An Introduction to `PottsUtils`

Dai Feng
dai_feng@merck.com

Package `PottsUtils` comprises several functions related to the Potts models defined on undirected graphs. The main purpose of the package is to make available several functions that generate samples from the models. To facilitate that, there are other utilities. Furthermore, there is a by-product of simulation functions. Altogether, there are three sets of functions. The first produces basic properties of a graph to facilitate the simulation functions (they maybe used for other purposes as well). The second provides various simulation functions. The third currently includes only one function which computes the normalizing constant based on simulation results.

This introduction was intended to help users to understand better the functions, the documentation (Rd files), and the source code. For more technical details (mathematical proof among others), we refer users to references herein.

Hereafter, first we introduce some basic concepts and definitions related to the Potts models. Second, algorithms used in simulation functions are presented in a concise way. Third, a function to obtain normalizing constant is introduced. Forth, we discuss some computational issues. Finally, some future work is outlined.

1 Notation and Terminology

In this section we introduce concepts and definitions involved in the discussions of the Potts models. The notations used are similar to those given in Winkler (2003). Based on that, several related functions in the package are introduced.

We consider the Potts model defined on a finite undirected graph. A graph describes a set of connections between objects. Each object is called a node or vertex. There are observations to characterize the properties of vertices. Let $\mathcal{V}$ be a finite set, the set of vertices; $\mathcal{V} = \{1, 2, , \ldots, N\}$, where N is the total number of vertices. For every vertex $i \in \mathcal{V}$, let z_i take values in a finite set of categories $\mathcal{Z} = \{1, 2, \ldots, k\}$, where k is the number of categories. In the package we use different colors to represent different categories and vertices from the same category are of the same color. The product $\mathbf{Z} = \mathcal{Z}^N$ is the space of configurations $\mathbf{z} = (z_i; i \in \mathcal{V})$. A strictly positive probability measure P on $\mathbf{Z}$ for every $\mathbf{z} \in \mathbf{Z}$ is called a *stochastic* or *random field*. Note that P has to be strictly positive on $\mathbf{Z}$ to satisfy the assumptions of the Hammersley-Clifford theorem; see Besag (1974) and Winkler (2003) for details.

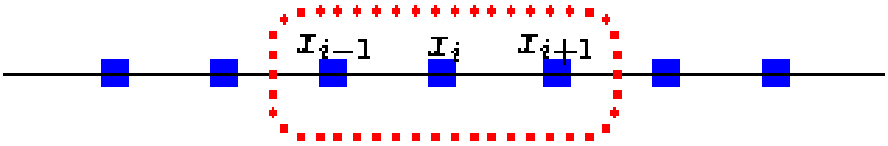


Figure 1: 2 neighbors in 1D

A collection $\partial = (\partial(v) : v \in \mathcal{V})$ of subsets of $\mathcal{V}$ is called a *neighborhood system*, if (i) $i \notin \partial(i)$ and (ii) $i \in \partial(j)$ if and only if $j \in \partial(i)$. The sites $j \in \partial(i)$ are called *neighbors* of i. We use $i \sim j$ to denote that i and j are neighbors of each other. There is an *edge* between i and j if and only if they are neighbors. Define a *graph* $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{E}$ is the set of edges. For a *finite undirected graph*, the number of vertices are finite and edge (i, j) is equivalent to edge (j, i). The function `getEdges()` can be used to get edges of a graph. When neighbors i and j are from the same category (of the same color), then there is a *bond* between them.

The random field P is a *Markov random field* (MRF) w.r.t. the neighborhood system ∂ if for all $\mathbf{z} \in \mathbf{Z}$,

$$P(z_i|z_j, j \neq i) = P(z_i|z_j, j \in \partial(i))$$

Probability measures of the form

$$P(\mathbf{z}) = \frac{\exp\{-H(\mathbf{z})\}}{\sum_{\mathbf{x}\in\mathbf{Z}} \exp\{-H(\mathbf{x})\}}$$

are called *Gibbs fields (or measures)*. H is called the *energy function* or *Hamiltonian*, and $\sum_{x\in\mathbf{Z}} \exp\{-H(x)\}$ is called the *partition function* or *normalizing constant*. For detailed account on MRF, Gibbs measures, and related issues, we refer to Winkler (2003).

When using Markov random field models, the first question is how to define neighbors of all vertices. For a 1D lattice, the simplest way to define neighbors is that every vertex (except those on the boundaries) has the two adjacent vertices as its neighbors, see Figure 1 for illustration. Of course, a vertex could have more than two neighbors.

For a 2D lattice, there are two common ways to define neighbors. One is that neighbors of a vertex comprise its available N, S, E, and W adjacencies. Another is that, besides those four, there are four diagonal adjacencies on its north-west, north-east, south-west, and south-east. See Figure 2 for illustrations. Probability measures defined on the former are called the first-order Markov random fields and the latter the second-order Markov random fields.

For a 3D lattice, besides defining six neighbors in the x, y, and z directions, one can add twelve diagonal neighbors in the $x - y$, $x - z$, and $y - z$ planes, and another eight on the 3D diagonals. This leads to a six neighbor structure, an eighteen neighbor structure, and a twenty-six neighbor structure. For illustration, see Figure 3.

The package provides a function called `getNeighbors()` to generate all neighbors of a graph.

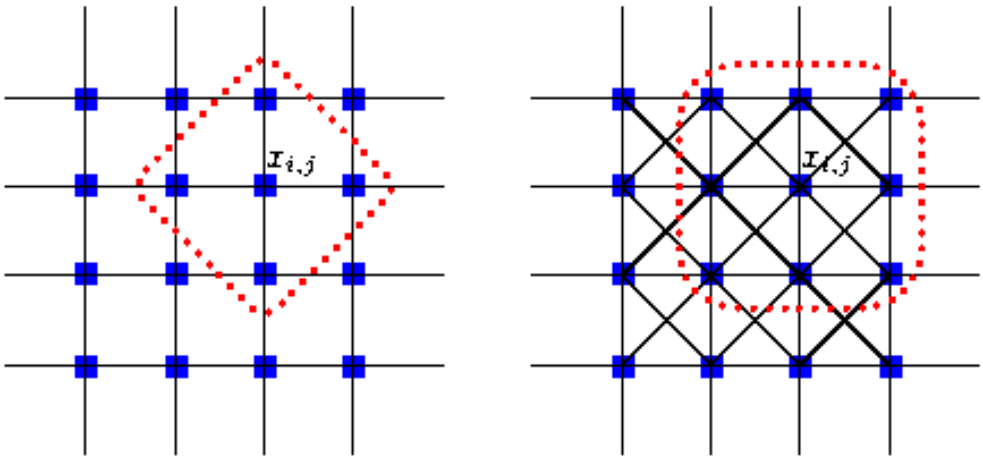
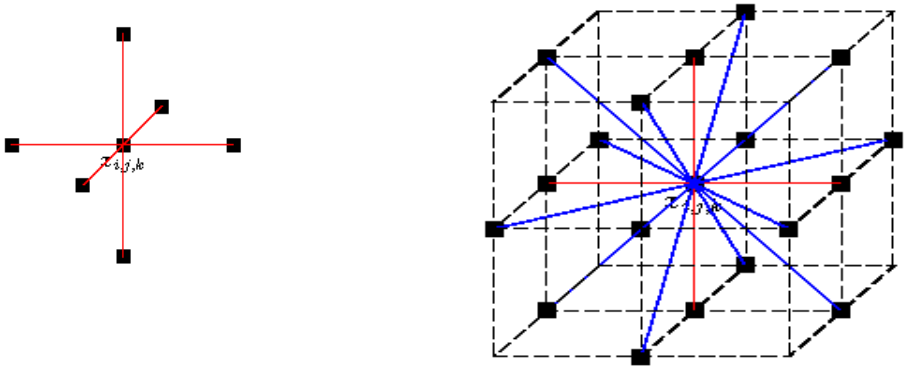
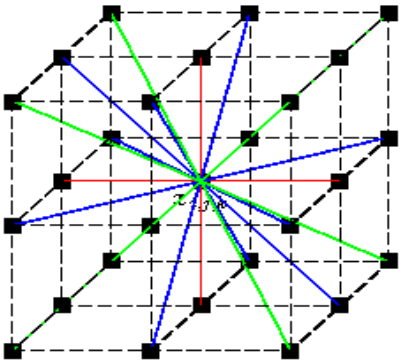


Figure 2: Four and eight neighbors in 2D



(a) six neighbors

(b) eighteen neighbors



(c) twenty-six neighbors

Figure 3: Illustration of neighbor structures in 3D

After defining neighbors, the second question is how to model the spatial relationship among neighbors. One choice is to use a model from the Potts model family, a set of MRF models with the Gibbs measure defined as follows.

$$p(\mathbf{z}) = C(\beta)^{-1} \exp\left\{ \sum_{i=1}^{N} \alpha_i(z_i) + \beta \sum_{i \sim j} w_{ij} f(z_i, z_j) \right\} \tag{1}$$

where $C(\beta)$ is a normalizing constant and $i \sim j$ indicates neighboring vertices. We need to define neighborhood structure and then assign relationships among neighboring vertices. The parameter β, called the *inverse temperature*, determines the level of spatial homogeneity between neighboring vertices in the graph. A zero β would imply that neighboring vertices are independent. We use positive β values. The w_{ij} are *weights* and we assume $w_{ij} > 0$. The term $\sum_{i=1}^{N} \alpha_i(z_i)$ is called the *external field*. The $\alpha_i(z_i)$ are functions of z_i. When $\beta = 0$, the external field completely characterizes the distribution of the independent $z_i, i = 1, 2, \ldots, N$.

When $f(z_i, z_j) = \mathrm{I}(z_i = z_j)$ model (1) becomes

$$p(\mathbf{z}) = C(\beta)^{-1} \exp\left\{ \sum_{i=1}^{N} \alpha_i(z_i) + \beta \sum_{i \sim j} \mathrm{I}(z_i = z_j) \right\} \tag{2}$$

For $k = 2$, this model is called the Ising model (Ising, 1925); for $k > 2$ it is the Potts (1953) model. The Ising model was originally proposed to describe the physical properties of magnets. Due to its flexibility and simplicity, the Ising model and its various versions have been widely used in other fields, such as brain models in cognitive science (Feng, 2008), information and machine learning theory (MacKay (2003) and references therein), economics (Bourgine and Nadal (2004) and references therein), sociology (Kohring, 1996) and game theory (Hauert and Szabó, 2005).

The most commonly used Potts model is the one without an external field and with $w_{ij} \equiv 1$,

$$p(\mathbf{z}) = C(\beta)^{-1} \exp\left\{ \beta \sum_{i \sim j} \mathrm{I}(z_i = z_j) \right\} \tag{3}$$

We refer to this as the *simple Potts model*.

Let $\alpha_i(z_i) \equiv 0$ and $f(z_i, z_j) = w_{ij}\mathrm{I}(z_i = z_j)$. Then (1) reduces to

$$p(\mathbf{z}) = C(\beta)^{-1} \exp\left\{ \beta \sum_{i \sim j} w_{ij}\mathrm{I}(z_i = z_j) \right\} \tag{4}$$

where w_{ij} is the weight between vertex i and j. For example we might take $w_{ij} = \frac{1}{d(z_i, z_j)}$ where $d(z_i, z_j)$ is a distance function, say Euclidean distance, between two vertices. This model is referred to as the *compound Potts model*.

In model (1), let $\alpha_i(z_i) = 0$ and define $f(z_i, z_j)$ as

$$f(z_i, z_j) = \begin{cases} a_1 & \text{if } z_i = z_j \\ a_2 & \text{if } |z_i - z_j| = 1 \\ a_3 & \text{otherwise} \end{cases} \tag{5}$$

where $a_1 \geq a_2 \geq a_3$. We call this model the *repulsion Potts model*. This model assumes an ordering of the colors and that neighboring vertices are most likely of the same color, and if they are different then it is more likely that they are similar than totally different. See Feng (2008) for more details.

2 Simulation of the Potts Models

It is very hard to find algorithms (such as inversion of CDF, rejection sampling, adaptive rejection sampling, or ratio-of-uniforms sampling) to generate i.i.d. samples from the Potts models, and Markov chain methods have to be used for the simulation.

To generate samples from model (1), single site updating, for example Gibbs sampling, is easy but may mix slowly. The Swendsen and Wang (1987) algorithm (SW) is widely used to generate random samples from the simple Potts model. Wolff's algorithm (Wolff, 1989) has been advocated as an alternative to the SW. A Gibbs sampler that takes advantage of the conditional independence structure to update variables $z_i, i = 1, 2, \ldots, N$, could make the simulation much faster than a single site updating scheme (Feng, 2008). When there is external field, the partial decoupling method might outperform the Gibbs sampling.

2.1 Swendsen-Wang Algorithm

The SW algorithm was originally proposed for the simulation of the simple Potts model. Drawing auxiliary variables $u_{ij}|\mathbf{z}$ for neighboring vertices (i, j) from independent and uniform distributions on $[0, \exp\{\beta \mathrm{I}(z_i = z_j)\}]$ makes the joint density

$$p(\mathbf{z}, \mathbf{u}) \propto \prod_{i \sim j} \mathrm{I}_{[0, \exp\{\beta \mathrm{I}(z_i = z_j)\}]}(u_{ij}) \tag{6}$$

The conditional distribution of $\mathbf{z}$ given $\mathbf{u}$ is also uniform on possible configurations. If $u_{ij} \geq 1$, then there is a bond between vertices i and j (when $u_{ij} \geq 1$ definitely $z_i = z_j$); otherwise there is no further constraint. Therefore, all vertices can be divided into patches (clusters). A patch is a collection of vertices, in which any two vertices are connected by a bond or a sequence of bonds. There is no bond between vertices from different patches. See Figure 4 for illustrations. The vertices within each patch should belong to the same category and the categories of different patches are i.i.d. from the discrete uniform distribution among all possible categories.

The SW algorithm is the seminal work on algorithms using auxiliary variables (in this case $\mathbf{u}$) in this area. A slightly generalized version of the original SW algorithm can be used to generate samples from a compound Potts model (Feng, 2008). Given all vertices, there are three steps in each iteration as follows.

1. Build bonds among vertices in neighbor with probability

$$1 - exp(-\beta w_{ij} \mathrm{I}(z_i = z_j))$$

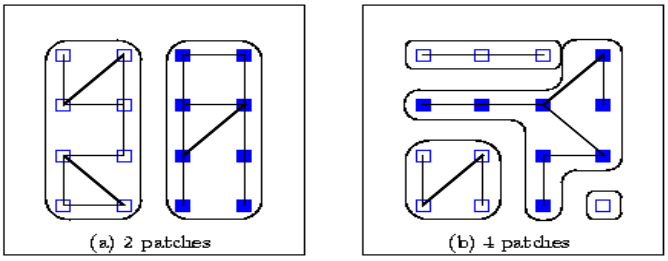


Figure 4: Illustration of the concept of patches.

2. Build patches for vertices bonded together

3. Flip the patches randomly to any color

The SW algorithm may outperform single-site updating samplers, especially when there is no external field and there is patchiness among vertices, in which simultaneous switching of clusters is necessary. For more details see Feng (2008).

2.2 Wolff's Algorithm

A modification of the Swendsen-Wang algorithm was proposed by Wolff (1989). The difference between the Wolff and the SW is that instead of flipping all patches randomly, one patch is chosen and all vertices in that patch are flipped to their opposites in the simple Potts model. To be more specific, for Wolff's algorithm, a vertex, say v, is selected randomly among all vertices, and bonds between v and its neighbors are set the same way as in the SW. If there are bonds between v and its neighbors, say C_v, then the bonds between vertices in C_v and their neighbors are set. Follow the same procedure recursively until no new bonds are created. Now there is a patch around v and all vertices in this patch are flipped to their opposites. Note, there is no randomness involved when flipping vertices for the Ising Model. Although Wolff's algorithm is similar to the SW, its proof is formalized from another perspective where detailed balance and irreducibility are verified (see Wolff (1989) for details).

To generate samples from a compound Potts model, a slighted generalized version of the original Wolff algorithm as follows can be adopted. There are four steps in each iteration.

1. Randomly select a vertex

2. Build a patch around it with probability

$$1 - exp(-\beta w_{ij} \mathrm{I}(z_i = z_j))$$

3. Continue building the patch till no additional vertex can be bonded together

4. Flip the patch randomly to another color

See Feng (2008) for more details.

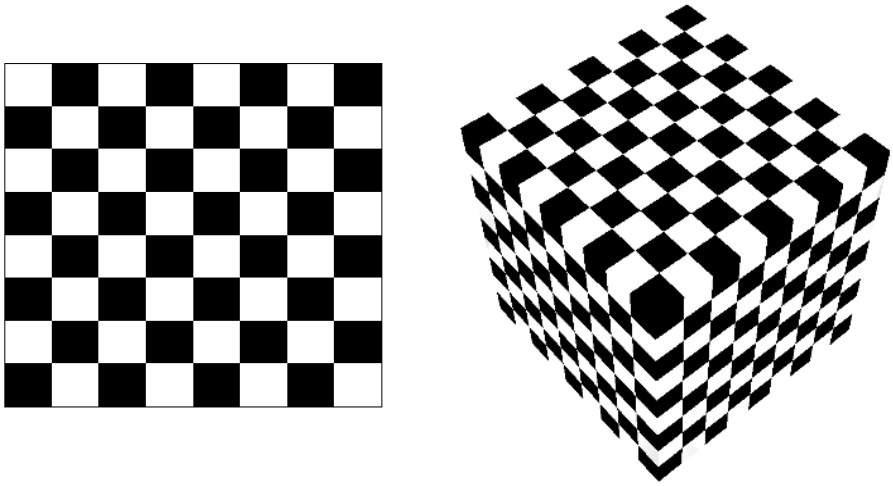
2.3 A Gibbs Sampler Using Conditional Independence

Various multiple-site sampling methods might outperform single-site updating, but sometimes they might not. Multi-site sampling methods like the SW algorithm could tackle the critical slowing down problem. When there is no external field, from the results pointed out in Higdon (1998), the SW algorithm outperforms single-site Metropolis updating when β is at the critical value. When there is an external field (a likelihood function in Bayesian inference for example), the SW slows down since it does not make good use of the data. Hurn (1997) and Smith and Smith (2006) suggested that when β is large, Gibbs samplers might be more effective. The choice of the best sampling method is likely to be problem-specific and there is no clear-cut winner as pointed out in Hurn (1997) and Higdon (1998). Furthermore, the previous studies just focus on relatively small grids in two dimensions with the number of categories equal to 2. What the convergence properties of various algorithms are on a three dimensional large grid with more than 2 categories, the case in MRI analysis for example (Feng, 2008), needs further study. Given that, a Gibbs sampler might be a good choice under certain circumstances. The package provides functions that uses the Gibbs sampler taking advantage of the conditional independence structure to update colors of vertices. It could make the simulation much faster than a one-site-after-another sampling scheme by using the vectorization functions in **R**.

The idea is that if we can divide variables that need to be updated into different blocks and given the variables in other blocks, all the variables within the same block are conditionally independent, then we can update all blocks iteratively with the variables within the same block being updated simultaneously. In Figure (a) of 5, under the four neighbor structure in 2D, given the black vertices, the whites are independent and vice versa. By this kind of independence, updating can be done in two steps: one for the blacks, one for the whites. The idea of taking advantage of this kind of independence can be traced back at least to the "Coding Methods" in Besag (1974). It was described in Wilkinson (2005) and detailed discussion can be found in Winkler (2003). This conditional independence can be generalized to 3D lattices with a six neighbor structure, see (b) of Figure 5 for illustration. Under six neighbor structure, given the blacks, the whites are independent and vice versa. The minimum number of blocks to make the vertices within each block independent given the other blocks is called the *chromatic number* in Winkler (2003). So the chromatic numbers for four neighbor configuration in 2D and six neighbor in 3D are both 2.

The extension to the eight neighbor configuration in 2D and eighteen and twenty-six neighbor in 3D is as follows. Under the eight neighbor structure in 2D, the chromatic number is four; under the eighteen neighbor configuration in 3D, the chromatic number is seven; under the twenty-six neighbor configuration in 3D, the chromatic number is eight. For more details, see Feng (2008).

The function to split vertices into conditional independent blocks is `getBlocks()`. Right



(a) 2D (b) 3D

Figure 5: Illustration of Conditional Independence

now, for a 2D graph, the vertices can be divided into either 2 or 4 blocks, and for a 3D graph either 2 or 8. The functions using the Gibbs sampler that takes advantage of conditional independence are `BlocksGibbs()` and `rPotts1()`. The difference between the two is that the first one can only generate samples from a Potts model without the external field while the second is good for all Potts models. The relationship among neighboring vertices can be defined by specifying the parameter `spatialMat`.

2.4 The Partial Decoupling Method

Besides the Gibbs sampling, another way to obtain samples from a Potts model with external field is to use the partial decoupling method through the function `rPotts1()`.

The partial decoupling method was originally developed by Higdon (1993) for an Ising model with an external field. Given the external field, the symmetry property is violated. Instead of drawing $u_{ij}|\mathbf{z}$ for neighbor points (i, j) from independent uniform distributions on $[0, \exp\{\beta \mathrm{I}(z_i = z_j)\}]$, draw $u_{ij}|\mathbf{z}$ from the uniform distribution $[0, \exp\{\delta_{ij}\beta \mathrm{I}(z_i = z_j)\}]$. Now, the bonding probability is not only controlled by β alone, but δ_{ij} as well. When $\delta_{ij} = 0$, it reduces to single site updating; when $\delta_{ij} = 1$, it corresponds to the SW algorithm. The smaller the δ_{ij}, the less likely bonds are formed. After setting the bonds, clusters are usually not independent anymore and Gibbs, Metropolis-Hastings, or even partial decoupling could be used to update the coarser model. The Gibbs sampling is used in the package.

The goal of choosing δ_{ij} is to improve the mixing when sampling from $\mathbf{z}|\mathbf{u}$. Strategies for the determination of $\{\delta_{ij}\}$ are based on the nature of the likelihood function of the data. For example, choosing $\delta_{ij} = 0$ for vertices on the boundaries of adjacent sub-graphs and $\delta_{ij} = 1$ for others can prevent the clusters from growing beyond certain limits. Another choice is

$\delta_{ij} = a\mathrm{I}\{y_i = y_j\}$ with $0 < a < 1$, where y_i and y_j are observations for vertex i and j. The details of the partial decoupling method can be found in Higdon (1998) and Hurn (1997).

3 Calculation of the Normalizing Constant

The normalizing constant is not critical and can be ignored if β is known. However, if β is treated as unknown, for example in studies using the Bayesian methods (Green and Richardson, 2002; Smith and Smith, 2006), then the normalizing constant is necessary when drawing samples from the conditional distribution of β by a Metropolis-Hastings algorithm.

The package provide a function `getNC()` to obtain the normalizing constant of a simple Potts model. The method adopted is similar to that in Green and Richardson (2002) and Smith and Smith (2006), in which the thermodynamic integration approach was used. The thermodynamic integration approach comes from the differential equations for describing thermodynamic relationships in physics. Basically, the thermodynamic integration method uses the fact that the normalizing constant can be obtained by solving the differential equation

$$\frac{\partial}{\partial\beta}\log(C(\beta)) = E(U(\mathbf{z})|\beta, k)$$

where $U(\mathbf{z}) = \sum_{i\sim j}\mathrm{I}(z_i = z_j)$ and k is the number of categories. Since

$$\log C(0) = N\log k$$

it follows

$$\log C(\beta) = N\log k + \int_0^\beta E(U(\mathbf{z})|\beta', k)d\beta' \tag{7}$$

In order to compute $\log C(\beta)$ we use the following steps:

1. Take k grid values $\{0 < \beta_1 < \beta_2 < \ldots < \beta_k \leq \beta\}$. For each $\beta_i, i = 1, 2, \ldots, k$, obtain n simulations of the graph and estimate $E(U(\mathbf{z})|\beta_i, k)$ by the average of the functions $U(\mathbf{z})$ from n simulations. Possible methods for simulation include Gibbs sampling with single site updating, or the SW, or the Wolff algorithms discussed previously.

2. Compute the integral by numerical integration.

4 Computational Issues

As mentioned in sub-section 2.3, it is beneficial to use vectorization functions in **R** to fulfill the idea of conditional independence. Furthermore, for the SW algorithm the major computational work is the identification and labeling of the patches of connected vertices. This is an instance of a connected component labeling problem for an undirected graph. We use Rem's algorithm (Dijkstra, 1976) to fulfill the task in the function `getPatches()`. Besides facilitating the SW algorithm, it can be used in other clustering methods as well. See Heller et al. (2006) for example.

5 Future Work

In the future, the package might be upgraded in several directions. First we might incorporate more available sampling methods for the Potts models. Second, the speed of current functions might be enhanced by using embedded **C** functions and parallel computations. Third, changes based on feedback of users.

References

Julian Besag. Spatial interaction and the statistical analysis of lattice systems (with discussion). *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):192–236, 1974.

Paul Bourgine and Jean Pierre Nadal, editors. *Cognitive Economics-and interdisciplinary approach*. Springer, 2004.

Edsger W. Dijkstra. *A Discipline of Programming*. Englewood Cliffs, New Jersey : Prentice-Hall, Inc, 1976.

Dai Feng. *Bayesian Hidden Markov Normal Mixture Models with Application to MRI Tissue Classification*. PhD thesis, The University of Iowa, 2008.

Peter J. Green and Sylvia Richardson. Hidden markov models and disease mapping. *Journal of the American Statistical Association*, 97:1055–1070, Dec. 2002.

Christoph Hauert and György Szabó. Game theory and physics. *American Journal of Physics*, 73(5):405–414, May 2005.

Ruth Heller, Damian Stanley, Daniel Yekutieli, Nava Rubin, and Yoav Benjamini. Cluster-based analysis of fmri data. *NeuroImage*, 33:599–608, 2006.

David M. Higdon. Comments on "Spatial Statistics and Bayesian Computation". *Journal of the Royal Statistical Society Series B-Statistical Methodology*, 55(1):78, 1993.

David M. Higdon. Auxiliary variable methods for Markov chain Monte Carlo with applications. *Journal of the American Statistical Association*, 93:585–595, 1998.

Merrilee Hurn. Difficulties in the use of auxiliary variables in Markov chain Monte Carlo methods. *Statics and Computing*, 7:35–44, 1997.

Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31:253–258, 1925.

G. A. Kohring. Ising models of social impact: the role of cumulative advantage. *Journal de Physique I*, 6(2):301–308, February 1996.

David J.C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

R. B. Potts. Some generalized order-disorder transformations. In *Cambridge Philosophic Society*, volume 48, pages 106–109, 1953.

Daniel Smith and Michael Smith. Estimation of binary Markov random fields using Markov chain Monte Carlo. *Journal of Computational and Graphical Statistics*, 15(1):207–227, March 2006.

Robert H. Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Physical Review Letters*, 58(2):86–88, 1987.

Darren J. Wilkinson. Parallel bayesian computation. In E. J. Kontoghiorghes, editor, *Handbook of Parallel Computing and Statistics*, pages 481–512. Marcel Dekker/CRC Press, 2005.

Gerhard Winkler. *Image Analysis, Random Fields and Dynamic Monte Carlo Methods: A Mathematical Introduction*. Springer-Verlag, second edition, 2003.

Ulli Wolff. Collective monte carlo updating for spin systems. *Physical Review Letters*, 62(4): 361–364, 1989.