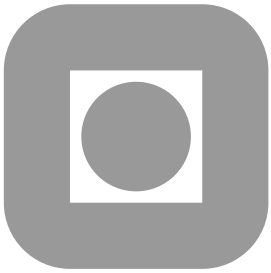
NORGES TEKNISK-NATURVITENSKAPELIGE
UNIVERSITET

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

by

Håkon Tjelmeland and Haakon Michael Austad

PREPRINT
STATISTICS NO. 2/2010



NORWEGIAN UNIVERSITY OF SCIENCE AND
TECHNOLOGY
TRONDHEIM, NORWAY

This report has URL http://www.math.ntnu.no/preprint/statistics/2010/S2-2010.pdf
Håkon Tjelmeland has homepage: http://www.math.ntnu.no/∼haakont
E-mail: haakont@stat.ntnu.no
Address: Department of Mathematical Sciences, Norwegian University of Science and Technology,
N-7491 Trondheim, Norway.

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
Trondheim, Norway

Abstract

In this paper we propose computationally feasible approximations to binary Markov random fields. The basis of the approximation is the forward-backward algorithm. This exact 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, this gives an algorithm that is computationally feasible both in terms of computation time and memory requirements. The resulting approximate forward 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.

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). A frequentist 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 Markov random fields. Alternatively, the x is a latent unobserved variable and in stead a vector y is observed from an 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. Let $p(y|x,\theta)$ denote an assumed distribution for an observed vector y given an unobserved latent vector x and a parameter vector θ. A Markov random field $p(x|\theta)$ prior is adopted for x and some prior $p(\theta)$ is chosen for θ. The focus of interest is then the posterior distribution for θ, $p(\theta|y)$, and the posterior distribution for x, $p(x|y)$. The main issue in these kind of problems is that the normalizing constant of the Markov random field $p(x|\theta)$ is computationally intractable. Thereby numerical optimization is infeasible for the frequentist problems and standard Markov chain Monte Carlo algorithms are not a viable alternative for the Bayesian problem. Similar problems also occur for other classes of models for which the normalizing constant is not analytically available, see for example the discussions in Møller et al. (2006) and Murray (2007).

The strategies for dealing with an intractable normalizing constant proposed in the literature can be categorized into three groups. The first is to replace the intractable constant

with an approximation that is easily computable. For MRFs Besag (1974) defines a pseudo-likelihood function as a product of full conditionals and proposes to estimate θ when x is observed by maximizing this instead of the intractable likelihood function. In Besag et al. (1991) and Rydén and Titterington (1998) the same pseudo-likelihood function is used as an approximation to the 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 an approximation based on exact calculations for smaller lattices by the so called forward-backward algorithm (Künsch, 2001; Scott, 2002; Bartolucci and Besag, 2002; Pettitt et al., 2003). These exact calculations on smaller lattices are thereafter glued together to give approximative results for larger lattices. One should note that the methods proposed in these two articles are only feasible for MRFs defined on very small neighborhoods, as otherwise exact computations are not feasible even for small lattices. The second approach is to estimate the intractable normalizing constant using Markov chain Monte Carlo (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 values of θ close to θ_0, and perform numerical optimization on the resulting estimated likelihood function. In Tjelmeland and Besag (1998) this strategy is adopted to find the maximum likelihood estimators for a discrete Markov random field. Gelman and Meng (1998) run independent MCMC chains for a series of θ values, use this to estimate the normalizing constant as a function of θ, and then run MCMC for $p(x, \theta|y)$ with the unknown normalizing constant replaced by the corresponding estimate. A recent third strategy to cope with intractable normalizing constants in the Bayesian formulation of the problem is to utilize that exact samples from $p(x|\theta)$ can be generated. In Møller et al. (2006) an auxiliary variable Metropolis–Hastings algorithm is defined, where the proposal distribution is constructed so that the intractable normalizing constant cancels from the Metropolis–Hastings ratio. However, the price to pay for not having to compute the intractable normalizing constant is that exact sampling from $p(x|\theta)$ is required. Moreover, for the algorithm to be efficient an approximation to $p(x|\theta)$ without an intractable normalizing constant must be available. See also Murray (2007) for other constructions to bypass an intractable normalizing constant by using perfect sampling.

In the present paper we define a deterministic approximation to the distribution $p(x|\theta)$ where the normalizing constant is easily computable. Thus, our solution can be categorized into the first class defined above. However, our approximation of $p(x|\theta)$ can also be used in the construction of Møller et al. (2006). We primarily focus on binary fields, but also discuss how our method can be generalized to more than two classes. As in Friel and Rue (2007) and Friel et al. (2009), our starting point is the exact (but for large lattices, computationally infeasible) forward-backward algorithm. The forward part of the forward-backward 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 and develop recursive formulas for these interaction parameters. The approximation is defined by approximating to zero all interaction parameters that are sufficiently close to zero. In addition, an interaction parameter is also approximated to zero whenever all associated lower level interactions are (approximated to) zero. If sufficiently many interactions are set to zero, this makes the algorithm feasible both in terms of computation time and memory requirements. It should be noted that our approach does not require exact computations even on small lattices, which allows the method to be used also for MRFs with somewhat larger neighborhoods.

The paper is organized as follows. In Section 2 we introduce necessary notation and define binary Markov random fields for a general graph. The exact forward-backward algorithm

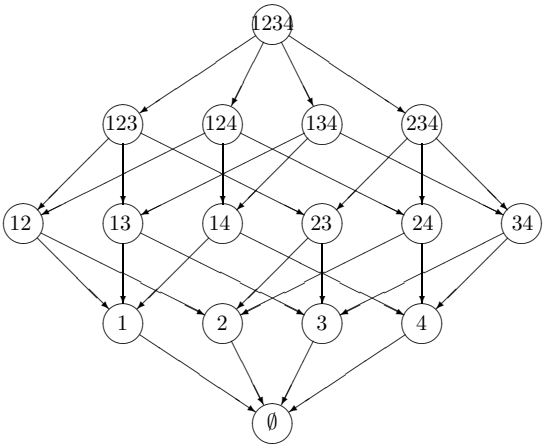


Figure 1: The graph $\mathcal{G}(\mathcal{P}(S))$ for $S = \{1, 2, 3, 4\}$.

for Markov random fields is specified in Section 3, and then in Section 4 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 discuss how to evaluate the quality of the approximations, and in Section 7 we present results for a number of example Markov random fields. Finally, in Section 8 we provide conclusions.

2 Binary Markov random fields

For a general introduction to Markov random fields, see, for example, Besag (1974), Kindermann and Snell (1980) or Cressie (1993). We give a description of binary Markov random fields which facilitates the development of exact and approximate recursive algorithms for such models. We start by discussing how general (i.e. not necessarily Markov) binary random fields can be characterized by a set of interaction parameters.

2.1 Binary random fields

We consider a set of n nodes, $S = \{1, \ldots, n\}$. To each node $k \in S$ we associate a binary variable $x_k \in \{0, 1\}$ and let $x = (x_1, \ldots, x_n) \in \Omega = \{0, 1\}^n$. We also use the 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 $k \in S$. Then we get 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 $\Lambda \in \mathcal{P}(S)$ we have the vector $x = (x_1, \ldots, x_n)$ where $x_k = 1$ for $k \in \Lambda$ and $x_k = 0$ otherwise. We denote the latter relation by $\chi(\Lambda)$, i.e.

$$\chi(\Lambda) = (I(1 \in \Lambda), \ldots, I(n \in \Lambda)), \tag{1}$$

where $I(\cdot)$ is the indicator function. As shown in Figure 1 for $n = 4$, associated with the

power set $\mathcal{P}(S)$ we have a directed acyclic graph $\mathcal{G}(\mathcal{P}(S))$ with one vertex for each subset of S and where the descendents of a vertex $\Lambda \subseteq S$ are all (proper) subsets of Λ.

Assume we are given a probability distribution for x denoted by $p(x), x \in \Omega$. We assume that $p(x)$ fulfil the so called positivity condition, i.e. $p(x) > 0$ for all $x \in \Omega$, so we can write

$$p(x) = c \exp\{-U(x)\}, \tag{2}$$

where c is a normalizing constant and $U(x)$ an energy function. In the following we represent the energy function in terms of a set of interaction parameters, $\beta = \{\beta(\Lambda), \Lambda \subseteq S\}$, defined by the identity

$$U(x) = \sum_{\Lambda \subseteq S} \beta(\Lambda) \prod_{k \in \Lambda} x_k. \tag{3}$$

As there is one interaction parameter for each vertex in the graph $\mathcal{G}(\mathcal{P}(S))$ it is natural to store the elements of β in their respective vertices. Thereby the distribution $p(x)$ can in principle be represented by this vertex-weighted graph, which we denote by $\mathcal{G}(\mathcal{P}(S), \beta)$. However, as the number of elements in $\mathcal{P}(S)$ grows exponentially with n this representation is in practice only feasible for small values of n. For larger values of n we need a more compact representation. We discuss below how this can be done for Markov random fields. First, however, we consider how one can compute the interaction parameters from a given energy function $U(x)$ in an efficient way.

Inserting $x = \chi(\Lambda)$ in (3) and solving with respect to $\beta(\Lambda)$ we get

$$\beta(\Lambda) = U(\chi(\Lambda)) - \sum_{A \subset \Lambda} \beta(A), \tag{4}$$

where the sum equals zero if $\Lambda = \emptyset$. Thereby the interaction parameters can be computed recursively, first computing $\beta(\emptyset)$, then $\beta(\{k\})$ for all $k \in S$, then $\beta(\Lambda)$ for all $|\Lambda| = 2$ and so on. In Figure 1 this corresponds to visiting the vertices of $\mathcal{G}(\mathcal{P}(S))$ sequentially from the bottom to the top. Note however that a direct implementation of (4) is computationally inefficient as it implies repeated calculations of the the same sum of interaction parameters. To see how to avoid this, define $\gamma_0(\Lambda) = \beta(\Lambda)$ for all $\Lambda \in \mathcal{P}(S)$ and

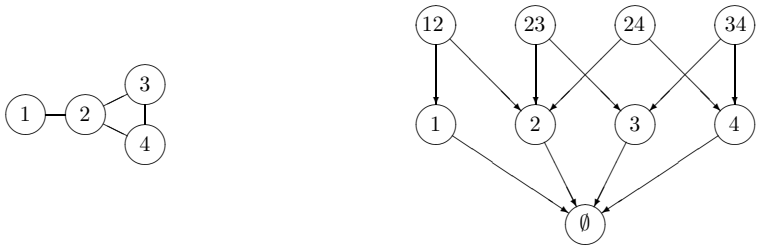
$$\gamma_g(\Lambda) = \frac{1}{g} \sum_{k \in \Lambda} \gamma_{g-1}(\Lambda \setminus \{k\}) \quad \text{for } g = 1, \ldots, |\Lambda|, \Lambda \in \mathcal{P}(S) \setminus \{\emptyset\}. \tag{5}$$

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

$$\beta(\Lambda) = U(\chi(\Lambda)) - \sum_{g=1}^{|\Lambda|} \gamma_g(\Lambda). \tag{6}$$

Thus, to calculate the interaction parameters one should visit all the vertices in $\mathcal{G}(\mathcal{P}(S))$ sequentially in the order discussed above. When visiting a vertex Λ one should first compute and store $\gamma_g(\Lambda)$ for $g = 1, \ldots, |\Lambda|$ using (5) and thereafter compute $\beta(\Lambda)$ by (6).

Reconsider now the vertex-weighted graph representation of $p(x)$ defined above. In the next section we show that for Markov random fields a large number of the interaction parameters are zero. In particular, $\beta(\Lambda) = 0$ for all sets Λ that are not cliques. In our graph representation of $p(x)$ it is clearly not necessary to include vertices that correspond to interaction parameters



(a) (b)

Figure 2: (a) An undirected graph $V = (S, E)$ visualizing the neighborhood system for a toy example Markov random field. (b) The corresponding unweighted graph $\mathcal{G}(\mathcal{B})$ if $\beta(\Lambda) \neq 0$ for $\Lambda \in \{\{1,2\},\{2,3\},\{2,4\},\{3,4\}\}$ and $\beta(\{2,3,4\}) = 0$.

that are equal to zero. Correspondingly, terms with $\beta(\Lambda) = 0$ do not need to be included in the sum in (3). In the following we therefore replace (3) by what will be our canonical representation for energy functions,

$$U(x) = \sum_{\Lambda \in \mathcal{B}} \beta(\Lambda) \prod_{k \in \Lambda} x_k, \quad \text{where} \quad \mathcal{B} = \bigcup_{\Lambda \in \mathcal{P}(S): \beta(\Lambda) \neq 0} \mathcal{P}(\Lambda) \tag{7}$$

and represent $p(x)$ by the corresponding vertex-weighted graph $G = \mathcal{G}(\mathcal{B}, \beta[\mathcal{B}])$, where $\beta[\mathcal{B}] = \{\beta(\Lambda) | \Lambda \in \mathcal{B}\}$. By definition the set $\mathcal{B} \subseteq \mathcal{P}(S)$ contains all $\Lambda \subseteq S$ with $\beta(\Lambda) \neq 0$, but clearly it may also contain some Λ with $\beta(\Lambda) = 0$. We include these 'zero vertices' in our representation of $p(x)$ because having $\mathcal{B}$ as a union of power sets simplifies our recursive algorithms in Sections 3 and 4. For the moment we just note that this choice imply that the elements of $\beta[\mathcal{B}]$ can still be computed recursively as discussed above. Figure 2(b) shows the unweighted graph $\mathcal{G}(\mathcal{B})$ for a $n = 4$ toy example when $\beta(\Lambda) \neq 0$ for $\Lambda \in \{\{1,2\},\{2,3\},\{2,4\},\{3,4\}\}$ and $\beta(\Lambda) = 0$ for $\Lambda \in \{\{1,3\},\{1,4\},\{1,2,3\},\{1,2,4\},\{2,3,4\},\{1,2,3,4\}\}$. Note that in this case $\mathcal{B}$, and thereby the graph $\mathcal{G}(\mathcal{B})$, does not depend on the value of $\beta(\Lambda)$ for $\Lambda \in \{\emptyset, \{1\}, \{2,\}, \{3\}, \{4\}\}$.

2.2 Markov assumption

Let $p(x)$ and $U(x)$ be defined as above, but now consider the situation when x is Markov with respect to a given neighborhood system that we denote by N.

Definition 1. A collection $N = \{N_1, \ldots, N_n\}$ is a neighborhood system for the set S, if $k \notin N_k$ for all $k \in S$ and $k \in N_l \Leftrightarrow l \in N_k$ for all distinct pairs of nodes $k, l \in S$

If $k \in N_l$ we say that k and l are neighbors. Following common practice we visualize a neighborhood system N by an undirected graph $V = (S, E)$ which has one vertex corresponding to each node in S and an edge between any pairs of nodes that are neighbors, i.e. $E = \{(k,l) | k \in N_l, k, l \in S\}$. Such a graph for a toy $n = 4$ example is shown in Figure 2(a).

Definition 2. A binary random field x is said to be Markov with respect to a neighborhood system N if for all $k \in S$ the full conditional $p(x_k | x_{-k})$ fulfils the following Markov property,

$$p(x_k | x_{-k}) = p(x_k | x_{N_k}). \tag{8}$$

We then also say that x is a Markov random field with respect to N.

Definition 3. A set $\Lambda \subseteq S$ is said to be a clique if $k \in N_l$ for all distinct pairs of nodes $k, l \in \Lambda$. We let $\mathcal{C}$ denote the set of all cliques.

For the example graph in Figure 2(a) we have $\mathcal{C} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{2,3\}, \{2,4\}, \{3,4\}, \{2,3,4\}\}$. The following theorem gives that all interaction parameters that do not correspond to a clique must be zero.

Theorem 1. Let x be a Markov random field with respect to a neighborhood system N, let $\mathcal{C}$ be the corresponding set of all cliques, and let $\{\beta(\Lambda), \Lambda \subseteq S\}$ be given by (3). Then $\beta(\Lambda) = 0$ for all $\Lambda \notin \mathcal{C}$.

This theorem is a direct consequence of the Hammersley–Clifford theorem (Besag, 1974; Clifford, 1990) and the fact that the functions $f_\Lambda(x) = \prod_{i \in \Lambda} x_i$, $\Lambda \in \mathcal{P}(S)$ are linearly independent. From the above theorem it follows that if x is known to be Markov with respect to a neighborhood system N we have that $\mathcal{B} \subseteq \mathcal{C}$. To find our graph representation $G = \mathcal{G}(\mathcal{B}, \beta[\mathcal{B}])$ of $p(x)$ we thereby only need to compute $\beta(\Lambda)$ for $\Lambda \in \mathcal{C}$.

We end this section by noting that if we have a vertex-weighted graph $G = \mathcal{G}(\mathcal{B}, \beta[\mathcal{B}])$ representing a distribution $p(x)$, it follows directly that $p(x)$ is Markov with respect to the neighborhood system $N = \{N_1, \ldots, N_n\}$ where $N_k = \{l \in S \setminus \{k\} | \{k, l\} \in \mathcal{B}\}$. Moreover, this neighborhood system is minimal for $p(x)$ in the sense that x is Markov with respect to a neighborhood system $N = \{N_1, \ldots, N_n\}$ if and only if $N_k \supseteq \{l \in S \setminus \{k\} | \{k, l\} \in \mathcal{B}\}$ for all $k \in S$.

3 Exact recursive computations

Let $p(x)$ be a binary Markov random field with respect to a given neighborhood system N. Assume we are representing $p(x)$ by the vertex-weighted graph $G = \mathcal{G}(\mathcal{B}, \beta[\mathcal{B}])$. If $n > 1$ one can clearly, for any node $r \in S$, decompose $p(x)$ into

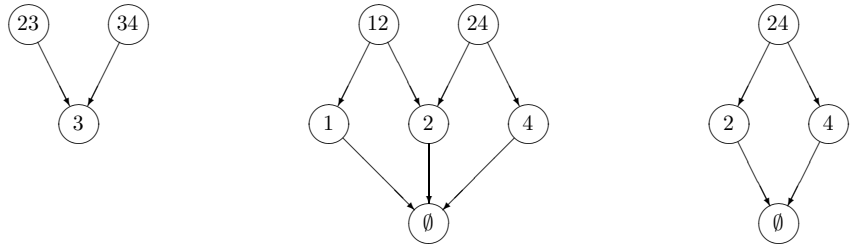
$$p(x_r | x_{-r}) \quad \text{and} \quad p(x_{-r}). \tag{9}$$

From (7) the full conditional $p(x_r|x_{-r})$ is, up to proportionality, given by

$$p(x_r | x_{-r}) \propto \exp\left\{ - \sum_{\Lambda \in \mathcal{B}_r} \beta(\Lambda) \prod_{k \in \Lambda} x_k \right\}, \quad \text{where} \quad \mathcal{B}_r = \{\Lambda \in \mathcal{B} | r \in \Lambda\}. \tag{10}$$

As $x_r \in \{0, 1\}$ the normalizing constant for this full conditional is readily available. Thereby the conditional distribution $p(x_r|x_{-r})$ can be represented by $\mathcal{B}_r$ and $\beta[\mathcal{B}_r] = \{\beta(\Lambda), \Lambda \in \mathcal{B}_r\}$. Again these values can be organized into a vertex-weighted directed acyclic graph, which we denote by $G_r = \mathcal{G}(\mathcal{B}_r, \beta[\mathcal{B}_r])$. For the toy $\mathcal{G}(\mathcal{B})$ in Figure 2(b), the corresponding unweighted graph $\mathcal{G}(\mathcal{B}_3)$ is shown in Figure 3(a). We see that to obtain G_r one just have to cut loose from G the subgraph containing all vertices associated to sets Λ that contain r. We let G_{-r} denote what is left of G after G_r has been removed. Thus, $G_{-r} = \mathcal{G}(\mathcal{B}_{-r}, \beta[\mathcal{B}_{-r}])$, where $\mathcal{B}_{-r} = \mathcal{B} \setminus \mathcal{B}_r = \{\Lambda \in \Lambda | r \notin \Lambda\}$ and $\beta[\mathcal{B}_{-r}] = \{\beta(\Lambda), \Lambda \in \mathcal{B}_{-r}\}$. For the toy $\mathcal{G}(\mathcal{B})$ in Figure 2(b), the unweighted graph $\mathcal{G}(\mathcal{B}_{-3})$ is shown in Figure 3(b).

The distribution $p(x_{-r})$ is a binary random field and as such all the concepts discussed in Sections 2.1 and 2.2 exist also for this distribution. We use $\star$ to mark that a quantity is



(a) $\mathcal{G}(\mathcal{B}_3)$ (b) $\mathcal{G}(\mathcal{B}_{-3})$ (c) $\mathcal{G}(\mathcal{P}(N_3))$

Figure 3: Unweighted graphs derived from the graph $\mathcal{G}(\mathcal{B})$ shown in Figure 2(b).

related to $p(x_{-r})$, so that $U_\star(x_{-r})$ and $G_\star = \mathcal{G}(\mathcal{B}_\star, \beta_\star)$ are the energy function and the vertex-weighted graph representation of $p(x_{-r})$, respectively. Thus, if we from G can determine $G_\star$ we can now represent $p(x)$ by G_r and $G_\star$. This procedure can clearly by iterated, so in the second step of the algorithm we decompose $p(x_{-r})$ into $p(x_s|x_{-\{r,s\}})$ and $p(x_{-\{r,s\}})$ for some $s \in S \setminus \{r\}$. Letting $t(1), \ldots, t(n)$ denote the ordering in which we are summing out the variables, after $n-1$ steps we have decomposed $p(x)$ into n factors,

$$p(x) = \left[\prod_{k=1}^{n-1} p(x_{t(k)}|x_{t(l)}, l = k+1, \ldots, n)\right] p(x_{t(n)}), \tag{11}$$

where each of the n factors are represented by a vertex-weighted graph. In particular the last factor, $p(x_{t(n)})$ is represented by a graph with only two nodes, $\emptyset$ and $\{t(n)\}$. The normalizing constant of $p(x)$ is included also in $p(x_{t(n)})$ and thereby this constant can be evaluated from the restriction $p(x_{t(n)} = 0) + p(x_{t(n)} = 1) = 1$. Moreover, simulation from $p(x)$ is straight forward by a backward pass, first simulating $x_{t(n)}$ from $p(x_{t(n)})$, then $x_{t(n-1)}$ from $p(x_{t(n-1)}|x_{t(n)})$ and so on. To fully define the recursive algorithm it remains to discuss how to obtain $G_\star$ from G. We first develop relations that characterize $G_\star$ and thereafter discuss the algorithmic details of how to obtain $G_\star$ from G.

3.1 Characterization of $G_\star$

The energy function $U_\star(x_{-r})$ for $p(x_{-r})$ is defined by $p(x_{-r}) = c\exp\{-U_\star(x_{-r})\}$, where c is the same normalizing constant as in (2). Combining this with (2) and (7) we get

$$U_\star(x_{-r}) = -\ln\left[\sum_{x_r} \exp\left\{-\sum_{\Lambda\in\mathcal{B}} \beta(\Lambda) \prod_{k\in\Lambda} x_k\right\}\right]. \tag{12}$$

Splitting the sum over $\Lambda \in \mathcal{B}$ into a sum of two sums, one over $\Lambda \in \mathcal{B}_r$ and one over $\Lambda \in \mathcal{B}_{-r}$, and using that the latter sum is then not a function of x_r, we get

$$U_\star(x_{-r}) = U_\star^1(x_{N_r}) + U_\star^2(x_{-r}), \tag{13}$$

where $N_r = \{k \in S_{-r} | \{k, r\} \in \mathcal{B}\}$ is the (minimal) set of neighbors of r,

$$U^1_\star(x_{N_r}) = -\ln\left[1 + \exp\left\{-\sum_{\Lambda\in\mathcal{B}_r} \beta(\Lambda) \prod_{k\in\Lambda\setminus\{r\}} x_k\right\}\right] \tag{14}$$

and

$$U^2_\star(x_{-r}) = \sum_{\Lambda\in\mathcal{B}_{-r}} \beta(\Lambda) \prod_{k\in\Lambda} x_k. \tag{15}$$

We note that $U^2_\star(x_{-r})$ is already in our canonical form. As discussed in Section 2.1, we can recursively compute a set of interaction parameters for $U^1_\star(x_{N_r})$, which we denote by $\Delta\beta = \{\Delta\beta(\Lambda), \Lambda \in \mathcal{P}(N_r)\}$, to get also $U^1_\star(x_{N_r})$ in the canonical form,

$$U^1_\star(x_{N_r}) = \sum_{\Lambda\in\mathcal{P}(N_r)} \Delta\beta(\Lambda) \prod_{k\in\Lambda} x_k. \tag{16}$$

Thus, the interaction parameters for $U_\star(x_{-r})$, which we denote by $\{\beta_\star(\Lambda), \Lambda \subseteq S \setminus \{r\}\}$, is given as a sum of one term from each of $U^1_\star(x_{N_r})$ and $U^2_\star(x_{-r})$. The contribution from $U^1_\star(x_{N_r})$ is always vanishing when $\Lambda \not\subseteq N_r$, whereas the contribution from $U^2_\star(x_{-r})$ may be non-zero only when $\Lambda \in \mathcal{B}_{-r}$. Thus, for $\Lambda \subseteq S \setminus \{r\}$ we get

$$\beta_\star(\Lambda) = \begin{cases} \beta(\Lambda) + \Delta\beta(\Lambda) & \text{if } \Lambda \in \mathcal{B}_{-r} \cap \mathcal{P}(N_r), \\ \beta(\Lambda) & \text{if } \Lambda \in \mathcal{B}_{-r} \setminus \mathcal{P}(N_r), \\ \Delta\beta(\Lambda) & \text{if } \Lambda \in \mathcal{P}(N_r) \setminus \mathcal{B}_{-r}, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

In particular, the zero value in this expression gives a restriction on the set $\mathcal{B}_\star$, namely

$$\mathcal{B}_\star \subseteq \mathcal{B}_{-r} \cup \mathcal{P}(N_r). \tag{18}$$

3.2 Computation of $G_\star$

Based on the above equations and our representation of $p(x)$ by the vertex-weighted graph G we now discuss how to generate the updated graph $G_\star$, representing $p(x_{-r})$. The resulting algorithm is summarized in Figure 4 and to illustrate the process we again consider the toy graph G shown in Figure 2(b). The first step is to split the graph G into G_r and G_{-r}. For the toy example (the unweighted versions of) G_r and G_{-r} for $r = 3$ are shown in Figures 3(a) and (b), respectively. Note that to do this split it is not necessary to traverse all the vertices in G, it is sufficient to traverse all ancestors of the vertex $\{r\}$. From G_r the (minimal) set of neighbors to r, N_r, is easily found and from this the corresponding unweighted power set graph, $\mathcal{G}(\mathcal{P}(N_r))$ is generated. In the toy example we have $N_3 = \{2, 4\}$ and $\mathcal{G}(\mathcal{P}(N_3))$ is shown in Figure 3(c) The next step is to compute recursively and store the interaction parameters $\Delta\beta$ associated to the graph $\mathcal{G}(\mathcal{P}(N_r))$, thereby producing the vertex-weighted graph $\mathcal{G}(\mathcal{P}(N_r), \Delta\beta)$. Thereafter $\mathcal{G}(\mathcal{P}(N_r), \Delta\beta)$ should be added to G_{-r}. This implies that we need to traverse all the nodes in $\mathcal{G}(\mathcal{P}(N_r), \Delta\beta)$, starting from the root node $\emptyset$ and going upwards. When visiting a node $\Lambda \in \mathcal{P}(N_r)$, if the node Λ also exists in G_{-r} we should just add the weight $\Delta\beta(\Lambda)$ to the existing weight for Λ in G_{-r}. If Λ does not exist in G_{-r}, such a vertex should be added to G_{-r} and given the weight $\Delta\beta(\Lambda)$. The resulting graph, which we denote by $\widetilde{G}_\star$ is a representation of $p(x_{-r})$. However, it may include vertices with zero

1. Split the graph $G = \mathcal{G}(\mathcal{B}, \beta[\mathcal{B}])$ into the two subgraphs $G_r = \mathcal{G}(\mathcal{B}_r, \beta[\mathcal{B}_r])$ and $G_{-r} = \mathcal{G}(\mathcal{B}_{-r}, \beta[\mathcal{B}_{-r}])$. Store G_r as a representation of the full conditional $p(x_r|x_{-r})$.

2. From G_r find the (minimal) set of neighbors of r, $N_r = \{k \in S_{-r} | \{k, r\} \in \mathcal{B}_r\}$ and establish the graph of the corresponding power set, $\mathcal{G}(\mathcal{P}(N_r))$.

3. Compute recursively the interaction parameters $\Delta\beta = \{\Delta\beta(\Lambda), \Lambda \in \mathcal{P}(N_r)\}$ and store them in the vertices of the graph $\mathcal{G}(\mathcal{P}(N_r))$, thereby generating the vertex-weighted graph $\mathcal{G}(\mathcal{P}(N_r), \Delta\beta)$.

4. Add the graph $\mathcal{G}(\mathcal{P}(N_r), \Delta\beta)$ to the graph to G_{-r}. Denote the resulting graph by $\widetilde{G}_\star = \mathcal{G}(\widetilde{\mathcal{B}}_\star, \beta_\star[\widetilde{\mathcal{B}}_\star])$, where $\widetilde{\mathcal{B}}_\star = \mathcal{B}_{-r} \cup \mathcal{P}(N_r)$.

5. Prune the graph $\widetilde{G}_\star$ by deleting all vertices (and associated edges) $\Lambda \in \widetilde{B}_\star$ for which $\beta_\star(A) = 0$ for all $A \in \{A \in \widetilde{B}_\star | \Lambda \subseteq A\}$. The resulting graph, $G_\star$, is then a representation of $p(x_{-r})$.

Figure 4: Algorithm for generating G_r and $G_\star$ from G.

weight that should not be included according to our criterion in (7). To obtain the canonical representation of $p(x_{-r})$ any such nodes (and associated edges) should be deleted. However, note that if this pruning is not performed the algorithm will still be correct in the sense that the same conditional distribution will be found, but potentially with a somewhat higher computational cost in later iterations of the algorithm. As one also saves computation time by not performing the pruning it is not clear whether it is beneficial to perform the pruning or not. In our current implementation of the algorithm we have chosen not to perform the pruning.

The computational complexity of finding G_r and $G_\star$ from G is proportional to the size of $\mathcal{P}(N_r)$, i.e. it is exponential in the number of neighbors of r. It is thereby only feasible to perform the algorithm when the number of neighbors is reasonably low. When iterating the procedure it becomes impossible to give general results for the cost of the algorithm, as this will depend on the whole neighborhood system and typically also the ordering $t(1), \ldots, t(n)$ used. Next we define a computationally cheaper, but approximate version of the exact recursive algorithm.

4 Approximate recursive computations

The exact algorithm described above is not computationally feasible when the number of neighbors to node r is large. There are two problems, both related to the large number of vertices in the vertex-weighted graph $\mathcal{G}(\mathcal{P}(N_r), \Delta\beta)$. First, it requires too much computation time to evaluate all $\Delta\beta(\Lambda), \Lambda \in \mathcal{P}(N_r)$. Second, even if the interaction parameters could have been computed it would require too much computer memory to store them all.

What we observe when computing and studying the interaction parameters $\Delta\beta(\Lambda), \Lambda \in \mathcal{P}(N_r)$ for frequently used Markov random field models (small enough to allow exact recursive computation) is that most of the interaction parameters have values very close to zero. Figure

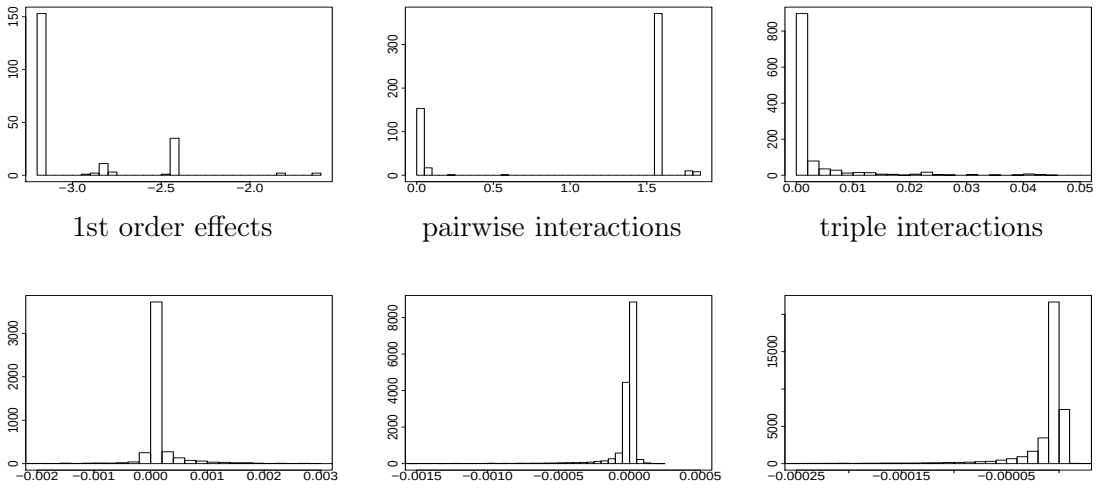


Figure 5: Values of interaction parameters for an Ising based model. The exact model used is specified in Section 7. In lexicographical order the six plots show histograms of the values of $\Delta\beta(\Lambda)$ for all sets $\Lambda \in \mathcal{P}(N_r)$ with $|\Lambda| = 1, 2, 3, 4, 5$ and 6, respectively. Note the difference in horizontal scale.

5 shows $\Delta\beta(\Lambda)$ for all $\Lambda \in \mathcal{P}(N_r)$ with $|\Lambda| = 1, 2, 3, 4, 5$ and 6 for an Ising based model. The exact model used here is defined in Section 7. A natural approximation strategy is therefore, for some threshold value ε, to approximate $\Delta\beta(\Lambda)$ to zero whenever $|\Delta\beta(\Lambda)| < \varepsilon$. This solves the memory problem discussed above, but not the computation time problem as we still have to compute $\Delta\beta(\Lambda)$ for all $\Lambda \in \mathcal{P}(N_r)$ in order to decide which to store and which to approximate to zero. To cope also with the computation time problem we assume that $\Delta\beta(\Lambda)$ is close to zero whenever $\Delta\beta(A)$ is close to zero for all sets $A \subset \Lambda$ with exactly one element less than Λ. Note that this is a frequently used assumption in statistics, higher order effects can be present only if corresponding lower order effects are present. Thus, in the graph $\mathcal{P}(N_r)$ we approximate $\Delta\beta(\Lambda)$ to zero, without computing its exact value, whenever the interaction parameters of all children vertices of Λ is already approximated to zero. We have checked this assumption in several frequently used Markov random fields and have not been able to find cases where it is violated. However, we do not expect the assumption to be valid in all cases and it should certainly be possible to construct models where it fails. However, in our experience it seems to be valid, and certainly very useful, for most frequently used Markov random field models.

The two approximation rules defined above define our approximation. The corresponding algorithm is equal to the exact version in Figure 4 except that $\mathcal{P}(N_r)$ is replaced by the smaller set of vertices for which the interaction parameters is not approximated to zero. Thus, the joint distribution $p(x)$ is decomposed into the exact full conditional $p(x_r|x_{-r})$ and an approximate $\widetilde{p}_\varepsilon(x_{-r})$. In the next step of the recursive algorithm $\widetilde{p}_\varepsilon(x_{-r})$ is decomposed into $\widetilde{p}_\varepsilon(x_s|x_{-\{r,s\}})$, which of course is only an approximation to $p(x_s|x_{-\{r,s\}})$, and after new approximations

$\widetilde{p}_\varepsilon(x_{-\{r,s\}})$. Ultimately we end up with an approximate version of (11),

$$p(x) \approx \widetilde{p}_\varepsilon(x) = p(x_{t(1)}|x_{t(l)}, l = 2, \ldots, n) \left[\prod_{k=2}^{n-1} \widetilde{p}_\varepsilon(x_{t(k)}|x_{t(l)}, l = k+1, \ldots, n) \right] \widetilde{p}_\varepsilon(x_{t(n)}). \quad (19)$$

Thus, the normalizing constant of $p(x)$ can be approximated with the normalizing constant of the right hand side, and approximate samples from $p(x)$ can be generated be sampling from the approximate distribution. We end this section by three remarks.

Remark 1. Our focus when defining the approximation is to make it computationally feasible to decompose $p(x)$ into $p(x_r|x_{-r})$ and $\widetilde{p}_\varepsilon(x_{-r})$. However, approximating interaction parameters to zero also introduces conditional independence to $\widetilde{p}_\varepsilon(x_{-r})$ that is not present in the corresponding exact $p(x_{-r})$. 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 $\mathcal{P}(N_r)$ into three groups. The first group consists of the sets Λ for which we compute $\Delta\beta(\Lambda)$, find $|\Delta\beta(\Lambda)| \geq \varepsilon$ and thereby store $\Delta\beta(\Lambda)$ in memory. The second group contains the sets Λ for which we compute $\Delta\beta(\Lambda)$ and find $|\Delta\beta(\Lambda)| < \varepsilon$. The third group consists of the remaining sets Λ, for which we do not compute $\Delta\beta(\Lambda)$. When the number of elements in N_r is large it is essential for the approximate algorithm to be feasible that most of the sets $\Lambda \in \mathcal{P}(N_r)$ end up in the third group.

Remark 3. The quality of the approximation, the computation time and the memory requirements of the approximate recursive algorithm clearly depend on the threshold value ε. How small ε needs to be for the algorithm to give a reasonable approximation needs to be explored empirically. In Section 7 we report our experience with both this and the associated computer resources required by the algorithm in a number of frequently used binary Markov random fields. In particular note that we have defined the approximations so that the algorithm becomes exact for $\varepsilon = 0$.

5 Beyond binary fields

The focus of this paper is binary Markov random fields. In this section, however, we shortly discuss how the above algorithms can be generalized to handle also discrete Markov random fields with more than two possible values. Thus, in this section 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 algorithm discussed above to such a situation. The first is to start with a generalized version of (3),

$$U_z(z) = \sum_{\Lambda \subseteq S} \beta(\Lambda, z_\Lambda) \prod_{k \in \Lambda} z_k. \quad (20)$$

Thus, in stead of having only one interaction parameter for a set Λ we now have $(K-1)^{|\Lambda|}$ parameters associated to Λ. Again the interaction parameters can be computed recursively and approximations following the same ideas as discussed in Section 4 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 Evaluation criteria for approximation

To choose criteria for evaluating the quality of an approximation we must take into account how the approximation will be used. Here we discuss three Bayesian applications of the approximation defined above. However, our approximation can clearly also be of use in frequentist settings. First, assume that a binary field x is observed and that this is assumed to be a realization from a Markov random field $p(x|\theta)$. A prior distribution for θ, $p(\theta)$, is adopted and the interest is in the resulting posterior distribution $p(\theta|x) \propto p(\theta)p(x|\theta)$. The computational problem then is that the intractable normalizing constant of the likelihood $p(x|\theta)$ is a function of θ. Thereby $p(\theta|x)$ is not easily available to us. By replacing the computationally problematic $p(x|\theta)$ by the approximation defined above, now denoted $\widetilde{p}_\varepsilon(x|\theta)$, we get an approximate posterior distribution $\widetilde{p}_\varepsilon(\theta|x) \propto p(\theta)\widetilde{p}_\varepsilon(x|\theta)$. To evaluate the quality of this approximation it is natural to focus on $\widetilde{p}_\varepsilon(\theta|x)$. For examples with a small number of nodes, so that exact computations of $p(x|\theta)$ are feasible, one can compare the approximate posterior $\widetilde{p}_\varepsilon(\theta|x)$ for different values of the threshold ε with the exact posterior $p(\theta|x)$. In more realistic situations, where the exact $p(x|\theta)$ is not computationally available, the natural alternative is to compare the approximate $\widetilde{p}_\varepsilon(\theta|x)$ for different values of ε. If $\widetilde{p}_\varepsilon(\theta|x)$ seems to stabilize for sufficiently small values of ε it is reasonable to trust the approximation.

The second application we consider is a small variation of the first. Now let x be a latent unobserved variable, still assumed distributed according to $p(x|\theta)$, and assume we observe a vector y according to a distribution $p(y|x,\theta)$. Still assuming a prior $p(\theta)$ for θ the posterior distribution of interest is $p(\theta|y) \propto p(\theta)p(y|\theta)$. To find an expression for the marginal likelihood $p(y|\theta)$ we can note that

$$p(x|y,\theta) = \frac{p(x|\theta)p(y|x,\theta)}{p(y|\theta)} \tag{21}$$

for any value of x. Thus, by solving this expression with respect to $p(y|\theta)$ we get

$$p(\theta|y) = \frac{p(\theta)p(x|\theta)p(y|x,\theta)}{p(x|y,\theta)}, \tag{22}$$

again for any value of x. Assuming $p(y|x,\theta)$ to be computationally available, the problematic factors in (22) are $p(x|\theta)$ and $p(x|y,\theta)$. Both are binary Markov random fields and thereby, if the normalizing constants of any of them is not computationally available, the problematic factor(s) can be replaced by the corresponding approximation(s) defined above. Typically, both normalizing constants are problematic and the approximation becomes

$$\widetilde{p}_\varepsilon(\theta|y) = \frac{p(\theta)\widetilde{p}_\varepsilon(x|\theta)p(y|x,\theta)}{\widetilde{p}_\varepsilon(x|y,\theta)}. \tag{23}$$

Here we have assumed that the same threshold value ε is used for both approximations, but this is of course not necessary. The right hand side of (22) is by construction constant as a function of x, whereas one should not expect this to be true for the corresponding approximation in (23). An important question when applying (23) is thereby what value to use for x. We have used two strategies, *i*) to sample x from $\widetilde{p}_\varepsilon(x|y,\theta)$ and *ii*) to let x be a vector of only zeros. Our experience is that if ε is sufficiently small both strategies works satisfactory, and conversely, if ε is not sufficiently small none of the two alternatives produce satisfactory results. To evaluate the quality of the approximation $\widetilde{p}_\varepsilon(\theta|y)$ we can use the same strategy as for $\widetilde{p}_\varepsilon(\theta|x)$, either compare $\widetilde{p}_\varepsilon(\theta|y)$ with the exact $p(\theta|y)$ if this is computationally available, or otherwise

compare the approximation $\widetilde{p}_\varepsilon(\theta|y)$ for different values of ε and trust the approximation if $\widetilde{p}_\varepsilon(\theta|y)$ stabilizes for sufficiently small values of ε.

The third application of the approximation we consider here is to use the approximation in the construction of Møller et al. (2006). As we discussed in the introduction, the goal is then to sample from the posterior $p(\theta|x) \propto p(\theta)p(x|\theta)$. For this an auxiliary variable Metropolis–Hastings algorithm is constructed. The algorithm simulates (θ, z) from the target distribution $p(\theta|x)\widetilde{p}_\varepsilon(z|\theta)$, i.e. z is an auxiliary variable of the same dimension as the observed x. In each iteration a potential new value for θ, θ', is first generated from a proposal distribution $q(\theta'|\theta)$. Then a potential new value for z, z', is generated from $p(z'|\theta')$ by exact sampling, and finally θ' and z' are jointly accepted with probability

$$\alpha(\theta', z'|\theta, z) = \min\left\{1, \frac{p(\theta')}{p(\theta)}\frac{p(x|\theta')}{p(x|\theta)}\frac{\widetilde{p}_\varepsilon(z'|\theta)}{\widetilde{p}_\varepsilon(z|\theta)}\frac{q(\theta|\theta')}{q(\theta'|\theta)}\frac{p(z|\theta)}{p(z'|\theta')}\right\}. \tag{24}$$

The ingenious part of this construction is that all the computationally intractable normalizing constants cancel from this expression. As pointed out in Møller et al. (2006) we can also note that if the approximation is perfect, then the third and fifth factors in (24) cancel and we are left with the acceptance probability of a standard Metropolis–Hastings algorithm simulating from the posterior $p(\theta|x)$. What we loose by using the approximation $\widetilde{p}_\varepsilon(\cdot|\cdot)$ in stead of the exact $p(\cdot|\cdot)$ is therefore given by the third and fifth factors of (24). Setting θ' equal to θ in (24) we are left with exactly these two factors,

$$\alpha(\theta, z'|\theta, z) = \min\left\{1, \frac{\widetilde{p}_\varepsilon(z'|\theta)}{\widetilde{p}_\varepsilon(z|\theta)}\frac{p(z|\theta)}{p(z'|\theta)}\right\}, \tag{25}$$

which we recognize as the acceptance probability of a independent proposal Metropolis–Hastings algorithm with target distribution $\widetilde{p}_\varepsilon(\cdot|\cdot)$ and proposal distribution $p(\cdot|\theta)$. To quantify the quality of the approximation in this setting it is therefore natural to focus on the average acceptance probability of such a Metropolis–Hastings algorithm, i.e. the mean value of (25) when $z \sim p(z|\theta)$ and $z' \sim \widetilde{p}_\varepsilon(z'|\theta)$, independently. We note in passing that this is also identical to the mean acceptance probability of an independent proposal Metropolis–Hastings algorithm with target distribution $p(\cdot|\cdot)$ and proposal distribution $\widetilde{p}_\varepsilon(\cdot|\cdot)$.

7 Examples

In this section we consider a number of Markov random fields and discuss the feasibility of our algorithms for these models. We start looking at simple 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 for small lattices only. Here we evaluate empirically the quality and effectiveness of the approximate algorithm for different values of the threshold value ε.

7.1 Models where the exact algorithm is particularly efficient

Let $S = \{1, \ldots, n\}$ and assume x to be Markov with respect to the neighborhood system defined by the graph in Figure 6(a). The most general form of the energy function is then

(a) (b)

Figure 6: Two undirected graphs $V = (S, E)$. A pairwise interaction, binary field which is Markov with respect to one of these graphs can very efficiently be handled by the exact algorithm described in Section 3. (a) A chain graph and (b) a somewhat more complex graph.

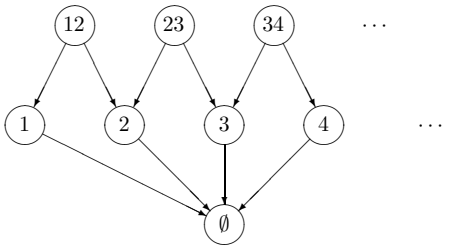


Figure 7: The graph $\mathcal{G}(\mathcal{B})$ for the energy function (26) if $\beta_{k,k+1} \neq 0$ for all $k = 1, \ldots, n-1$.

$$U(x) = \alpha_0 + \sum_{k=1}^{n} \alpha_k x_k + \sum_{k=1}^{n-1} \beta_{k,k+1} x_k x_{k+1}, \tag{26}$$

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 the observed data. It is easy to see that for this model the exact algorithm with $t(k) = k, k = 1, \ldots, n$ 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 $\mathcal{G}(\mathcal{B})$ and how this develops when running the iterations. The function (26) is already in the canonical form (7) so $\beta(\emptyset) = \alpha_0$, $\beta(\{k\}) = \alpha_k$ for $k = 1, \ldots, n$ and $\beta(\{k, k+1\}) = \beta_{k,k+1}$ for $k = 1, \ldots, n-1$. If none of the $\beta_{k,k+1}$'s are equal to zero the graph $\mathcal{G}(\mathcal{B})$ becomes as shown in Figure 7. In the first iteration of the recursive algorithm, when summing out x_1, we get $\mathcal{B}_1 = \{\{1\}, \{1, 2\}\}$ and thereby $N_1 = \{2\}$ and $\mathcal{P}(N_1) = \{\emptyset, \{2\}\}$. In turn this gives $B_\star = \mathcal{B}_{-1}$, so $\mathcal{B}_\star$ is of the same form as $\mathcal{B}$. Thereby, the situation repeats when summing out x_2, x_3 and so on. What is important for the efficiency of the algorithm is that all distributions $p(x_{-t(1)}), p(x_{-\{t(1),t(2)\}}), \ldots$ are pairwise interaction models. For this to be the case the neighbor set $N_{t(1)}$, 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 for any ordering $t(1), \ldots, t(n)$. For

example, if $t(1) = 3$ we get $N_3 = \{2, 4\}$ and $\mathcal{P}(N_3) = \{\emptyset, \{2\}, \{4\}, \{2, 4\}\}$, so $\mathcal{B}_\star = \mathcal{B}_{-3}$.

The requirement that $N_{t(1)}$, and corresponding sets in later iterations, contains at most two elements is, for suitable orderings $t(1), \ldots, t(n)$, fulfilled also for other models than just (26). By drawing up the relevant graphs it is easy to check that any pairwise interaction, binary field that are Markov with respect to a neighborhood system defined by a graph that contains no loops, i.e. a tree, fulfils the requirement. Moreover, even for graphs that contain loops the corresponding pairwise interaction Markov random field may fulfil the formulated condition. An example of this with $n = 11$ is shown in Figure 6(b). Again drawing up the relevant graphs, one can easily check that the ordering $t(k) = k, k = 1, \ldots, 11$ works in this case. However, if in that graph we add an edge between 2 and 3 there no longer exists any ordering that satisfies the set requirements. We have not been able to formulate an easy to check criterion for undirected graphs that can characterize the set of Markov random fields that fulfil the requirements. However, our discussion here should clearly demonstrate that the exact recursive algorithm is highly efficient for a larger class of models than where the forward-backward algorithm is typically used today. Of course, the algorithm may also be highly efficient even if a few of the neighborhood sets contains slightly more than two elements.

7.2 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 (7) with $\mathcal{B} = \{\emptyset, \{1\}, \ldots, \{n\}\} \cup \{(k, l) | l \in N_k, k, l = 1, \ldots, n\}$, i.e.

$$U(x) = \beta(\emptyset) + \sum_{k=1}^{n} \beta(\{k\}) x_k + \frac{1}{2} \sum_{k=1}^{n} \sum_{l \in N_k} \beta(\{k, l\}) x_k x_l. \tag{27}$$

Strictly speaking the models discussed in Section 7.1 are thereby autologistic models. However, the term is usually reserved for models defined on a two dimensional rectangular lattice, so 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 the lexicographical order. Except when many of the $\beta(\{k, l\})$'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 in the previous section. The algorithms in Pettitt et al. (2003), Reeves and Pettitt (2004) and Friel and Rue (2007) are essentially the same as our exact algorithm with the ordering $t(k) = k$ for $k = 1, \ldots, n$. 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 limit that can be handled by the exact algorithm is even smaller. In the following we focus on evaluating empirically the performance of the approximate algorithm of Section 4 for autologistic models.

To evaluate the approximate algorithm we consider the computation time for running the algorithm and the quality of the approximation. In the simulation examples below we first focus on the latter. To monitor the quality of the approximation we adopt the three strategies discussed in Section 6.

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

$\varepsilon \setminus \theta_{\text{true}}$	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	$1.77 \cdot 10^{-1}$	$1.11 \cdot 10^{-1}$	$9.81 \cdot 10^{-2}$

7.3 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

$$U(x) = -\frac{\theta}{2} \sum_{k=1}^{n} \sum_{l \in N_k} I(x_k = x_l), \tag{28}$$

where $I(\cdot)$ is the indicator function. 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 (27) and gives

$$\begin{aligned} \theta(\emptyset) &= -\frac{\theta}{2} \sum_{k=1}^{n} |N_k|, \\ \theta(\{k\}) &= |N_k|\theta, \text{ for } k = 1, \ldots, n, \\ \theta(\{k, l\}) &= -2\theta, \text{ for } l \in N_k, k, l = 1, \ldots, n, \end{aligned} \tag{29}$$

where $|N_k|$ is the number of elements in the set N_k. We first consider the quality of the approximation 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, consider the resulting posterior distribution for θ given x, i.e. $p(\theta|x)$. As prior 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 between the mesh values. Finally, we numerically evaluate

$$d_0(\varepsilon, x) = \int_0^\infty |\widetilde{p}_\varepsilon(\theta|x) - p(\theta|x)| \, \mathrm{d}\theta \tag{30}$$

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

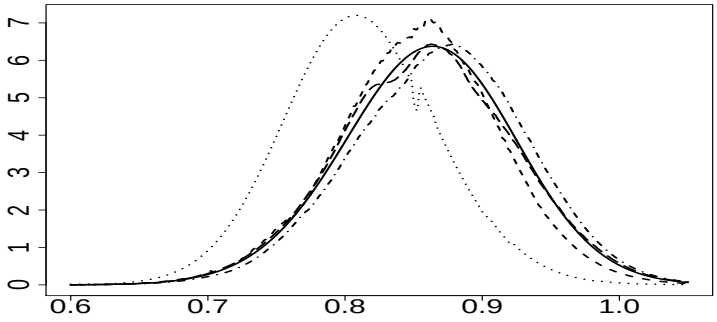


Figure 8: Results for the Ising model on a 15×15 lattice: $p(\theta|x)$ (solid) and $\widetilde{p}_\varepsilon(\theta|x)$ for $\varepsilon = 10^{-1}$ (dotted) , 10^{-2} (dashed), 10^{-3} (dot-dashed) and 10^{-4} (long dashed). With the resolution used here the approximations for $\varepsilon = 10^{-5}$ and 10^{-6} are visually indistinguishable from the exact $p(\theta|x)$.

Table 2: Results for the Ising model on a 100×100 lattice: Values of $d(\varepsilon, x)$ for realizations x generated from the Ising model for each of the parameter values $\theta_{\text{true}} = 0.4$, 0.6 and 0.8. Results are given for five different values of ε and for the pseudo likelihood and block pseudo likelihood approximation.

$\varepsilon \setminus \theta_{\text{true}}$	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

ε. Not surprisingly, smaller values of ε is necessary to obtain a good approximation when the value of θ_{true} is larger. For comparison we also compute the same quantities when using the pseudo likelihood approximation to $p(\theta|x)$ and for a pseudo block likelihood approximation with 15×5 blocks. The results are again given in Table 1 and we see that $\widetilde{p}_\varepsilon(\theta|x)$ gives a much better approximation except for the larger values of ε. To get a visual impression of the accuracy of the approximations Figure 8 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.

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

$$d(\varepsilon, x) = \int_0^\infty \left| \widetilde{p}_\varepsilon(\theta|x) - \widetilde{p}_{\varepsilon/10}(\theta|x) \right| \mathrm{d}\theta \tag{31}$$

for the same values for θ_{true} and ε considered above. Table 2 summarizes the results. For the

Table 3: Results for the hidden Ising model on a 100×100 lattice: Values of $d(\varepsilon, y)$ for realizations y generated based on realizations from the Ising model for each of the parameter values $\theta_{\text{true}} = 0.4$, 0.6 and 0.8. Results are given for five different values of ε.

$\varepsilon \setminus \theta_{\text{true}}$	0.4	0.6	0.8
10^{-1}	$5.26 \cdot 10^{-1}$	1.99	2.00
10^{-2}	$2.49 \cdot 10^{-1}$	1.02	1.72
10^{-3}	$5.13 \cdot 10^{-2}$	$5.89 \cdot 10^{-2}$	NA
10^{-4}	$1.32 \cdot 10^{-2}$	$2.34 \cdot 10^{-2}$	NA
10^{-5}	$2.15 \cdot 10^{-3}$	NA	NA

$\theta_{\text{true}} = 0.8$ case, the computer resources required to compute $\widetilde{p}_\varepsilon(\theta|x)$ for $\varepsilon = 10^{-5}$ and 10^{-6} was very large, so we did not run these cases. In the evaluation of the pseudo likelihood and pseudo block likelihood approximations we compute the difference to $\widetilde{p}_\varepsilon(\theta|x)$ with $\varepsilon = 10^{-6}$, and for the pseudo block likelihood we use blocks of size 100×5. 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.

Then we consider the hidden Markov random field situation discussed in Section 6. More precisely, for each $\theta_{\text{true}} = 0.4$, 0.6 and 0.8 we first generate a realization, $x = (x_1, \ldots, x_n)$, from the Ising model on a 100×100 lattice and thereafter generate $y = (y_1, \ldots, y_n)$ where $y_1, \ldots, y_n$ are conditionally independent given x and $y_i|x \sim \mathrm{N}(x_i, 0.25^2)$. As before we adopt a uniform prior distribution for θ on $[0, \infty)$. As defined in Section 6 we compute approximate posterior distributions $\widetilde{p}_\varepsilon(\theta|y)$ for the same six values of ε as before, and finally evaluate numerically an integral corresponding to the one in (31), but with x replaced by y. The resulting values are given in Table 3, again some cases are not evaluated because they required too much computer resources. When looking at $\widetilde{p}_\varepsilon(\theta|y)$ for large values of ε one clearly sees the effect of evaluating the right hand side of (23) for a random x, the curve gets a "noisy" appearance. As we should expect the noisy appearance vanishes if in stead $x = 0$ is used, but when comparing results for various values of ε to find how small ε needs to be to get a good approximation, both approaches produce the same conclusion. We therefore favor to evaluate the right hand side of (23) at a random x as then one can use the "noisiness" to diagnose whether a given value for ε gives a good approximation.

Finally, we evaluated the approximation by estimating the mean acceptance probability of a Metropolis–Hastings algorithm with target distribution $p(z|\theta)$ and proposal distribution $\widetilde{p}_\varepsilon(z|\theta)$. For each of $\theta = 0.4$, 0.6 and 0.8 we generated 1000 independent realizations from $p(x|\theta)$ by the coupling from the past algorithm (Propp and Wilson, 1996) and for the various values of ε, 1000 realizations from the approximation $\widetilde{p}_\varepsilon(x|\theta)$. We then estimated the mean acceptance probability by taking the mean of the 1000000 acceptance probabilities generated by combining each of the 1000 realizations from $p(x|\theta)$ with each of the 1000 realizations from $\widetilde{p}_\varepsilon(x|\theta)$. The results are shown in Figure 9. The solid, dotted and dashed curves are for $\theta = 0.4$, 0.6 and 0.8, respectively. Again we have no available results for the small values of ε when $\theta = 0.8$ because the necessary computations required too much computation time. Consistent with what we saw in Table 2 we find that the approximation is indeed very good for $\varepsilon \leq 10^{-4}$ for $\theta = 0.4$ and 0.6, whereas for $\theta = 0.8$ we are not able to find a very good

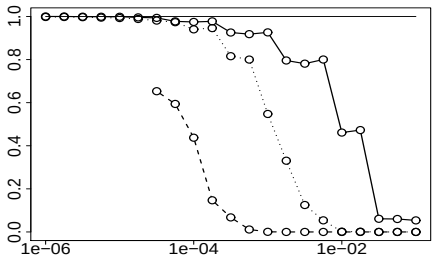


Figure 9: Results for the Ising model on a 100×100 lattice: Estimated acceptance rates as a function of ε for an independent proposal Metropolis–Hastings algorithm with an Ising target distribution when the corresponding approximate $\widetilde{p}_\varepsilon(x|\theta)$ is used as proposal distribution. The solid, dotted and dashed curves show results for $\theta = 0.4$, 0.6 and 0.8, respectively.

Table 4: Results for the Ising model on a 100×100 lattice: Computation time used to establish the approximation $\widetilde{p}_\varepsilon(x|\theta)$ and the additional time necessary for generating 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

$\varepsilon \setminus \theta$	0.4	0.6	0.8
10^{-1}	0.11	0.12	0.19
10^{-2}	0.19	0.39	2.32
10^{-3}	0.82	3.83	60.01
10^{-4}	2.38	27.89	5 640.50
10^{-5}	15.21	279.00	370 303.80
10^{-6}	41.27	1 948.37	NA

Additional time per realization

$\varepsilon \setminus \theta$	0.4	0.6	0.8
10^{-1}	0.07	0.06	0.05
10^{-2}	0.09	0.12	0.21
10^{-3}	0.17	0.41	1.52
10^{-4}	0.27	1.10	34.56
10^{-5}	0.68	3.70	921.93
10^{-6}	1.14	11.52	NA

approximation within reasonable computation time.

Above we have evaluated the approximation quality as a function of θ and ε. To fully evaluate the usefulness of the approach one clearly also needs to consider the computation times required. The computations involved can be divided into 1) what is necessary to establish the approximation $\widetilde{p}_\varepsilon(x|\theta)$ and 2) the additional time required to generate one realization from the approximation or to evaluate $\widetilde{p}(x|\theta)$ for a given x. For our implementation and a 100×100 lattice, Table 4 shows both computation times for the θ and ε values used above. Recalling that for $\theta = 0.4$ and 0.6 very good approximations are produced for $\varepsilon = 10^{-4}$ we see that these approximations are also quite efficiently available, especially for $\theta = 0.4$. For $\theta = 0.8$ the situation is less favorable and whether or not the approximation is at all of any use for this value of θ depends on the problem of interest.

7.4 A pairwise interaction 5×5 neighborhood model

Last 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

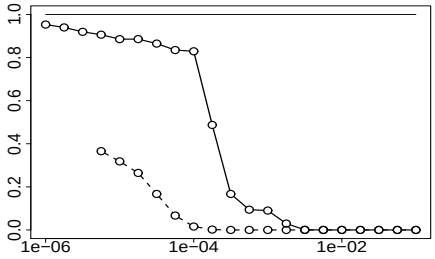


Figure 10: Results for the pairwise interaction 5×5 neighborhood model on a 100×100 lattice: Estimated acceptance rates as a function of ε for an independent proposal Metropolis–Hastings algorithm when the pairwise interaction 5×5 neighborhood model is used as target distribution and the corresponding approximate $\widetilde{p}_\varepsilon(x|\theta)$ is used as proposal distribution. The solid and dashed curves show results for $\theta = 0.1$ and 0.15, respectively.

Table 5: Results for the pairwise interaction 5×5 neighborhood model on a 100×100 lattice: Computation time used to establish the approximation $\widetilde{p}_\varepsilon(x|\theta)$ 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 4.

Time to establish approximation

$\varepsilon \setminus \theta$	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

$\varepsilon \setminus \theta$	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

the locations of the nodes to be positioned from 1 to 100 along each coordinate axis and let $D(i, j)$ denote the Eucledian distance between nodes i and j. The energy function we consider can then be expressed as

$$U(x) = -\theta \sum_{k \sim l} \frac{I(x_k = x_l)}{D(k, l)}, \tag{32}$$

where the sum is over pairs of nodes that are neighbors. We consider the model for $\theta = 0.1$ and for $\theta = 0.15$, and apply our approximate recursive algorithm for various values of ε. To evaluate the quality of the resulting approximations we estimate 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 10, and Table 5 gives the associated computation times.

8 Closing remarks

Using a canonical representation of binary Markov random fields we have defined exact and approximate recursive algorithms for this model class. The exact algorithm is essentially the same as previously defined in Reeves and Pettitt (2004) and Friel and Rue (2007). What is new is how we combine our canonical representation with the recursive scheme and how this enables us to define our corresponding approximative recursive algorithm. We have explored the quality of our approximate algorithm in a number of simulation examples and demonstrated how it can be used in several scenarios. In particular we have obtained accurate approximations for the Ising model, a model that is frequently used in spatial statistics. Even though we here have limited the attention to binary fields the approximation strategy we have used is applicable also for stochastic fields with more than two colors. However, the computational complexity grows rapidly with the number of colors, so it is feasible only for a small number of colors.

The results in our simulation examples demonstrate that the procedure we propose is feasible in situations of practical importance. However, our results also shows the limitations of our 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 other models not discussed in this paper, including higher order interaction Markov random fields (Descombes et al., 1995; Tjelmeland and Besag, 1998). What is important for the practicality of our approximation algorithm seems not to be interaction level, but rather the degree of correlation between node values in $p(x)$.

Our setup can be modified in several ways. First, we adopted the basis functions $f_\Lambda(x) = \prod_{i\in\Lambda} x_i$, $\Lambda \in \mathcal{P}(S)$ to define our canonical representation of $p(x)$. One may imagine alternative sets of basis functions. What is important for the efficiency of the algorithm is that the corresponding parameters $\beta(\Lambda)$ can be computed recursively and that most of these parameters can be approximated to zero. Second, by maximizing over each variable in turn in stead of summing them out as we do here, one may define approximate variants of the Viterbi algorithm, see for example Künsch (2001), thereby finding an approximation to the most probable state. Third, whenever the target distribution $p(x)$ is defined on a regular lattice, the parameters $\beta(\Lambda)$ 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 nodes. Thus, taking advantage of this stationarity we only need to sum out in detail the first few lines and the last one. This idea has previously been used for a different approximation scheme in Pettitt et al. (2003). In our simulation examples we have not taken advantage of this idea, so here there is a potential for drastically reducing the computation times reported in Tables 2 and 5. Last, even though we have formulated our summation procedure for a general permutation $t(1), \ldots, t(n)$, we are summing out the node variables in the lexicographical order in all our simulation examples presented in this paper. In runs not discussed here we have also tried other permutations, including permutations based on the multigrid (Goodman and Sokal, 1989) and divide and conquer (Golub and van Loan, 1996; Rue, 2001) ideas, but the results are inferior relative to summing out the variables in the lexicographical order. Still, however, we suspect that summing orders better than the lexicographical one exist.

It should be noted that the approximate distribution we are constructing, $\widetilde{p}_\varepsilon(x)$, is in fact a partially ordered Markov model (POMM), see the definition in Cressie and Davidson (1998). The induced partial ordering can be studied by looking at $\widetilde{p}_\varepsilon(x_{t(k)}|x_{t(l)}, l = k+1, \ldots, n)$. This

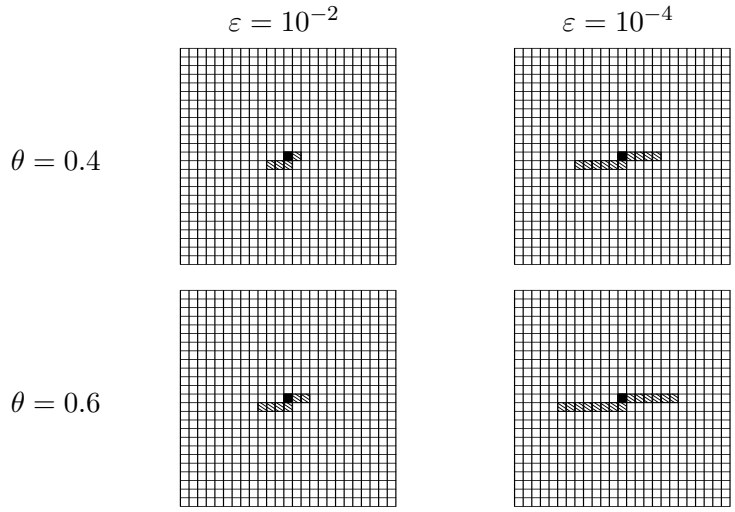


Figure 11: Conditional dependence structure for $\widetilde{p}_\varepsilon(x_{t(k)}|x_{t(l)}, l = k+1, \ldots, n)$ corresponding to an Ising model when the node variables of summed out in the lexicographical order. Results of shown for a node k well away from borders and for two values of θ and two values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

conditional distribution is really only a function of a subset of $x_{t(l)}, l = k+1, \ldots, n$, partly because of the Markov property of the original $p(x)$ and partly because of the approximation we are doing when setting some of the interaction parameters equal to zero. When summing out the variables in an Ising model in the lexicographical order, Figure 11 shows the sets of variables that the $\widetilde{p}_\varepsilon(x_{t(k)}|x_{t(l)}, l = k+1, \ldots, n)$ ends up being 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 the nodes that are close to $t(k)$, and it becomes a function of more variables when θ increases or ε decreases. From the partial ordering given by the dependence structure, level sets can be identified as discussed in Cressie and Davidson (1998) and in the backward simulation part of the algorithm variables in the same level set can be simulated in parallel. Even though our $\widetilde{p}_\varepsilon(x)$ is formerly a POMM, the procedure we propose for constructing the POMM in this paper 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 apriori specified.

Acknowledgements

We acknowledge support from The Research Council of Norway, Statoil and ENI.

References

Bartolucci, F. and Besag, J. (2002). A recursive algorithm for Markov random fields, *Biometrika* **89**: 724–730.

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems, *J. R. Statist. Soc. B* **36**: 192–225.

Besag, J. (1986). On the statistical analysis of dirty pictures (with discussion), *J. R. Statist. Soc. B* **48**: 259–302.

Besag, J., York, J. and Mollié, A. (1991). Bayesian image restoration, with two applications in spatial statistics, *Annals of the Institute of Statistical Mathematics* **43**: 1–59.

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, *J. Am. Statist. Ass.* **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.

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

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, *J. R. Statist. Soc. 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, *J. R. Statist. Soc. B* **63**.

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, *Scand. J. Statist.* **25**: 415–433.