$1.15 \cdot 10^{-1}$
10^{-4}	$1.57 \cdot 10^{-3}$	$9.60 \cdot 10^{-3}$	$4.42 \cdot 10^{-2}$
10^{-5}	$3.47 \cdot 10^{-4}$	$2.35 \cdot 10^{-3}$	$3.22 \cdot 10^{-3}$
10^{-6}	$3.32 \cdot 10^{-5}$	$2.44 \cdot 10^{-4}$	$1.92 \cdot 10^{-4}$
pl	$1.60 \cdot 10^{-1}$	$1.19 \cdot 10^{-1}$	$8.82 \cdot 10^{-1}$
block-pl	$1.77 \cdot 10^{-1}$	$1.11 \cdot 10^{-1}$	$9.81 \cdot 10^{-2}$
RDA-5	$4.71 \cdot 10^{-5}$	$2.21 \cdot 10^{-3}$	$6.93 \cdot 10^{-2}$
RDA-10	$2.41 \cdot 10^{-5}$	$3.91 \cdot 10^{-5}$	$3.67 \cdot 10^{-3}$

consider the resulting posterior distribution for θ given x, i.e. $p(\theta|x)$. As prior for θ we adopt a uniform (improper) distribution on $[0, \infty)$. We compute the exact posteriors (i.e. $\varepsilon = 0$) and corresponding approximations $\widetilde{p}_\varepsilon(\theta|x)$ for $\varepsilon = 10^{-s}$, $s = 1, 2, 3, 4, 5$ and 6. More precisely, we compute $p(\theta|x)$ and $\widetilde{p}_\varepsilon(\theta|x)$ for a mesh of θ values and use interpolating spline for $\ln p(\theta|x)$ and $\ln \widetilde{p}_\varepsilon(\theta|x)$, respectively, to interpolate the results. Finally, we numerically evaluate

$$d_0(\varepsilon, x) = \int_0^\infty |\widetilde{p}_\varepsilon(\theta|x) - p(\theta|x)| \, \mathrm{d}\theta \tag{18}$$

for each realization x and value ε, see the results in Table 1. In all three cases we see that the approximation is not very accurate for $\varepsilon = 10^{-1}$, but becomes very good for smaller values of ε. Not surprisingly, smaller values of ε is necessary to obtain a good approximation when θ_{true} is larger. For comparison we also compute the same quantities when using the pseudo likelihood approximation to $p(\theta|x)$, for a pseudo block likelihood approximation with 15×5 blocks, and for two reduced dependence approximations (RDA), see Friel et al. (2009). Recalling that we are considering a $u \times v$ lattice, the RDA computes exact normalizing constants for $r \times v$ sublattices, where $r < u$. We try the approximation for $r = 5$ and $r = 10$. The results are again given in Table 1 and we see that except for the larger values of ε, the results for $\widetilde{p}_\varepsilon(\theta|x)$ are much better than for the pseudo likelihood and pseudo block likelihood approximations. The results for RDA are comparable to $\widetilde{p}_\varepsilon(\theta|x)$ for $\varepsilon = 10^{-6}$. To get a visual impression of the accuracy of the approximations, Figure 7 gives, for the $\theta_{\text{true}} = 0.8$ case, the exact $p(\theta|x)$ and the approximations $\widetilde{p}_\varepsilon(\theta|x)$ for the various values of ε.

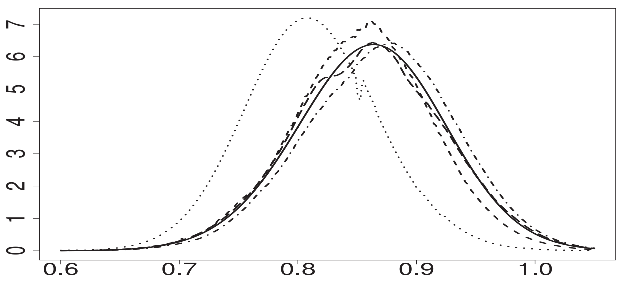


Figure 7: Results for the Ising model on a 15×15 lattice: $p(\theta|x)$ (solid) and $\widetilde{p}_\varepsilon(\theta|x)$ for $\varepsilon = 10^{-1}$ (dotted) , 10^{-2} (dashed), 10^{-3} (dot-dashed) and 10^{-4} (long dashed). With the resolution used here the approximations for $\varepsilon = 10^{-5}$ and 10^{-6} are visually indistinguishable from the exact $p(\theta|x)$.

Next we repeat the same exercise for a 100×100 lattice. Then the exact posteriors are not available, so instead of $d_0(\varepsilon, x)$ we consider

$$d(\varepsilon, x) = \int_0^\infty \left|\widetilde{p}_\varepsilon(\theta|x) - \widetilde{p}_{\varepsilon/10}(\theta|x)\right| \mathrm{d}\theta \tag{19}$$

for the same values of θ_{true} and ε as in the 15×15 lattice case. The results are again summarized in Table 2. For the $\theta_{\text{true}} = 0.8$ case, the computer resources required to compute $\widetilde{p}_\varepsilon(\theta|x)$ for $\varepsilon = 10^{-5}$ and 10^{-6} are very large, so we did not run these cases. In the evaluation of the pseudo likelihood, pseudo block likelihood and RDA approximations we compute the difference to $\widetilde{p}_\varepsilon(\theta|x)$ with $\varepsilon = 10^{-6}$. For the pseudo block likelihood we use blocks of size 100×5 and for RDA we again consider $r \times v$ sublattices for $r = 5$ and $r = 10$. As one would expect, the approximations are less accurate for the 100×100 lattice than for the 15×15 lattice. However, for the $\theta_{\text{true}} = 0.4$ and 0.6 cases, the results for the smaller values of ε clearly indicate good approximations. For $\theta_{\text{true}} = 0.8$ the results are much less favorable. The results for RDA are again comparable to the results for $\widetilde{p}_\varepsilon(\theta|x)$ for $\varepsilon = 10^{-6}$, whereas the results for the pseudo likelihood and pseudo block likelihood approximations are again much less favorable.

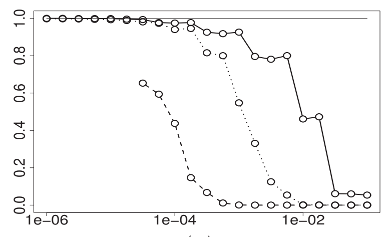
We also evaluated our approximation by estimating the mean acceptance probability of an independent proposal Metropolis–Hastings algorithm with target distribution $p(x)$ and proposal distribution $\widetilde{p}_\varepsilon(x)$. For each of $\theta = 0.4$, 0.6 and 0.8 we generated 1 000 independent realizations from $p(x)$ by the coupling from the past algorithm (Propp and Wilson, 1996) and for various values of ε, 1 000 realizations from the approximation $\widetilde{p}_\varepsilon(x)$. We then estimated the mean acceptance probability by taking the mean of the 1 000 000 acceptance probabilities generated by combining each of the 1 000 realizations from $p(x)$ with each of the 1 000 realizations from $\widetilde{p}_\varepsilon(x)$. The results are shown in Figure 8(a), where the solid, dotted and dashed curves are for $\theta = 0.4$, 0.6 and 0.8, respectively. Again we have no available results for small values of ε when $\theta = 0.8$ because the computations

Table 2: Results for the Ising model on 100×100 lattice: For realizations x generated from the Ising model for each of the parameter values $\theta_{\text{true}} = 0.4$, 0.6 and 0.8, values of $d(\varepsilon, x)$ for the 100×100 lattice. Results are given for different values of ε, for the pseudo likelihood and block pseudo likelihood approximations, and for two variants of RDA (Friel et al., 2009).

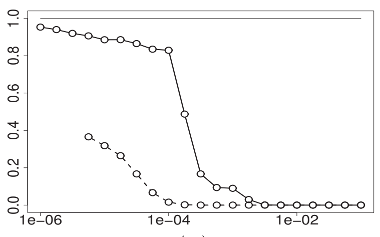
	100×100 lattice		
$\varepsilon \backslash \theta_{\text{true}}$	0.4	0.6	0.8
10^{-1}	$3.98 \cdot 10^{-1}$	2.00	2.00
10^{-2}	$1.70 \cdot 10^{-1}$	$9.02 \cdot 10^{-1}$	1.64
10^{-3}	$6.44 \cdot 10^{-2}$	$4.44 \cdot 10^{-2}$	1.11
10^{-4}	$1.16 \cdot 10^{-2}$	$2.46 \cdot 10^{-2}$	NA
10^{-5}	$3.45 \cdot 10^{-3}$	$2.85 \cdot 10^{-3}$	NA
pl	$6.26 \cdot 10^{-1}$	$4.35 \cdot 10^{-1}$	NA
block-pl	$8.63 \cdot 10^{-2}$	$2.82 \cdot 10^{-1}$	NA
RDA-5	$3.29 \cdot 10^{-3}$	$5.13 \cdot 10^{-3}$	NA
RDA-10	$3.29 \cdot 10^{-3}$	$3.47 \cdot 10^{-3}$	NA

required too much computation time. Consistent with what we saw in Table 1 and 2 we find that the approximation is indeed very good for $\varepsilon \leqslant 10^{-4}$ for $\theta = 0.4$ and 0.6, whereas for $\theta = 0.8$ we do not get good approximations within reasonable computation time. One should note that unlike $\widetilde{p}_\varepsilon(x)$, the pseudo likelihood, pseudo block likelihood and RDA approximations are not proper distributions, so a corresponding evaluation of these is not possible.

Above we have evaluated the approximation quality as a function of θ and ε. To evaluate the usefulness of the approach one also needs to consider the computation times required. The computations involved can be divided into 1) what is necessary to establish the approximation and 2) the additional time required to generate one realization from the approximation or to evaluate a corresponding likelihood for a given x. For our implementation and a 100×100 lattice, Table 3 shows computation times for $\widetilde{p}_\varepsilon(x)$ and RDA. We do not give computation times for the pseudo likelihood and pseudo block likelihood approximations as the inferior approximation quality for these methods makes such numbers less interesting. In the original definition of RDA in Friel et al. (2009), RDA is defined for a stationary MRF and this makes the approximation especially efficient. Stationarity can also be used to define a more efficient version of our approximation. If the target distribution $p(x)$ is defined on a regular lattice, the parameters $\beta_U(\Lambda)$ are stationary and we are summing out the variables in the lexicographical order, we quickly get into an essentially stationary phase after having summed out the first few lines of variables. Thus, in this case we only need to sum out in detail the first few lines and the last one. However, we find it more interesting to study the computational efficiency of the algorithms without requiring stationarity, so for both $\widetilde{p}_\varepsilon(x)$ and RDA the figures in Table 3 are for implementations



(a)



(b)

Figure 8: Results for (a) the Ising and (b) the 5×5 neighborhood models on 100×100 lattices: Estimated acceptance rates as a function of ε for an independent proposal Metropolis–Hastings algorithm with target distribution $p(x)$ when the corresponding approximation $\widetilde{p}_\varepsilon(x)$ is used as proposal distribution. In (a) the solid, dotted and dashed curves show results for $\theta = 0.4$, 0.6 and 0.8, respectively, whereas in (b) the solid and dashed curves show results for $\theta = 0.1$ and 0.15, respectively.

Table 3: Results for the Ising model on a 100×100 lattice: Computation time used to establish the approximations $\widetilde{p}_\varepsilon(x)$ and RDA, and the additional time necessary to evaluate the likelihood for a new data set or (for $\widetilde{p}_\varepsilon(x)$) to generate one realization from the approximate distribution. The numbers are computation times in seconds on a machine with an Intel Quad-Core X5365 3.0Hz cpu.

	Time to establish approximation			Additional time per realization		
$\varepsilon \backslash \theta$	0.4	0.6	0.8	0.4	0.6	0.8
10^{-1}	0.11	0.12	0.19	0.07	0.06	0.05
10^{-2}	0.19	0.39	2.32	0.09	0.12	0.21
10^{-3}	0.82	3.83	60.01	0.17	0.41	1.52
10^{-4}	2.38	27.89	5 640.50	0.27	1.10	34.56
10^{-5}	15.21	279.00	370 303.80	0.68	3.70	921.93
10^{-6}	41.27	1 948.37	NA	1.14	11.52	NA
RDA-5	19.0	19.0	19.0	0.13	0.13	0.13
RDA-10	1620.00	1620.00	1620.00	0.13	0.13	0.13

not assuming stationarity. Recalling that $\widetilde{p}_\varepsilon(x)$ for $\theta = 0.4$ and 0.6 gave very good approximations for $\varepsilon = 10^{-4}$, we see that these approximations are also efficiently available, especially for $\theta = 0.4$. For $\theta = 0.8$ the situation is less favorable and whether or not the approximation is of any use here depends on the problem of interest. By construction, the RDA computation times do not vary with θ, and the time to evaluate a realization neither varies with level of approximation. Comparing the computation times to establish approximations for $\widetilde{p}_\varepsilon(x)$ and RDA we see that the computation times are comparable for similar approximation qualities when $\theta = 0.4$ and 0.6. For $\theta = 0.8$ the results are of less interest as no good approximations are produced in this case. The computation times to evaluate the likelihood for an additional data set is typically lower for RDA than for $\widetilde{p}_\varepsilon(x)$, but as one can not generate realizations from the RDA model these figures are of less interest for RDA than for $\widetilde{p}_\varepsilon(x)$.

6.4 A pairwise interaction 5×5 neighborhood model

Next we consider an autologistic model on a rectangular 100×100 lattice with a 5×5 neighborhood. We adopt torus boundary conditions so all nodes have 24 neighbors. We consider the locations of the nodes to be positioned from 1 to 100 along each coordinate axis and let $D(k,l)$ denote the Eucledian distance between nodes k and l. The energy function we consider can then be expressed as

$$U(x) = -\frac{\theta}{2} \sum_{k=1}^{n} \sum_{l \in N_k} \frac{I(x_k = x_l)}{D(k,l)}. \tag{20}$$

We consider the model for $\theta = 0.1$ and 0.15, and again apply our approximate recursive algorithm for various values of ε. To evaluate the quality of the resulting approximations we report the mean acceptance probability of an independent proposal Metropolis–Hastings algorithm, corresponding to what we did for the Ising model above. The results are given in Figure 8(b), and Table 4 gives corresponding computation times. We do not include results for the pseudo likelihood, pseudo block likelihood and RDA approximations. The inferior results for the pseudo likelihood and pseudo block likelihood approximations for the simple Ising model make these approximations less interesting for this more complicated model, whereas RDA is not defined for torus boundary conditions and it is not clear how to generalize the approximation strategy to this situation.

7 A Bayesian model for presence/absence of deer

In this section we reconsider a dataset of census counts of red deer in the Grampians Region of north-east Scotland. Our intention is to demonstrate how the proposed approximation scheme can be used to analyze such data, not to present a full analysis of the given dataset. A thorough description of the dataset can be found in Buckland and Elston (1993) and Augustin et al. (1996). Following these two articles we reduce the counts to

Table 4: Results for the pairwise interaction 5×5 neighborhood model on a 100×100 lattice: Computation time used to establish the approximation $\widetilde{p}_\varepsilon(x)$ and the additional time necessary for generating one realization from the approximate distribution. The numbers are computation times in seconds on a the same machine as specified in the caption of Table 3.

Time to establish approximation

$\varepsilon \backslash \theta$	0.10	0.15
10^{-1}	0.65	0.62
10^{-2}	1.57	2.63
10^{-3}	37.73	104.66
10^{-4}	402.82	2 085.03
10^{-5}	2 741.99	61 086.34
10^{-6}	32 478.77	NA

Additional time per realization

$\varepsilon \backslash \theta$	0.10	0.15
10^{-1}	0.30	0.19
10^{-2}	0.34	0.24
10^{-3}	0.54	0.95
10^{-4}	2.56	7.57
10^{-5}	8.10	73.60
10^{-6}	36.30	NA

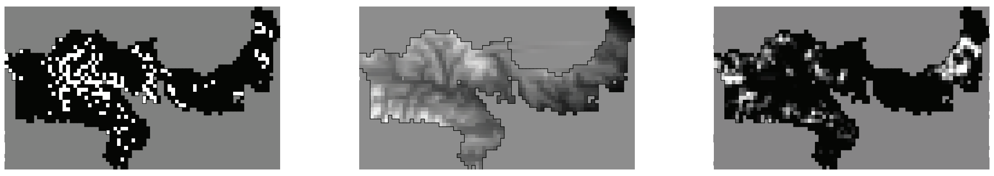


Figure 9: Data for presence/absence of deer in the Grampians Region of Scotland. Left: Observed presence (white)/absence (black) of red deer in 1 km squares. Middle: Altitude covariate (dark for low values). Right: Mires covariate (dark for low values)

presence/absence data in 1 km squares, see Figure 9. We define the response variable x_k to be 0 if deer are absent from square k and 1 if deer are present. In a regression model where observed presence/absence in regions are assumed independent, Buckland and Elston (1993) found four covariates to be significant, namely altitude, mires and the Cartesian coordinates easting and northing. In our analysis we use these four variables to form 14 potential covariates, namely the four variables themselves after standardizing them to have zero mean and unit standard deviation, each of the four standardized variables squared, and all pairwise products. Including also a potential covariate constantly equal to 1, we get a total of 15 potential covariates. Allowing each of these covariates to be included or excluded from the model we obtain $2^{15} = 32\,768$ possible covariate sets, which we number from 1 to 32 768. For each $s \in \{1, \ldots, 32\,768\}$ we adopt an autologistic model similar to Hoeting et al. (2000),

$$p(x|\theta, s, \varphi^s) \propto \exp\left\{\frac{\theta}{2}\sum_{k=1}^{n}\sum_{l \in N_k} I(x_k = x_l) + \sum_{k=1}^{n}\left[x_k \sum_{i=1}^{\nu(s)} \varphi_i^s z_{ki}^s\right]\right\}, \tag{21}$$

where $\nu(s)$ is number of covariates in covariate set number s, and $\varphi^s = (\varphi_1^s, \ldots, \varphi_{\nu(s)}^s)$ and $z_k^s = (z_{k1}^s, \ldots, z_{k,\nu(s)}^s)$ are the corresponding parameter vector and vector of covariates for cell number k, respectively. We adopt (21) as likelihood. As our focus is mainly to demonstrate how our approximation can be used to analyze the given data, we adopt a simple uniform prior for s. Following Hoeting et al. (2000) we use for θ a broad gamma prior with mean 2/3 and standard deviation $\sqrt{2/9}$, and given s we assume independent normal priors with zero mean and variance 20 for each component in φ^s. The resulting posterior is

$$p(\theta, s, \varphi^s|x) \propto p(x|\theta, s, \varphi^s)p(\theta)p(s)\prod_{i=1}^{\nu(s)} p(\varphi_i^s|s), \tag{22}$$

where one should remember that the likelihood factor includes an intractable normalizing constant which is a function of the model parameters. We approximate the posterior by replacing the likelihood with the corresponding approximation $\widetilde{p}_\varepsilon(x|\theta, s, \varphi^s)$, and try both $\varepsilon = 10^{-3}$ and 10^{-4}. We simulate from the approximate posterior by a reversible jump algorithm (Green, 1995). In each iteration we either, with probability 0.5, propose a new value for one of the current parameters or, with probability 0.25 for each, propose to add a new covariate or to remove one of the current covariates. When a potential new value is to be sampled for φ_i, say, we generate this from a $\mathrm{N}(\varphi_i, 0.1^2)$ distribution. To propose to remove one covariate we simply draw at random from the current set of covariates what covariate to remove. Finally, to propose to add a new covariate we draw at random what covariate to include from the covariates that are not included in the current model, and sample the corresponding potential parameter value from its prior distribution.

We run the algorithm for 1 000 000 iterations for each of the two ε values. Except for Monte Carlo error we found no difference in the results of the two runs. In the following we report results for the $\varepsilon = 10^{-3}$ run. From a visual inspection of the trace plots we concluded that our simple proposal distributions give a Markov chain with reasonable convergence

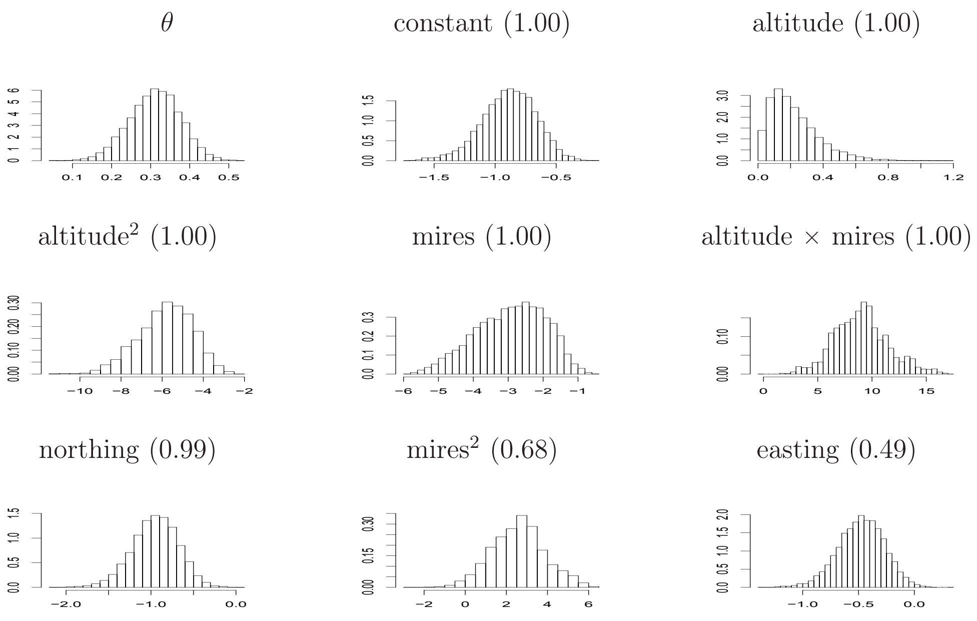


Figure 10: Deer example: Estimated marginal posterior distributions for the spatial interaction parameter θ and the parameters associated to the eight covariates with highest posterior probabilities for being included in the model. The posterior probability for being included in the model is given in parenthesis. The histograms shown estimate the marginal posterior distributions given that the covariate is included in the model.

and mixing properties. Better tuning of the proposal variances is clearly possible, but this is a tedious and time consuming process with the number of parameters we have, so we did not try this. The algorithm seem to converge in less than 50 000 iterations. We also run the algorithm with different initial states, but found no indication of convergence problems. The acceptance rates, after convergence, for the random walk, add a new covariate, and remove a covariate proposals are 0.74, 0.10 and 0.10, respectively.

The posterior gives significant probability to a large number of the possible covariate sets. The most probable covariate set has probability 4.6% and includes seven covariates, namely the constant, altitude, altitude squared, mires, the product of altitude and mires, northing, and mires squared. The same seven covariates have posterior marginal probabilities larger than a half for being included in the model, see Figure 10. Most of the posterior probability is given to covariate sets close to the most probable one. Letting A_0 be the most probable set of covariates and A an arbitrary other covariate set, one can for example see this by summing the posterior probabilities for all covariate sets A for which $|(A \backslash A_0) \cup (A_0 \backslash A)| \leqslant q$. For $q = 1, \ldots, 5$ this gives 0.19, 0.43, 0.68, 0.85 and 0.95, respectively. The posterior mean of the spatial interaction parameter θ is 0.31, see Figure

10 for probability histograms of the simulated values for both θ and eight other model parameters. The spatial interaction is weak, but clearly significant. To further study the importance of spatial interaction we also tried a similar non-spatial model, i.e. with $\theta = 0$. This gives a very different distribution for some of the regression coefficients. For example, the posterior probabilities for easting and easting squared to be included in the model are both estimated to 1.00 in the non-spatial model, whereas the corresponding figures in the spatial model are 0.49 and 0.33, respectively.

8 Closing remarks

Using a canonical representation of binary MRFs we define exact and approximate recursive algorithms for this model class. The exact algorithm is essentially the same as in Reeves and Pettitt (2004) and Friel and Rue (2007). What is new is to combine the canonical representation and the recursive scheme, which enables us to define the corresponding approximative recursive algorithm by thresholding small interaction parameters to zero. It is not obvious that this will produce a good approximation. Moreover, so far we do not have theoretical results available for the quality of the approximation. Thereby it is essential to explore the quality of the approximate algorithm empirically, as we have done in a number of simulation examples. In particular we obtained accurate approximations for the popular Ising model.

Our examples demonstrate that the procedure we propose is feasible in situations of practical importance. However, the results also show the limitations of the approach. For example, our approximation procedure is not feasible for the Ising model with $\theta = 0.8$ unless one is willing to do a lot of computations. We have also tested our procedure on models not discussed in this paper, including higher order interaction MRFs (Descombes et al., 1995; Tjelmeland and Besag, 1998) and hidden MRF models. What is important for the practicality of the approximate algorithm seems not to be the interaction level, but the degree of correlation between variables in $p(x)$.

Some examples of how the approximation can be applied are given in the examples. In a model with few parameters, the likelihood function or the posterior distribution can be explored as in Section 6 by substituting the likelihood by an approximation. For a likelihood function with a larger number of parameters this is computationally infeasible, but then MCMC can be used to explore the (approximate) posterior distribution. The example in Section 7 demonstrates this for a stochastic number of parameters. The approximation can also be applied to cope with hidden MRFs by combining the approximation with the trick detailed in Section 3.2 of Friel and Rue (2007). The (approximate) posterior can again be explored deterministically if the dimension of the parameter space is low, or by an MCMC procedure if the model has many parameters.

Our setup can be modified in several ways. First, we use basis functions $f_\Lambda(x) = \prod_{i\in\Lambda} x_i$, $\Lambda \in \mathcal{P}(S)$ to define a canonical representation. One may imagine other basis functions. What is important for the efficiency of the algorithm is that the corresponding parameters $\beta_U(\Lambda)$ can be efficiently computed and that most of these parameters can be

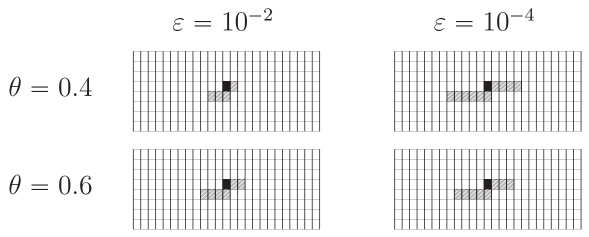


Figure 11: Conditional dependence structure for $\widetilde{p}_\varepsilon(x_k|x_l, l = k+1, \ldots, n)$ corresponding to an Ising model when variables of summed out in the lexicographical order. Results of shown for a k well away from lattice borders and for two values of θ and two values of ε. The node k is black and other nodes that influence $\widetilde{p}_\varepsilon(x_k|x_l, l = k+1, \ldots, n)$ are shaded.

approximated to zero. Second, by maximizing over each variable in turn instead of summing them out, one may define approximate variants of the Viterbi algorithm (Künsch, 2001). Third, as discussed in Section 6.3, a more efficient version of our algorithm can be defined whenever $p(x)$ is defined on a regular lattice, the parameters $\beta_U(\Lambda)$ are stationary and the variables are summed out in the lexicographical order. Last, in all our examples we are summing out the lattice variables in the lexicographical order. In simulations not reported here we have also tried other orderings of the variables, including orderings based on the multigrid (Goodman and Sokal, 1989) and divide and conquer (Golub and van Loan, 1996; Rue, 2001) ideas, but we have so far not been able to identify better summing orders than the lexicographical one. Still, however, we suspect that better summing orders exist.

It should be noted that the approximate distribution $\widetilde{p}_\varepsilon(x)$ is a partially ordered Markov model (POMM), see Cressie and Davidson (1998). The induced partial ordering can be studied by looking at $\widetilde{p}_\varepsilon(x_k|x_l, l = k+1, \ldots, n)$. This distribution is only a function of a subset of $x_l, l = k+1, \ldots, n$, partly because of the Markov property of $p(x)$ and partly as a result of the approximations. Summing out the variables in an Ising model in the lexicographical order, Figure 11 shows the set of variables that $\widetilde{p}_\varepsilon(x_k|x_l, l = k+1, \ldots, n)$ is a function of for a node k well away from lattice borders. Results are shown for two values of θ and two values of ε. Corresponding to what is intuitively reasonable, the approximate conditional distribution is a function of nodes that are close to k, and it becomes a function of more variables when θ increases or ε decreases. Even though our $\widetilde{p}_\varepsilon(x)$ is formally a POMM, the procedure we propose to construct the POMM is very different from what is discussed in Cressie and Davidson (1998). Our approximation algorithm is automatically finding a suitable partial ordering that fits to the $p(x)$ of interest, whereas in Cressie and Davidson (1998) the partial ordering is a priori specified.

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 Section 7.

SUPPLEMENTAL MATERIALS

R and C code for the recursive algorithm: R-package containing code to run the exact and approximative recursive algorithms proposed in the article. (GNU zipped tar file)

References

Augustin, 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. (1974). Spatial interaction and the statistical analysis of lattice systems, *Journal of the Royal Statistical Society, Series B* **36**: 192–225.

Besag, J. (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.

Descombes, X., Mangin, J., Pechersky, E. and Sigelle, M. (1995). Fine structures preserving model for image processing, *Proc. 9th SCIA 95, Uppsala, Sweden*, pp. 349–356.

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 Statistics* **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. (1995). Annealing Markov chain Monte Carlo with applications to ancestral inference, *Journal of American Statistical Association* **90**: 909–920.

Golub, G. H. and van Loan, C. F. (1996). *Matrix Computations, 3rd edn*, John Hopkins University Press, Baltimore.

Goodman, J. and Sokal, A. D. (1989). Multigrid Monte Carlo method. Conceptual foundations, *Physical Review D* **40**: 2035–2071.

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

Heikkinen, J. and Högmander, H. (1994). Fully Bayesian approach to image restoration with an application in biogeography, *Applied Statistics* **43**: 569–582.

Hoeting, J. A., Leecaster, M. and Bowden, D. (2000). An improved model for spatially correlated binary responses, *Journal of Agricultutal, Biological, and Environmental Statistics* **5**: 102–114.

Huang, F. and Ogata, Y. (2002). Generalized pseudo-likelihood estimates for Markov random fields on lattice, *Annals of the Institute of Statistical Mathematics* **54**: 1–18.

Kindermann, R. and Snell, J. L. (1980). *Markov random fields and their applications*, American Mathematical Society, Providence, R.I.

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

Murray, I. (2007). *Advances in Markov chain Monte Carlo methods*, PhD thesis, Gatsby Computational Neuroscience Unit, University College London.

Pettitt, A. N., Friel, N. and Reeves, R. (2003). Efficient calculation of the normalising 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. B. (1996). Exact sampling with coupled Markov chains and applications to statistical mechanics, *Random Stuctures and Algorithms* **9**: 223–252.

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

Rue, H. (2001). Fast sampling of Gaussian Markov random fields, *Journal of the Royal Statistical Society, Series B* **63**: 325–338.

Rydén, T. and Titterington, D. M. (1998). Computational Bayesian analysis of hidden Markov models, *Journal of Computational and Graphical Statistics* **7**: 194–211.

Scott, A. L. (2002). Bayesian methods for hidden Markov models: Recursive compution in the 21st century, *Journal of the American Statistical Association* **97**: 337–351.

Tjelmeland, H. and Besag, J. (1998). Markov random fields with higher order interactions, *Scandinavian Journal of Statistics* **25**: 415–433.

A Efficient recursive computation of $\{\beta_U(\Lambda), \Lambda \in \mathcal{P}(S)\}$

To see how to compute the interaction parameters $\{\beta_U(\Lambda), \Lambda \in \mathcal{P}(S)\}$ efficiently, define $\gamma_0(\Lambda) = \beta_U(\Lambda)$ for all $\Lambda \in \mathcal{P}(S)$ and

$$\gamma_g(\Lambda) = \frac{1}{g}\sum_{k\in\Lambda}\gamma_{g-1}(\Lambda\backslash\{k\}) \quad \text{for } g = 1,\ldots,|\Lambda|, \Lambda \in \mathcal{P}(S)\backslash\{\varnothing\}. \tag{23}$$

In the graph $\mathcal{G}(\mathcal{P}(S))$, $\gamma_1(\Lambda)$ is then the sum of the interaction parameters of the children of the vertex Λ, $\gamma_2(\Lambda)$ is the the sum of the interaction parameters of the grandchildren of Λ and so on. Thereby we get

$$\beta_U(\Lambda) = U(\chi(\Lambda)) - \sum_{g=1}^{|\Lambda|}\gamma_g(\Lambda). \tag{24}$$

Thus, to calculate the interaction parameters one should visit all the vertices in $\mathcal{G}(\mathcal{P}(S))$ sequentially from the bottom to the top. When visiting a vertex Λ one should first compute and store $\gamma_g(\Lambda)$ for $g = 1,\ldots,|\Lambda|$ using (23) and thereafter compute and store $\gamma_0(\Lambda) = \beta_U(\Lambda)$ by (24).

B Proof of Theorem 1

By assumption the function $U(x)$ can be written as in (3). Moreover, from our discussion in Section 2.1 it follows that each function $V_\Lambda(x_\Lambda)$ can be expressed in terms of corresponding interactions parameters, i.e.

$$V_\Lambda(x_\Lambda) = \sum_{A\in\mathcal{P}(\Lambda)}\beta_{V_\Lambda}(A)\prod_{k\in A}x_k. \tag{25}$$

Inserting (25) in (3), changing the order of summation, and using that whenever a set $\Lambda \in \mathcal{C}$ we also have that $A \in \mathcal{C}$ for any subset $A \subseteq \Lambda$, we get

$$U(x) = \sum_{A\in\mathcal{C}}\left[\sum_{\Lambda\in\mathcal{C}:A\subseteq\Lambda}\beta_{V_\Lambda}(A)\prod_{k\in A}x_k\right]. \tag{26}$$

Thus, we have shown that $U(x)$ can be expressed as

$$U(x) = \sum_{A\in\mathcal{P}(S)}\beta_U(A)\prod_{k\in A}x_k, \tag{27}$$

where

$$\beta_U(A) = \begin{cases} \sum_{\Lambda \in \mathcal{C} : A \subseteq \Lambda} \beta_{V_\Lambda}(A) & \text{if } A \in \mathcal{C}, \\ 0 & \text{otherwise.} \end{cases} \tag{28}$$

We see that (27) is in the canonical form (4). Moreover, from the discussion in Section 2.1 we know that the values of the interaction parameters are uniquely given by (4). Thus, from (28) we have that $\beta_U(\Lambda) = 0$ for all $\Lambda \notin \mathcal{C}$.

Part II:

Deterministic Bayesian inference for the p^* model.

Haakon Michael Austad and Nial Friel

Paper published in *Journal of Machine Learning Research:*
W&CP, AISTATS 2010, vol. 9.

Deterministic Bayesian inference for the p^* model

Haakon Austad
Norwegian University of Science and Technology

Nial Friel
University College Dublin

Abstract

The p* model is widely used in social network analysis. The likelihood of a network under this model is impossible to calculate for all but trivially small networks. Various approximation have been presented in the literature, and the pseudolikelihood approximation is the most popular. The aim of this paper is to introduce two likelihood approximations which have the pseudolikelihood estimator as a special case. We show, for the examples that we have considered, that both approximations result in improved estimation of model parameters with respect to the standard methodological approaches. We provide a deterministic approach and also illustrate how Bayesian model choice can be carried out in this setting.

1 Introduction

Many probability models have been developed in order to summarise the general structure of networks. For example, the Bernoulli random graph model (Erdös and Rényi, 1959) assumes that edges are considered independent of each other; the p_1 model (Holland and Leinhardt, 1981) assumes independent edge variables. The Markov random graph model (Frank and Strauss, 1986) assumes that each pair of edges is conditionally dependent given the rest of the graph. The family of p^* or exponential random graph models (Wasserman and Pattison (1996), see also Robins et al. (2007) for a recent review) is a generalisation of the latter model and is thought to be a flexible way to model the complex dependence structure of network graphs. The p^* model is arguably the most widely used model in social network analysis.

Appearing in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9. Copyright 2010 by the authors.

Despite this popularity, the main drawback to the p^* model is that the likelihood is generally unavailable, since it involves a summation over $2^{m(m-1)/2}$ terms for a network with m nodes. Clearly the size of this summation grows super-exponentially with m. For this reason various approximations have been presented in the literature. The most widely used approximation is the pseudolikelihood estimator (Strauss and Ikeda, 1990) which dates back to (Besag, 1974). It is well understood that this approximation can give poor performance, for example in the context of the autologistic distribution (Friel et al., 2009). However no formal assessment of the performance the pseudolikelihood estimator in the context of the p^* model has yet been established, and a partial aim of this paper is to address this problem. The main contribution of this paper is to introduce and investigate the use of two likelihood approximations which have the pseudolikelihood estimator as a special case.

We consider a deterministic simulation-free inference approach, avoiding the need for Markov chain Monte Carlo methods, along the lines of Rue et al. (2009). Essentially, the dimension of the parameter space is usually quite small, often with 5 or less parameters. Therefore evaluating the unnormalised posterior distribution on a fine grid is possible. An advantage of this approach is that estimates of posterior model probabilities are then available for all models nested within a model of maximal dimension using a grid evaluated for the model of maximal dimension.

There are other approaches which one could take to carry out inference for the p^* model, using simulation methods, for example. A popular choice in this setting is the Monte Carlo MLE approach of Geyer and Thompson (1992). This involves an importance sampling estimator of a ratio of normalising constants for the different parameter values of the p^* model. This method turns out to be quite difficult to implement – it involves drawing graphs from the likelihood with pre-specified initial parameters. However the choice of initial parameters is crucial, since a poorly chosen initial parameters, lying in the degenerate region of the parameter, for example, may result in simulated graphs which are empty or full. This in turn impacts negatively on the parameter estimation.

The paper is organised as follows. In section 2 we introduce the exponential random graph model. The new likelihood approximations are outlined in Section 3. While Section 5 demonstrates their performance in a number of examples, where we consider not only posterior estimation but also Bayesian model choice. Finally Section 6 offers some conclusions and discusses possible improvements to the methodology.

2 The exponential random graph model

Consider a random adjacency matrix $\boldsymbol{Y}$ representing a graph on m nodes. It can be defined by the set $\{Y_{ij} : i = 1, \dots, m; j = 1, \dots, m\}$ where the dyad $Y_{ij} = 1$ if the pair (i, j) is connected, and $Y_{ij} = 0$ otherwise. The diagonal entries of $\boldsymbol{y}$ take the value 0. The edges in the graphs could be either directed on undirected. In this study we have chosen to only look at undirected graphs, but similar techniques could be established for directed graphs as well. Let $\mathcal{Y}$ denote the set of all possible graphs on n nodes and let $\boldsymbol{y}$ be a realisation of $\boldsymbol{Y}$. The p^* model writes the probability distribution of $\boldsymbol{Y}$ as

$$\pi(\boldsymbol{y}|\beta) = \frac{\exp\{\beta^t s(\boldsymbol{y})\}}{z(\beta)} \tag{1}$$

where $s(\boldsymbol{y})$ is a known vector of sufficient statistics, for example,

$s_1(\boldsymbol{y}) = \sum_{i<j} y_{ij}$,	number of edges,
$s_2(\boldsymbol{y}) = \sum_{i<j<k} y_{ik}y_{jk}$,	number of two-stars,
$s_3(\boldsymbol{y}) = \sum_{i<j<k<l} y_{il}y_{jl}y_{kl}$,	number of three-stars,
$s_4(\boldsymbol{y}) = \sum_{i<j<k} y_{jk}y_{ik}y_{ij}$	number of triangles.

Finally β are model parameters corresponding the collection of sufficient statistics. Formally the p^* model is a Markov random field, where two edges are neighbours of one another if they share a common node. A graph with m nodes contains $n = m(m-1)/2$ edges, each of which can take values 0 or 1. Thus $\mathcal{Y}$ contains 2^n possible undirected graphs and the normalising constant $z(\beta) = \sum_{\boldsymbol{y}\in\mathcal{Y}} \exp\{\beta^t s(\boldsymbol{y})\}$ is consequently extremely difficult to evaluate for all but trivially small graphs.

2.1 Model degeneracy

Model degeneracy is an important issue concerning p^* models and was largely treated in Handcock (2003) and more recently in Rinaldo et al. (2009). The term degeneracy refers to the fact that for a network with a given number of nodes, there are so-called degenerate regions of the parameter space from which simulated networks will be either empty or full (complete). In fact, the non-degenerate region is typically a very thin region in the parameter space. Model degeneracy presents a considerable challenge for parameter estimation. Consider the mean parameterisation for the p^* model defined by $\mu = \mathbf{E}[s(\boldsymbol{y})]$. Let C be the convex hull of the set $\{s(\boldsymbol{y}) : \boldsymbol{y} \in \mathcal{Y}\}$, $ri(C)$ its relative interior and $rbd(C)$ its relative boundary. It turns out that if $\mu(\boldsymbol{\theta})$ is close to $rbd(C)$, that the model places most of the probability mass on graphs belonging to the set $deg(\mathcal{Y}) = \{\boldsymbol{y} \in \mathcal{Y} : s(\boldsymbol{y}) \in rbd(C)\}$. It is also known that the MLE exists if and only if $s(\boldsymbol{y}) \in ri(C)$ and if it exists it is unique. In the context of the likelihood approximations which we present in the next Section, it will be important to examine whether the corresponding approximate posterior distribution supports parameter values in the degenerate region.

3 Likelihood approximations

In this section we introduce three likelihood approximations for the p^* model. Suppose that the collection of all possible dyads have been ordered as $(y_1, y_2, \dots, y_n)$. From this point onward, for ease of notation, we will denote each dyad by a single index. We will also use the notation $y_{1:i}$ to denote the edges $\{y_1, \dots, y_i\}$ and y_{-i} to denote the edges $\{y_{1:i-1}, y_{i+1:n}\}$.

3.1 Maximum pseudolikelihood estimation (MPLE)

A standard approach to approximate the distribution of a Markov random field is to use a pseudolikelihood approximation, first proposed in Besag (1974) and adapted for social network models in Strauss and Ikeda (1990). This approximation consists of a product of easily normalised full-conditional distributions

$$\begin{aligned}\pi(\boldsymbol{y}|\beta) &\approx \pi_{pseudo}(\boldsymbol{y}|\beta) \\ &= \prod_{i=1}^{n} \pi(y_i|\boldsymbol{y}_{-i}, \beta) \\ &= \prod_{i=1}^{n} \frac{\pi(y_i = 1|\boldsymbol{y}_{-i}, \beta)^{y_i}}{\left[1 - \pi(y_i = 0|\boldsymbol{y}_{-i}, \beta)\right]^{y_i - 1}}.\end{aligned} \tag{2}$$

The basic idea underlying this method is the assumption of weak dependence between the variables in the graph so that the likelihood can be well approximated by the pseudolikelihood function. This leads to a fast estimation, and can be implemented using standard generalised linear model software. Nevertheless this approach turns out to be generally inadequate since it only uses local information whereas the structure of the graph is affected by global interaction.

3.2 Maximum block-pseudolikelihood estimation (MBPLE)

An obvious extension to the pseudolikelihood estimator is the block-pseudolikelihood estimator. The MBPLE relies on the same idea, but evaluates the full conditional of blocks of variables rather than the full conditionals of single variables.

$$\pi(\boldsymbol{y}|\beta) \approx \pi_{blockpseudo}(\boldsymbol{y}|\beta) = \prod_{b=1}^{B} \pi(\boldsymbol{y}_b|\boldsymbol{y}_{-b},\beta), \tag{3}$$

for some partitioning of the dyads into B disjoint groups of size b, such that $\bigcup_{b=1}^{B} y_b = \boldsymbol{y}$ and $\bigcap_{b=1}^{B} y_b = \varnothing$. Assuming the largest of the blocks contains no more than roughly 20 dyads we calculate each full conditional. This approximation is also quite fast, and attempts to capture larger interactions within the graph. Note that similar block pseudolikelihood approximations have been considered in the context of hidden binary Markov random fields, see Rydén and Titterington (1998) and Friel et al. (2009), where superior performance of the block pseudolikelihood estimators was observed.

3.3 Relaxed dependence approximation (RDA)

The joint distribution of $\boldsymbol{y}$ can be written as

$$p(\boldsymbol{y}|\beta) = p(y_{1:b}|y_{b+1:n},\beta)\prod_{i=b+1}^{n-1} p(y_i|y_{i+1:n},\beta)p(y_n|\beta).$$

The RDA, in essence, attempts to approximate the distribution of $p(y_i|y_{i+1:n},\beta)$. First, notice that

$$\begin{aligned} p(y_i|y_{i+1:n},\beta) &= \frac{p(y_{1:i-1},y_i|y_{i+1:n},\beta)}{p(y_{1:i-1}|y_{i:n},\beta)} \\ &= \frac{p(y_{\bar{A}^i}|y_{i+1:n},\beta)}{p(y_{\bar{A}^i}|y_{i:n},\beta)} \times \\ &\quad \frac{p(y_{A^i},y_i|y_{-\{A^i,i\}},\beta)}{p(y_{A^i}|y_{-\{A^i\}},\beta)} \end{aligned} \tag{4}$$

where $A^i = \{A^i_1,\dots,A^i_{b-1}\} \subset \{1,\dots,i-1\}$, $\bar{A}^i = \{1,\dots,i-1\} \setminus A^i$ and $y_{-\{A\}} = \boldsymbol{y} \setminus y_A$, for $i = b+1,\dots,n-1$. We introduce an approximation to (4) by writing

$$p(y_i|y_{i+1:n},\beta) \approx \frac{p(y_{A^i},y_i|y_{-\{A^i,i\}},\beta)}{p(y_{A^i}|y_{-\{A^i\}},\beta)} \tag{5}$$

Similarly, we approximate

$$p(y_n) \approx \frac{p(y_{A^n},y_n|y_{-\{A^n,n\}},\beta)}{p(y_{A^n}|y_{-\{A^n\}},\beta)}. \tag{6}$$

Here we define a *block* of size b to be the set $\{y_{A^i},y_i\}$, for $i = b+1,\dots,n$. We further denote a block of size 1 to correspond to $A^i = \emptyset$, and in this instance, (5) and (6) reduce to

$$p(y_i|y_{i+1:n},\beta) \approx p(y_i|y_{-\{i\}},\beta)$$

and

$$p(y_n|\beta) \approx p(y_n|y_{-\{n\}},\beta),$$

respectively.

Plugging (5) and (6) into (4) yields the approximation

$$\begin{aligned} p(y_1,\dots,y_n|\beta) \approx & p(y_{1:b}|y_{b+1:n},\beta) \\ & \prod_{i=b+1}^{n-1} \frac{p(y_{A^i},y_i|y_{-\{A^i,i\}},\beta)}{p(y_{A^i}|y_{-\{A^i\}},\beta)} \\ & \times \frac{p(y_{A^n},y_n|y_{-\{A^n,n\}},\beta)}{p(y_{A^n}|y_{-\{A^n\}},\beta)}. \end{aligned} \tag{7}$$

In effect, (5) and (6) assume that

$$\frac{p(y_{A^i},y_i|y_{-\{A^i,i\}},\beta)}{p(y_{A^i}|y_{-\{A^i\}},\beta)} = 1, \quad i = b+1,\dots,n-1 \tag{8}$$

and

$$\frac{p(y_{\bar{A}^n}|\beta)}{p(y_{\bar{A}^n}|y_n,\beta)} = 1, \tag{9}$$

respectively.

Finally, a nice property of our approximation is that it can be seen as a natural expansion of the pseudolikelihood approximation, which corresponds to a block size of 1. Note that an estimator similar to RDA has been explored in the context of binary Markov random fields on the lattice Friel et al. (2009), and has been implemented in a variational Bayes setting in McGrory et al. (2009).

3.3.1 Ordering the dyads and selecting each blocks

The RDA and MBPLE approaches require that the n dyads in $\boldsymbol{y}$ follow some index ordering. Moreover, for the RDA approach, there is a need to choose, for each $i = b+1,\dots,n$, the set $y_{A_i} \subset \{y_1,\dots,y_{i-1}\}$ of dyads in block $\{y_{A_i},y_i\}$. It is unclear to us how to provide guidance for the former requirement. We are able to offer some guidance as to how the set of dyads y_{A_i} is chosen, however. Our intuition is that each block should consist of as many dyads from $\{y_1,\dots,y_{i-1}\}$ which share a common node with y_i, since these are the dyads which most influence y_i. If there are more than $b-1$ dyads in $\{y_1,\dots,y_{i-1}\}$ sharing a node with y_i, then $b-1$ such dyads are chosen uniformly at random. While if there are less than $b-1$ dyads sharing a common node with y_i, each of these are selected, and the remainder chosen uniformly at random from the set of dyads not sharing a common node with y_i.

4 Monte Carlo approaches

An alternative to approximating the likelihood is to try to estimate the true posterior distribution. This is the approach taken by the Monte Carlo maximum likelihood (MC-MLE) algorithm introduced by Geyer and Thompson (1992). This algorithm has been widely used to carry out maximum likelihood estimation for the p^* model. A key identity is the following

$$\begin{aligned}\frac{z(\beta)}{z(\beta_0)} &= \mathbf{E}_{\boldsymbol{y}|\beta_0}\left[\frac{q(\boldsymbol{y}|\beta)}{q(\boldsymbol{y}|\beta_0)}\right] \\ &= \sum_{\boldsymbol{y}} \frac{q(\boldsymbol{y}|\beta)}{q(\boldsymbol{y}|\beta_0)}\frac{q(\boldsymbol{y}|\beta_0)}{z(\beta_0)} \\ &\approx \frac{1}{m}\sum_{i=1}^{m} \exp\left\{(\beta-\beta_0)^t s(\boldsymbol{y}_i)\right\}\end{aligned}$$

where β_0 is fixed set of parameter values, and $\mathbf{E}_{\boldsymbol{y}|\beta_0}$ denotes an expectation taken with respect to the distribution $\pi(\boldsymbol{y}|\beta_0)$. In practice this ratio of normalising constants is approximated using graphs $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m$ sampled via MCMC from the stationary distribution defined by β_0 and importance sampling. This yields the following approximated log likelihood ratio:

$$\begin{aligned}\hat{l}_{\beta_0}(\beta) \approx l(\beta_0) + (\beta-\beta_0)^t s(\boldsymbol{y}) - \\ \log\left[\frac{1}{m}\sum_{i=1}^{m}\exp\left\{(\beta-\beta_0)^t s(\boldsymbol{y}_i)\right\}\right]. \qquad (10)\end{aligned}$$

This is then viewed as a function of β, and its maximum value serves as a Monte Carlo estimate of the MLE.

A crucial aspect of this algorithm is the choice of β_0. Ideally β_0 should be very close to the maximum likelihood estimator of β. In fact $\hat{l}_{\beta_0}(\beta)$ is very sensitive to the choice of β_0. A poorly chosen value of β_0 may lead to an objective function (10) that cannot even be maximised, see Handcock (2003).

In pratice, β_0 is often chosen as the maximiser of (2), although this itself maybe a very biased estimator. Indeed, (10) may also be sensitive to numerical instability, since it effectively computes the ratio of a normalising constant, but it is well understood that the normalising constants can vary by orders of magnitude with $\boldsymbol{\theta}$.

5 Examples

In this section we consider two real dataset to test and compare the different methods introduced in the previous section. For each example we choose to fit the same model as was introduced in section 2. For both datasets our goal is to gather information about the posterior distribution of β, using our likelihood approximations, $\tilde{p}(\boldsymbol{y}|\beta)$,

$$p(\beta|\boldsymbol{y}) \propto p(\boldsymbol{y}|\beta)p(\beta) \approx \tilde{p}(\boldsymbol{y}|\beta)p(\beta). \qquad (11)$$

We do this by proceeding in the following manner. First we locate the mode of the posterior distribution by plugging our approximate likelihood function combined with an uninformative prior into a black-box optimiser. Once the mode is located we design a grid surrounding the mode and evaluate an approximate unnormalised posterior distribution (the right hand side of (11)) at each grid point. This allows us to produce numerical approximations to marginals, means, variances and other aspects of the posterior distribution. We also want to ensure that our approximations have not moved us into parameter space that produces degenerate graphs. So once the posterior is calculated we sample 500 values for β from the posterior. We then use the `ergm` package for `R` (Hunter et al., 2008) to simulate a graph from each of these sets of parameters and study the resulting graphs to check for degeneracy. For both the RDA and blockpseudo the blocksize is critical, the bigger the blocks, the better we expect our approximation to be, but at the price of increased run-time, also, avaliable memory restricts us to blocks of size ≤ 20. As mentioned earlier, a block size of 1 is equivalent to standard pseudolikelihood.

5.1 Molecule example

Our first example examined the dataset illustrated in figure 1.

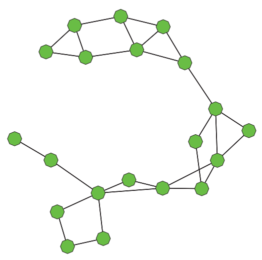


Figure 1: Molecule graph.

The graph consists of 20 nodes in a quite sparse configuration, which gives us a dataset of 190 variables. In figure 2 we have plotted the approximate posterior mode returned from the optimisation algorithm for the RDA and blockpseudo approach, with blocksizes ranging from 1 to 18 for the rda and 1 to 16 for blockpseudo.

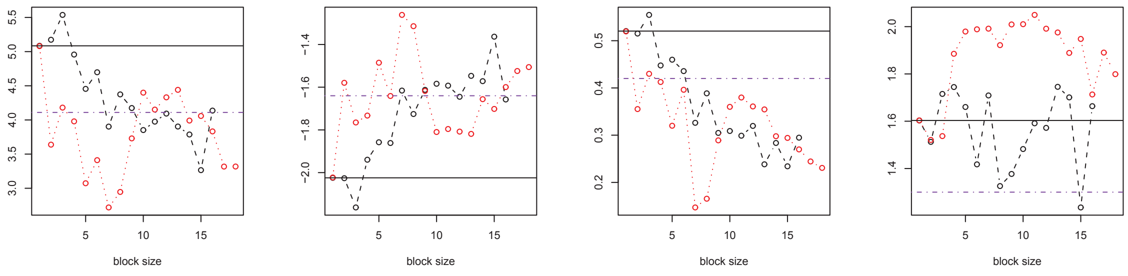


Figure 2: Molecule dataset: Approximate maximum a posteriori parameters estimates for β_1, β_2, β_3 and β_4, plotted from left to right for blocksizes from 1 to 16 (blockpseudo) and 1 to 18 (rda). The black dotted line represents the blockpseudo approach while the red stapled line represents the RDA approximation. The horizontal black line represents the pseudolikelihood approximation, while the horizontal purple stapled line represents an MC-MLE approximation method.

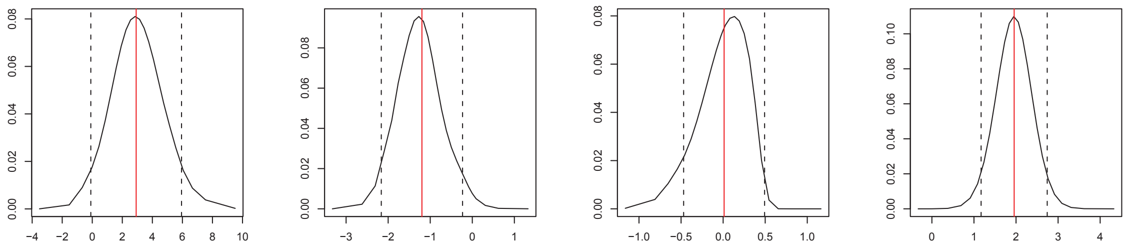


Figure 3: Molecule dataset: Approximate posterior marginals for β_1, β_2, β_3 and β_4 using the RDA approximation. The red dotted line represents the mean and the black dotted lines indicate distances of two posterior standard deviations from the mean.

The vertical line is placed at the value for blocksize 1, which represents the pseudolikelihood approximation. As we can see the RDA and blockpseudo seem to give similar parameter estimates (with the exception of β_4) and each parameter estimate is quite different from the pseudolikelihood estimator. The purple stapled line represents an estimate returned by an MC-MLE method. We note that this comes closer to the approximations returned by the RDA and blockpseudo. The posterior distribution of β was next evaluated in a grid surrounding the mode of the RDA approximation with a blocksize of 10. The grid contained 25 points in each dimension. Figure 3 shows the approximate marginal distributions $p(\beta_i|\boldsymbol{y})$ with means and credible intervals, calculated from the full posterior distribution. Note that parameter estimates for pseudolikelihood and MC-MLE found using the `ergm` package gave rise to realisations that are almost all degenerate. By constrast realisations conditional on parameters from the posterior were not degenerate, being neither complete or empty.

We also estimate the marginal distribution of the data. Recall that in equation (11) the normalizing constant is the approximate marginal likelihood of y. Hence we can estimate the marginal likelihood of our data by completing the finite sum over the parameters,

$$p(\boldsymbol{y}) \approx \tilde{p}(\boldsymbol{y}) = \sum_{\beta_1}\sum_{\beta_2}\sum_{\beta_3}\sum_{\beta_4} \tilde{p}(\boldsymbol{y}|\beta)p(\beta). \qquad (12)$$

Here, recall that the summands on the right hand side of (12) are available and therefore an estimate of the marginal likelihood results from summing these over all grid points, β. The grid computed for the saturated model can then be used to estimate the marginal likelihood for any model nested within the saturated model, by simply setting those parameter values which are not included in the model to zero. This was then done for all 15 different model configurations containing at least one parameter. Assigning equal weights to each of the models the posterior model probabilities can be estimated as,

$$\tilde{p}(m_i|y) = \frac{\tilde{p}(y|m_i)p(m_i)}{\sum_j \tilde{p}(y|m_j)p(m_j)} \quad i = 1, \ldots, 15. \qquad (13)$$

The models with highest posterior model probability turned out to be the saturated model, the model containing the parameters $\{\beta_2, \beta_3, \beta_4\}$ and the model containing the parameters $\{\beta_1, \beta_2, \beta_4\}$, these achieved probabilities 0.64, 0.14 and 0.11 respectively.

5.2 Karate example

For our second example we studied the dataset illustrated in figure 4.

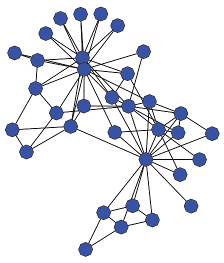


Figure 4: Karate graph.

This graph is larger than the previous example and consists of 34 nodes in a not so sparse configuration, which gives us a dataset of 561 variables. In figure 5 we have plotted the approximate posterior mode returned from the optimisation algorithm for the RDA and blockpseudo approach, with blocksizes ranging from 1 to 16. As in the plot in the previous example the vertical line is placed at the value for blocksize 1, which represents the pseudolikelihood approximation. The RDA and blockpseudo seem to return slightly, but not entirely dissimilar parameter estimates and again each parameter estimate is quite different from the pseudolikelihood estimator. We evaluated the posterior distribution of β as in the previous example with a grid surrounding the mode of the RDA approximation with a blocksize of 10. The grid contained 20 points in each dimension. Figure 6 shows the approximate marginal distributions $p(\beta_i|\boldsymbol{y})$ with means and credible intervals, calculated from the full posterior distribution. As for the previous example, parameter estimates for pseudolikelihood and MC-MLE found using the `ergm` package gave rise to realisations that are almost all degenerate. By constrast realisations conditional on parameters from the posterior based on RDA and block pseudolikelihood were not degenerate, being neither complete or empty. Exactly as in the molecule example we also estimate the marginal distribution of the data. The two models with highest posterior model probability turned out to be the saturated model and the model containing the parameters $\{\beta_1, \beta_2, \beta_3\}$, these achieved probabilities 0.705 and 0.236 respectively.

6 Discussion

Despite the widespread use of the p^* model in social network analysis, the inferential methods used to service this model are lacking in many respects. The approximations which we have outlined in this paper ad-

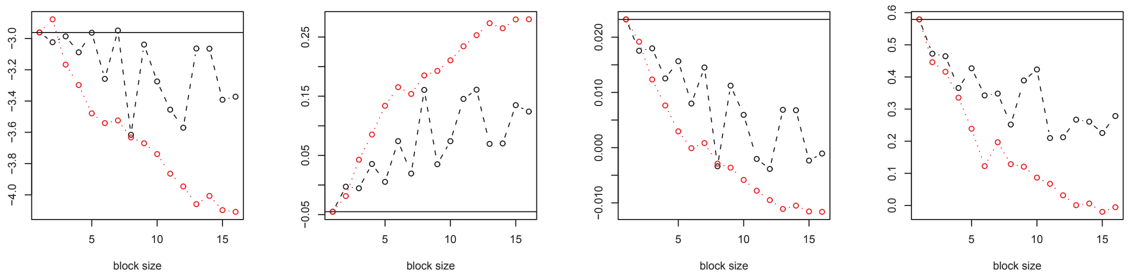


Figure 5: Karate dataset: Approximate maximum a posteriori parameters estimates for β_1, β_2, β_3 and β_4, plotted from left to right for blocksizes from 1 to 16. The black dotted line represents the blockpseudo approach while the red stapled line represents the RDA approximation. The horizontal black line represents the pseudolikelihood approximation, while the horizontal purple stapled line represents an MC-MLE approximation method.

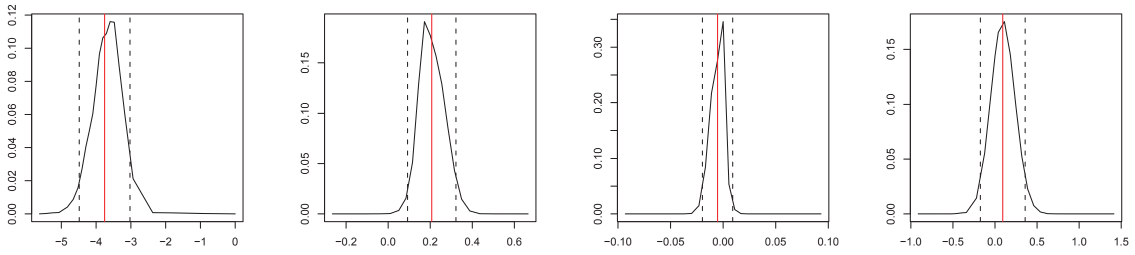


Figure 6: Karate dataset: Approximate posterior marginals for β_1, β_2, β_3 and β_4 using the RDA approximation. The red dotted line represents the mean and the black dotted lines indicate distances of two posterior standard deviations from the mean.

dress this issue. Both of these approximations extend the standard pseudolikelihood approximation. Our approximations can be considered as composite likelihood approximations, where the composite factors in the likelihood involve at most 20 dyad variables. Composite likelihood methods are popular in the statistics literature, for example Heagerty and Lele (1998), Cox and Reid (2004), and the theory surrounding such methods is well established. However composite likelihoods have received relatively little attention in the Bayesian literature, and future work in this directions would be useful.

As mentioned in section 3, it is unclear how to choose an index ordering of the dyads for RDA and blockpseudolikelihood. However for the RDA approach, we believe that including as many as of the dyads which share a common node with y_i, represents a reasonable way to select composite blocks.

We note that our inference methods for the p^* model provide an appealing simulation-free alternative to the usual Markov chain Monte Carlo approaches. In particular our methods can be considered as an inference machine for these types of model providing the end user with the possibility to explore probabilistic uncertainty for the model parameters and also for the uncertainty estimates for the model itself. Finally, we are currently automating our computer code to provide the end user with a suite of routines to carry out the inference tasks outlined in this paper.

Acknowledgement: Haakon Austad would like to acknowledge support from The Research Council of Norway, Statoil and ENI. Nial Friel's research was supported by a Science Foundation Ireland Research Frontiers Program grant, 09/RFP/MTH2199.

References

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

Cox, D. R. and Reid, N. (2004), "A note on pseudolikelihood constructed from marginal densities," *Biometrika*, 91, 729–737.

Erdös, P. and Rényi, A. (1959), "On random graphs," *Publicationes Mathematicae*, 6, 290–297.

Frank, O. and Strauss, D. (1986), "Markov Graphs," *Journal of the American Statistical Association*, 81, 832–842.

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 Statistics*, 18, 243–261.

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.

Handcock, M. S. (2003), "Assessing Degeneracy in Statistical Models of Social Networks," *Working Paper no.39, Center for Statistics and the Social Sciences, University of Washington*.

Heagerty, P. J. and Lele, S. R. (1998), "A composite likelihood approach to binary spatial data," *Journal of the American Statistical Association*, 93, 1099–1111.

Holland, P. W. and Leinhardt, S. (1981), "An exponential family of probability distributions for directed graphs (with discussion)," *Journal of the American Statistical Association*, 76, 33–65.

Hunter, D. R., Handcock, M. S., Butts, C. T., Goodreau, S. M., and Morris, M. (2008), "ergm: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks," *Journal of Statistical Software*, 24, 1–29.

McGrory, C. A., Titterington, D. M., Reeves, R., and Pettitt, A. N. (2009), "Variational Bayes for estimating the parameters of a hidden Potts model." *Statistics and Computing*, 19, 329–340.

Rinaldo, A., Fienberg, S. E., and Zhou, Y. (2009), "On the geometry of descrete exponential random families with application to exponential random graph models," *Electronic Journal of Statistics*, 3, 446–484.

Robins, G., Snijders, T., Wang, P., Handcock, M., and Pattison, P. (2007), "Recent developments in exponential random graph (p^*) models for social networks," *Social Networks*, 29, 192–215.

Rue, H., Martino, S., and Chopin, N. (2009), "Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations (with discussion)," *Journal of the Royal Statistical Society, Series B*, 71, 319–392.

Rydén, T. and Titterington, D. M. (1998), "Computational Bayesian analysis of hidden Markov models," *Journal of Computational and Graphical Statistics*, 7, 194–211.

Strauss, D. and Ikeda, M. (1990), "Pseudolikelihood estimation for social networks," *Journal of the American Statistical Association*, 5, 204–212.

Wasserman, S. and Pattison, P. (1996), "Logit models and logistic regression for social networks: I. An introduction to Markov graphs and p^*," *Psycometrica*, 61, 401–425.

Part III:

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

Haakon Michael Austad and Håkon Tjelmeland

Technical report.

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 (2010) 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 demonstrating 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 $f : \{0, 1\}^n \rightarrow \mathbb{R}$. 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.

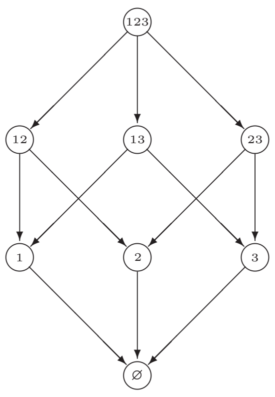


Figure 1: DAG representation of the pseudo-Boolean function in (4). Nodes can be thought of as either representing the set of interactions S or the set of states Ω.

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 $S = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$ and,

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

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, $S^c_\lambda = S \backslash S_\lambda$ and $\Omega^c_\lambda = \Omega \backslash \Omega_\lambda$. Lastly we define $S^0_\lambda = \{\Lambda \in S : \lambda \cap \Lambda = \varnothing\}$ and $\Omega^0_\lambda = \{x \in \Omega : x_k = 0,\ \forall k \in \lambda\}$. If we think of the sets S_λ and Ω_λ as the sets where λ is on, then S^0_λ and Ω^0_λ are the sets where λ is off. Again, using our example we have for instance $S^0_{\{1,2\}} = \{\varnothing, 3\}$ and $\Omega^0_{\{1,2\}} = \{\{0,0,0\},\{0,0,1\}\}$. Note that in general $S^c_\lambda \neq S^0_\lambda$ and equivalently $\Omega^c_\lambda \neq \Omega^0_\lambda$.

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),

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

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 $\tilde{S}$.

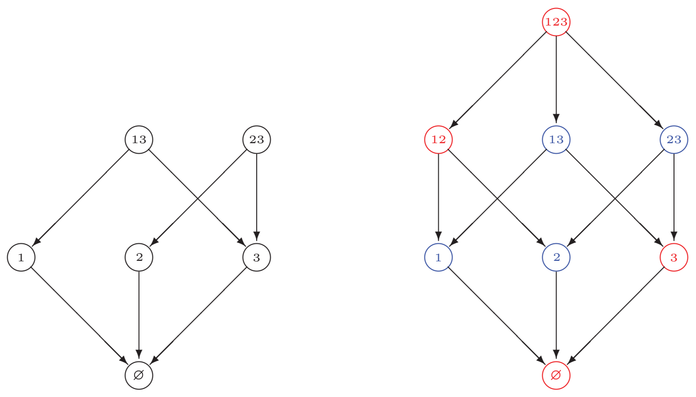


Figure 2: Left: DAG representation of the pseudo-Boolean function defined over the set $\tilde{S} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,3\}, \{2,3\}\}$. Right: Distribution of the error $f(x) - \tilde{f}(x)$ when $S = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\}$ and $\tilde{S} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,3\}, \{2,3\}\}$. 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 $\tilde{S} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,3\}, \{2,3\}\}$, 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 $\text{SSE}(f,\tilde{\tilde{f}})=\text{SSE}(f,\tilde{f})+\text{SSE}(\tilde{f},\tilde{\tilde{f}})$.

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

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

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

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

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 $\mathrm{SSE}(f,\tilde{f})$.

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,

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

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

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

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 $\tilde S = S\setminus\lambda$, i.e we remove only one interaction β^λ. Then,

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

With these theorems in hand we can go from S to $\tilde S$ by removing all nodes in $S\setminus\tilde S$. 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,

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

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

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

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),

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

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 $S = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\}$ and $\tilde{S} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,3\}, \{2,3\}\}$, 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,

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

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,

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

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,

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

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 $f(x) - \tilde{f}(x)$,

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

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

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,

$$\begin{aligned} \sum_{x \in \Omega_\lambda} \Big[f(x) - \tilde{f}(x) \Big] &= \sum_{x \in \Omega_\lambda} \left[\sum_{\Lambda \in S_{\{i,j\}}} \left[\Delta f^{\Lambda}(x_i, x_j) \prod_{k \in \Lambda \backslash \{i,j\}} x_k \right] \right] \\ &= \sum_{\Lambda \in S_{\{i,j\}}} \left[\sum_{x \in \Omega_\lambda} \left[\Delta f^{\Lambda}(x_i, x_j) \prod_{k \in \Lambda \backslash \{i,j\}} x_k \right] \right] \\ &= \sum_{\Lambda \in S_{\{i,j\}}} \left[\sum_{x \in \Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}} \Delta f^{\Lambda}(x_i, x_j) \right] = 0, \ \forall \ \lambda \in \tilde{S}. \end{aligned} \tag{26}$$

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,

$$\begin{aligned} \sum_{x \in \Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}} \Delta f^{\Lambda}(x_i, x_j) = &\frac{|\Omega_{\lambda \cup (\Lambda \backslash \{i,j\})}|}{4} (\Delta f^{\Lambda}(x_i = 0, x_j = 0) + \Delta f^{\Lambda}(x_i = 1, x_j = 0) \\ &+ \Delta f^{\Lambda}(x_i = 0, x_j = 1) + \Delta f^{\Lambda}(x_i = 1, x_j = 1)). \end{aligned} \tag{27}$$

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,

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

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,

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

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

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

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,

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

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

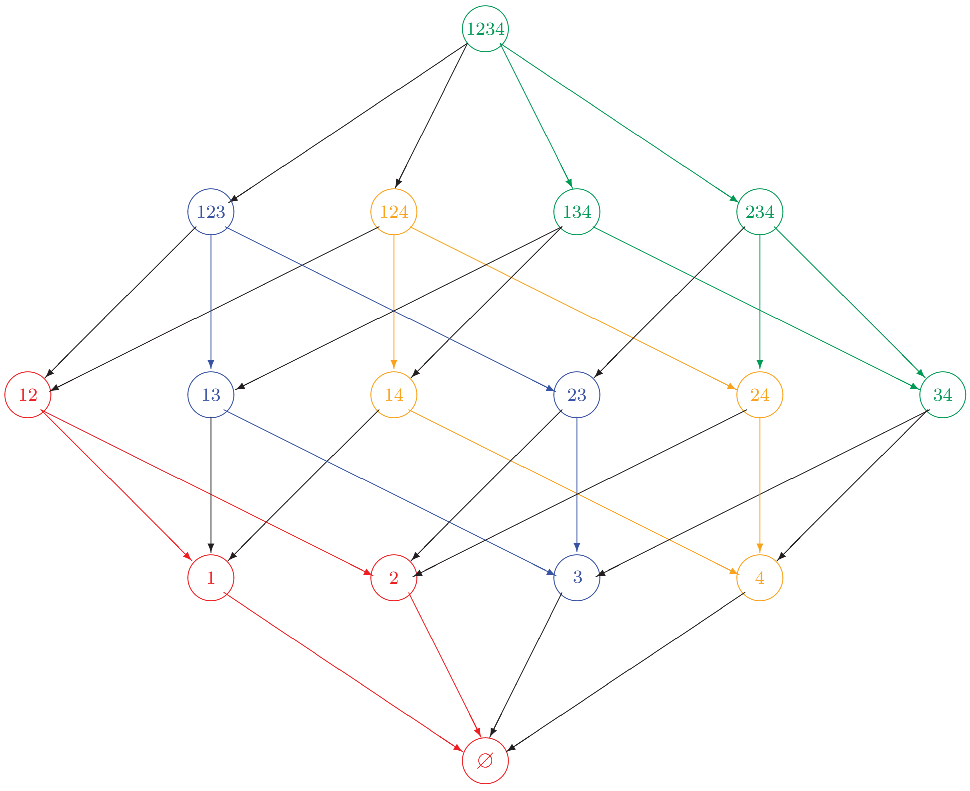


Figure 3: Graph representation of Ω when the dimension of the pseudo-Boolean function is $n = 4$. Colors illustrate the distribution of the absolute value of the error $|f(x) - \tilde{f}(x)|$, when $\tilde{S} = S \backslash S_{\{1,2\}}$. States for which the nodes have the same color have the same absolute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

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

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 (2010). 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 $x = \{x_1, \dots, x_n\} \in \Omega = \{0,1\}^n$, $N = \{1, \dots, n\}$. Let $\mathcal{N} = \{\mathcal{N}_1, \dots, \mathcal{N}_n\}$ 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 (2010). 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 $x_{-i} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$, 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 (2010). 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 $x_{1:i-1} = (x_1, \dots, x_{i-1})$, have an MRF with a neighborhood system $\check{\mathcal{N}} = \{\check{\mathcal{N}}_i, \dots, \check{\mathcal{N}}_n\}$ 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 $p(x_{\text{max}})$, with the property that $p(x_{\text{max}}) \geqslant p(x)$ for all $x \in \Omega$ and $p(x_{\text{max}}) = p(x)$ 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,

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

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

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

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 $p(x_{\text{max}})$. 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 $p(x_{\text{max}})$. 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 (2010).

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 (2010) 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 (2010) 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 (2010).

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 (2010), 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 (2010) and

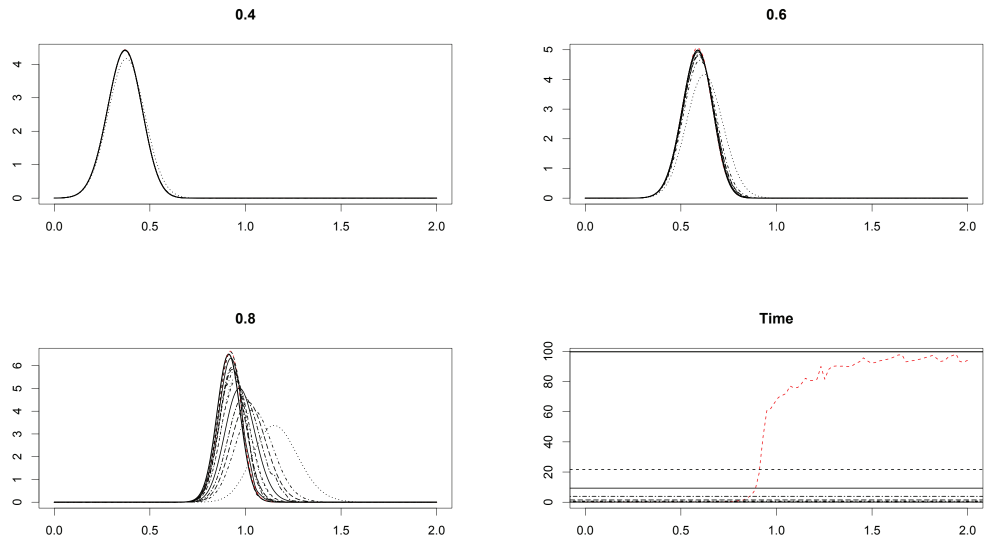


Figure 4: Exact posterior distributions $p(\theta|x)$ (thick continuous black curve), approximations with $\nu = 2$ up to $\nu = 13$ (dashed curves moving from right to left for increasing value of ν) and the approximation of Tjelmeland and Austad (2010)(dashed red curve). Results for $\theta_{\text{true}} = 0.4$ (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 (2010)(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.

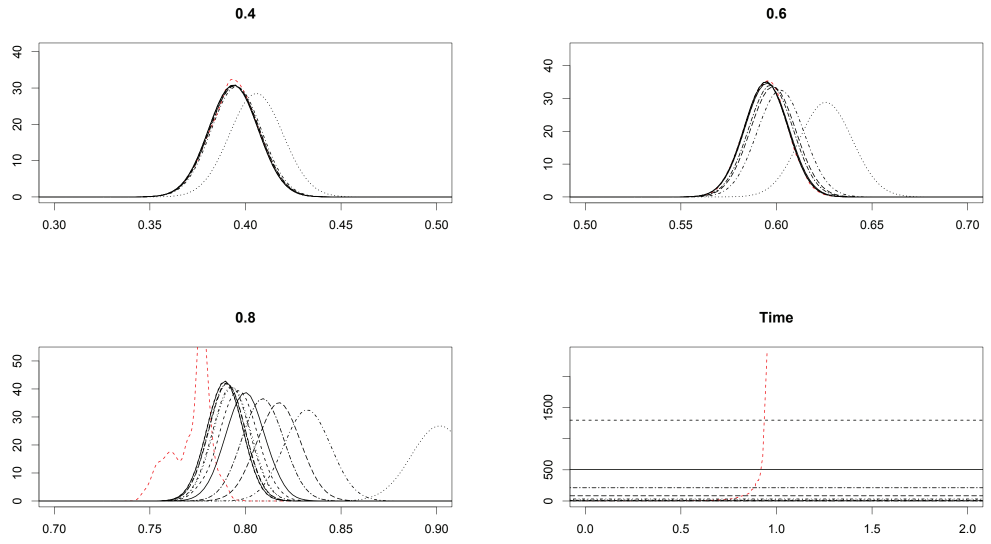


Figure 5: Approximations with $\nu = 2$ up to $\nu = 13$ (dashed curves moving from right to left for increasing value of ν) and the approximation of Tjelmeland and Austad (2010)(dashed red curve). Results for $\theta_{\text{true}} = 0.4$ (upper left), $\theta_{\text{true}} = 0.6$ (upper right) and $\theta_{\text{true}} = 0.8$ (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 (2010)(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 (2010). As we can see from the plot, once θ becomes large enough, the run time for the approximation in Tjelmeland and Austad (2010) 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 (2010) for the case of $\theta_{\text{true}} = 0.8$ on the 100×100 lattice. Here the approximation in Tjelmeland and Austad (2010) 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 (2010) $\tilde{c}_\epsilon(\theta)$, 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 (2010) converges to $0.6931 = \log(2)$ as θ increases. This happens

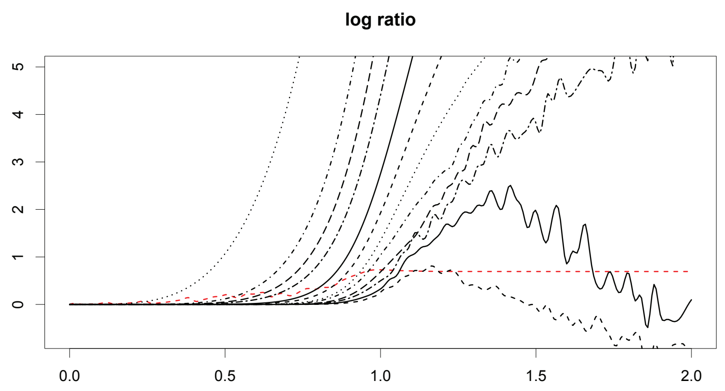


Figure 6: $\log(c(\theta)) - \log(\tilde{c}_\nu(\theta))$ for $\nu = 2$ up to $\nu = 13$) (black dotted curves moving from left to right for increasing value of ν) as well as $\log(c(\theta)) - \log(\tilde{c}_\epsilon(\theta))$ for $\epsilon = 0.001$ (dotted red curve). Results are shown for values of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

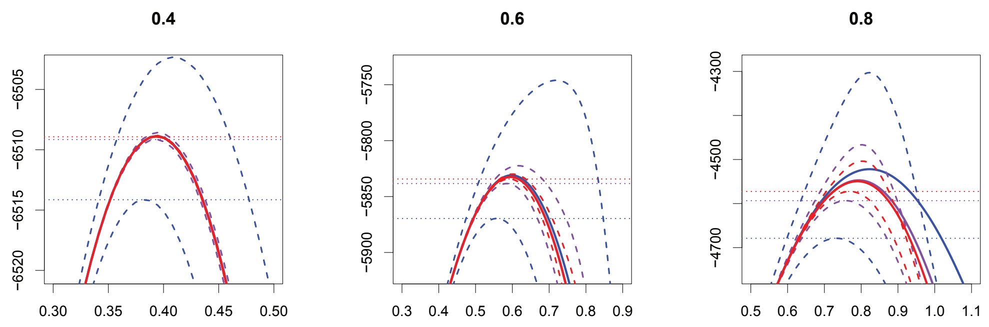


Figure 7: Approximate log-likelihood functions (continuous curves) for $\nu = 6$ (blue), $\nu = 10$ (purple) and $\nu = 13$ (red), for $\theta_{\text{true}} = 0.4$ (left), $\theta_{\text{true}} = 0.6$ (middle) and $\theta_{\text{true}} = 0.8$ (right) for the Ising model on a 100×100 lattice. Upper and 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 $k^* = \tilde{r}_U(\tilde{x}_{max}) \geqslant r(x_{max})$.

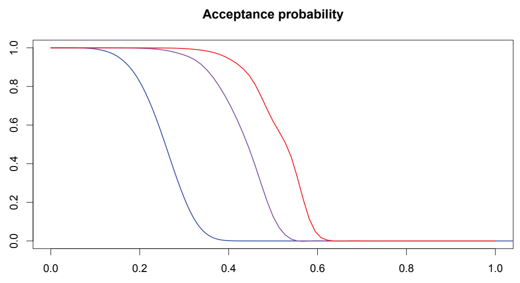


Figure 8: Estimated acceptance probabilities as a function of θ for the rejection sampling algorithm when sampling from an Ising model on a 100×100 lattice. Three values for ν were tested, $\nu = 6$ (blue), $\nu = 10$ (purple) and $\nu = 13$ (red).

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, $r(x) = \frac{1}{c} exp(U(x) - \tilde{U}(x))$. 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 $y = \{y_1, \dots, y_n\}$ 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

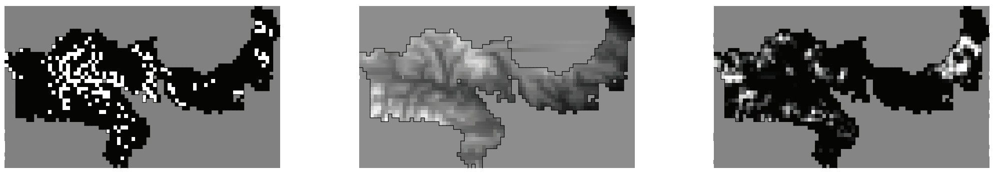


Figure 9: Red deer data set. Left plot shows presence/absence of reed deer, white indicating presence ($y_i = 1$) and black representing absence ($y_i = 0$). Middle plot shows altitude covariate, white indicating high altitude and black indicating low altitude. Right plot shows mires covariate, again 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 (2010), 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,

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

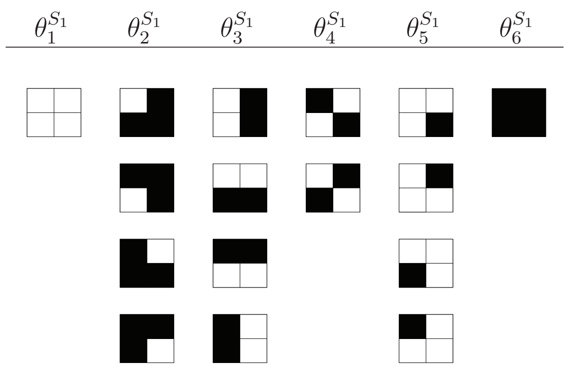


Figure 10: Classes of clique configurations for a translation and rotation invariant 2 × 2 clique. Each column includes the configurations of the clique represented by the associated parameter. Each gridcell is either 1 (white) or 0 (black).

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

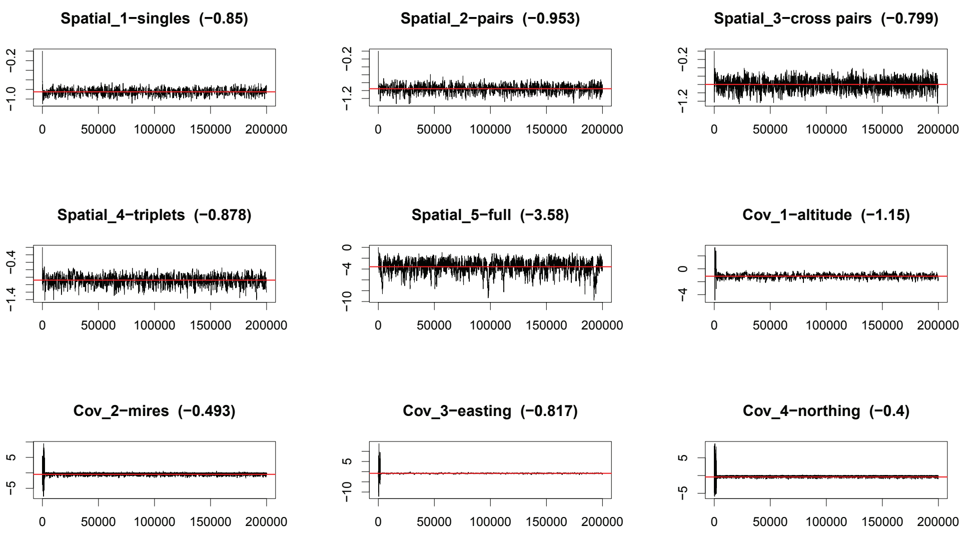


Figure 11: Trace plot for a RJMCMC run using spatial model S_1, from top to bottom, left to right, the first five plots show the spatial parameters $(\theta_1^{S_1}, \theta_2^{S_1}, \theta_3^{S_1}, \theta_4^{S_1}, \theta_5^{S_1})$ followed by covariate parameters $(\theta_1^C, \theta_2^C, \theta_3^C, \theta_4^C)$ 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.

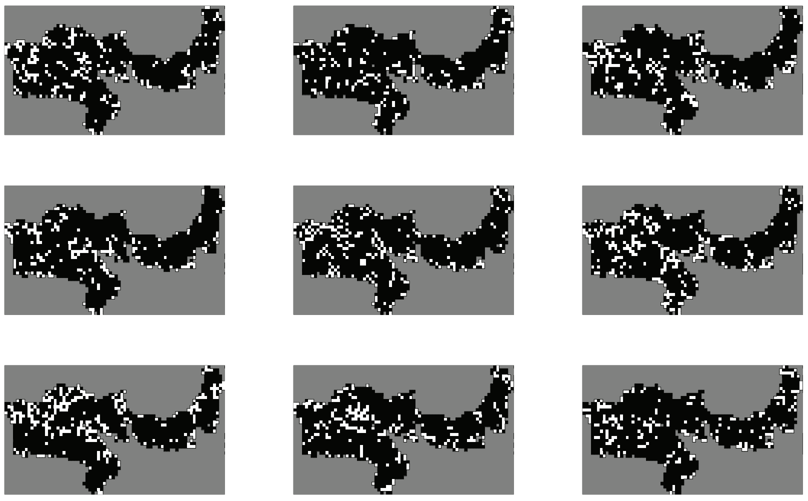


Figure 12: 9 realizations from $p_2(y|\theta^{S_2}, \theta^C, z)$ for values of θ^{S_2} and θ^C sampled from the RJMCMC run (discounting a burn-in of 50000 iterations).

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

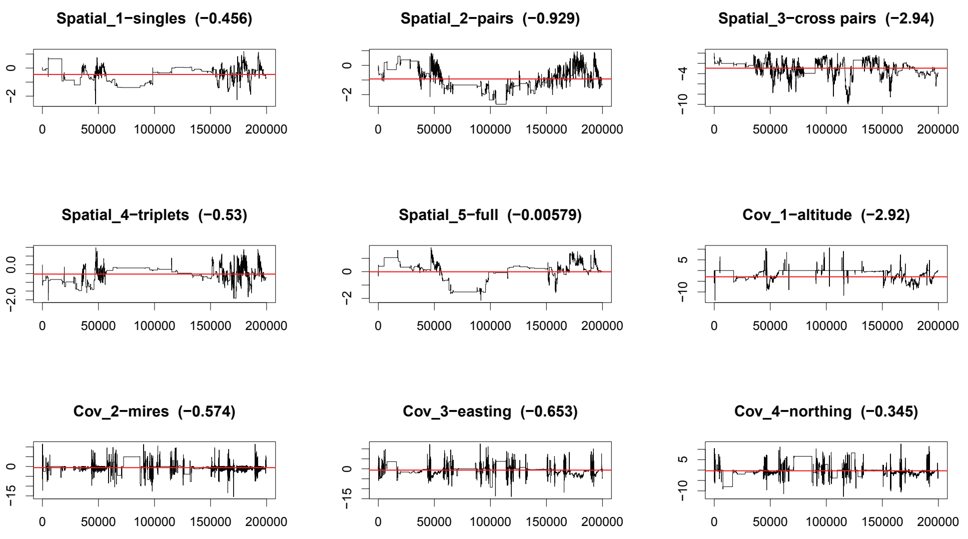


Figure 13: Trace plot for a RJMCMC run using spatial model S_1 and including the observation probability $\mathcal{P}$, from top to bottom, left to right, the first five plots show the spatial parameters $(\theta_1^{S_1}, \theta_2^{S_1}, \theta_3^{S_1}, \theta_4^{S_1}, \theta_5^{S_1})$ followed by covariate 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.

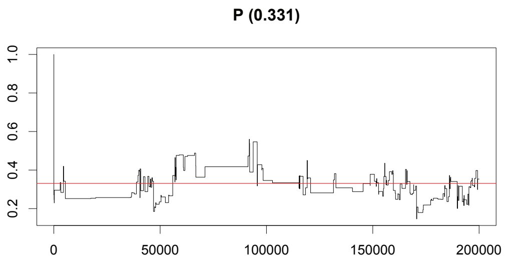


Figure 14: Trace plot for $\mathcal{P}$ for RJMCMC run using spatial model S_1 with probability of observing deer $\mathcal{P}$.

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 $\times$ 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

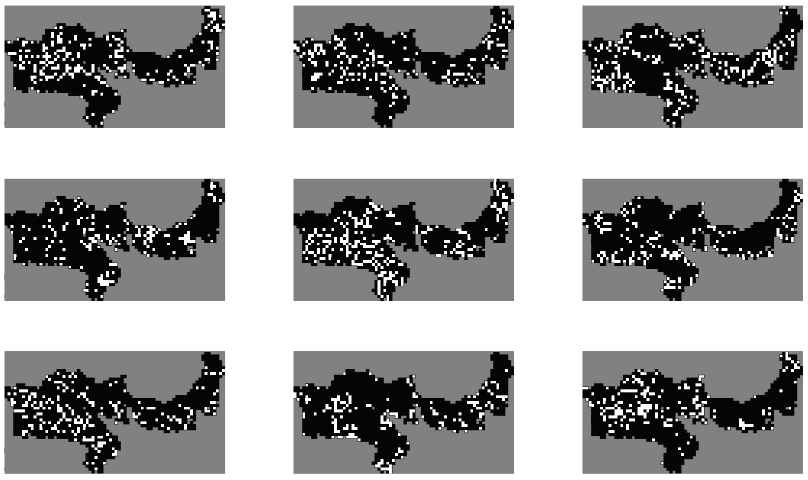


Figure 15: Nine realizations from $p(y|x, \mathcal{P})$ where x is simulated from $p(x|\theta^S, \theta^C, z)$ for values of θ^S, θ^C and $\mathcal{P}$ sampled from the RJMCMC run (discarding a burn-in of 50000 iterations).

closing remarks.

The approximation we have defined was inspired by the work in Tjelmeland and Austad (2010) 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,

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

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 $\breve{\mathcal{N}}_i$ 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. (2010). Exact and approximate recursive calculations for binary Markov random fields defined on graphs, *Technical report*, Department of Mathematical Sciences, Norwegian University of Science and Technology.

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. $\square$

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,

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

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

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

or,

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

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

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

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 $f_L^*(x) = f_L(x)$.