Package: PottsUtils (via r-universe)

October 29, 2026

Title Utility Functions of the Potts Models

Version 0.3-3.1

Description There are three sets of functions. The first produces basic properties of a graph and generates samples from multinomial distributions to facilitate the simulation functions (they maybe used for other purposes as well). The second provides various simulation functions for a Potts model in Potts, R. B. (1952) <doi:10.1017/S0305004100027419>. The third currently includes only one function which computes the normalizing constant of a Potts model based on simulation results.

Depends R (>= 3.0.2)

Imports miscF (>= 0.1-4)

License GPL-2

NeedsCompilation yes

Author Dai Feng [aut, cre], Luke Tierney [ctb]

Maintainer Dai Feng <daifeng.stat@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2024-11-11 15:14:24 UTC

RemoteUrl https://github.com/cran/PottsUtils

RemoteRef HEAD

RemoteSha 71cee0b32271f4bbde1fd01fb4f60e9ca180592f

Contents

BlocksGibbs 2
getBlocks 3
getConfs 4
getEdges 5
getNC 6
getNeighbors 8

getPatches 10
getWeights 11
rPotts1 12
SW 15
Wolff 16

Index **18**

`BlocksGibbs` *Generate Random Samples from a Potts Model Using the Checkerboard Idea*

Description

Generate random samples from a Potts model by Gibbs Sampling that takes advantage of conditional independence.

Usage

```
BlocksGibbs(n, nvertex, ncolor, neighbors, blocks,
            weights=1, spatialMat=NULL, beta)
```

Arguments

`n`	number of samples.
`nvertex`	number of vertices in a graph.
`ncolor`	number of colors each vertex can take.
`neighbors`	a matrix of all neighbors in a graph, one row per vertex.
`blocks`	a list of blocks of vertices in a graph.
`weights`	weights between neighbors. One for each corresponding neighbor in `neighbors`. The default values are 1s for all.
`spatialMat`	the matrix that describes the relationship among vertices in neighbor. The default value is `NULL` corresponding to the simple or compound Potts model.
`beta`	the parameter inverse temperature of the Potts model.

Details

We use the Gibbs algorithm that takes advantage of conditional independence to speed up the generation of random samples from a Potts model. 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.

The `spatialMat` is the argument used to specify the relationship among vertices in neighbor. See `rPotts1` for more information on the Potts model and `spatialMat`.

Value

The output is a nvertex by n matrix with the kth column being the kth sample.

References

Dai Feng (2008) Bayesian Hidden Markov Normal Mixture Models with Application to MRI Tissue Classification *Ph. D. Dissertation, The University of Iowa*

See Also

Wolff, SW

Examples

```
#Example 1: Generate 100 samples from a repulsion Potts model with the
#           neighborhood structure corresponding to a first-order
#           Markov random field defined on a 3*3 2D graph.
#           The number of colors is 3 and beta=0.1,a_1=2,a_2=1,a_3=0.
#           All weights are equal to 1.

neighbors <- getNeighbors(mask=matrix(1, 3, 3), neiStruc=c(2,2,0,0))
blocks <- getBlocks(mask=matrix(1, 3, 3), nblock=2)
spatialMat <- matrix(c(2,1,0, 1,2,1,0,1,2), ncol=3)
BlocksGibbs(n=100, nvertex=9, ncolor=3, neighbors=neighbors, blocks=blocks,
            spatialMat=spatialMat, beta=0.1)
```

---

getBlocks *Get Blocks of a Graph*

Description

Obtain blocks of vertices of a 1D, 2D, or 3D graph, in order to use the conditional independence to speed up the simulation (checkerboard idea).

Usage

```
getBlocks(mask, nblock)
```

Arguments

mask	a vector, matrix, or 3D array specifying vertices of a graph. Vertices of value 1 are within the graph and 0 are not.
nblock	a scalar specifying the number of blocks. For a 2D graph nblock could be either 2 or 4, and for a 3D graph nblock could be either 2 or 8.

Details

The vertices within each block are mutually independent given the vertices in other blocks. Some blocks could be empty.

Value

A list with the number of components equal to nblock. Each component consists of vertices within the same block.

References

Darren J. Wilkinson Parallel Bayesian Computation *Handbook of Parallel Computing and Statistics* 481-512 Marcel Dekker/CRC Press 2005

Examples

```
#Example 1: split a line into 2 blocks
getBlocks(mask=c(1,1,1,1,0,0,1,1,0), nblock=2)

#Example 2: split a 4*4 2D graph into 4 blocks in order
#           to use the checkerboard idea for a neighborhood structure
#           corresponding to the second-order Markov random field.
getBlocks(mask=matrix(1, nrow=4, ncol=4), nblock=4)

#Example 3: split a 3*3*3 3D graph into 8 blocks
#           in order to use the checkerboard idea for a neighborhood
#           structure based on the 18 neighbors definition, where the
#           neighbors of a vertex comprise its available
#           adjacencies sharing the same edges or faces.
mask <- array(1, dim=rep(3,3))
getBlocks(mask, nblock=8)
```

---

getConfs *Generate Configurations of a Graph*

Description

Using recursive method to generate all possible configurations of a graph.

Usage

```
getConfs(nvertex, ncolor)
```

Arguments

nvertex	number of vertices in a graph.
ncolor	number of colors each vertex can take.

Details

Suppose there are n vertices and each can take values from $1, 2, \ldots, ncolor$. This function generates all possible configurations. For example, if there are two vertices and each can be either 1 or 2, then the possible configurations are (1,1), (1,2), (2,1) and (2,2).

Value

A matrix of all possible configurations. Each column corresponds to one configuration.

Examples

```
#Example 1: There are two vertices and each is either of
#           color 1 or 2.
getConfs(2,2)
```

`getEdges` *Get Edges of a Graph*

Description

Obtain edges of a 1D, 2D, or 3D graph based on the neighborhood structure.

Usage

```
getEdges(mask, neiStruc)
```

Arguments

`mask`	a vector, matrix, or 3D array specifying vertices of a graph. Vertices of value 1 are within the graph and 0 are not.
`neiStruc`	a scalar, vector of four components, or 3×4 matrix corresponding to 1D, 2D, or 3D graphs. It specifies the neighborhood structure. See `getNeighbors` for details.

Details

There could be more than one way to define the same 3D neighborhood structure for a graph (see Example 4 for illustration).

Value

A matrix of two columns with one edge per row. The edges connecting vertices and their corresponding first neighbors are listed first, and then those corresponding to the second neighbors, and so on and so forth. The order of neighbors is the same as in `getNeighbors`.

References

Gerhard Winkler (1995) Image Analysis, Random Fields and Dynamic Monte Carlo Methods *Springer-Verlag*

Dai Feng (2008) Bayesian Hidden Markov Normal Mixture Models with Application to MRI Tissue Classification *Ph. D. Dissertation, The University of Iowa*

Examples

```
#Example 1: get all edges of a 1D graph.
mask <- c(0,0,rep(1,4),0,1,1,0,0)
getEdges(mask, neiStruc=2)

#Example 2: get all edges of a 2D graph based on neighborhood structure
#           corresponding to the first-order Markov random field.
mask <- matrix(1 ,nrow=2, ncol=3)
getEdges(mask, neiStruc=c(2,2,0,0))

#Example 3: get all edges of a 2D graph based on neighborhood structure
#           corresponding to the second-order Markov random field.
mask <- matrix(1 ,nrow=3, ncol=3)
getEdges(mask, neiStruc=c(2,2,2,2))

#Example 4: get all edges of a 3D graph based on 6 neighbors structure
#           where the neighbors of a vertex comprise its available
#           N,S,E,W, upper and lower adjacencies. To achieve it, there
#           are several ways, including the two below.
mask <- array(1, dim=rep(3,3))
n61 <- matrix(c(2,2,0,0,
                0,2,0,0,
                0,0,0,0), nrow=3, byrow=TRUE)
n62 <- matrix(c(2,0,0,0,
                0,2,0,0,
                2,0,0,0), nrow=3, byrow=TRUE)
e1 <- getEdges(mask, neiStruc=n61)
e2 <- getEdges(mask, neiStruc=n62)
e1 <- e1[order(e1[,1], e1[,2]),]
e2 <- e2[order(e2[,1], e2[,2]),]
all(e1==e2)

#Example 5: get all edges of a 3D graph based on 18 neighbors structure
#           where the neighbors of a vertex comprise its available
#           adjacencies sharing the same edges or faces.
#           To achieve it, there are several ways, including the one below.

n18 <- matrix(c(2,2,2,2,
                0,2,2,2,
                0,0,2,2), nrow=3, byrow=TRUE)
mask <- array(1, dim=rep(3,3))
getEdges(mask, neiStruc=n18)
```

---

getNC *Calculate the Normalizing Constant of a Simple Potts Model*

Description

Use the thermodynamic integration approach to calculate the normalizing constant of a Simple Potts Model.

Usage

```
getNC(beta, subbetas, nvertex, ncolor,
      edges, neighbors=NULL, blocks=NULL,
      algorithm=c("SwendsenWang", "Gibbs", "Wolff"), n, burn)
```

Arguments

beta	the inverse temperature parameter of the Potts model.
subbetas	vector of betas used for the integration.
nvertex	number of vertices in a graph.
ncolor	number of colors each vertex can take.
edges	all edges in a graph.
neighbors	all neighbors in a graph. The default is NULL. If the sampling algorithm is "BlocksGibbs" or "Wolff", then this has to be specified.
blocks	the blocks of vertices of a graph. The default is NULL. If the sampling algorithm is "BlocksGibbs", then this has to be specified.
algorithm	a character string specifying the algorithm used to generate samples. It must be one of "SwendsenWang", "Gibbs", or "Wolff" and may be abbreviated. The default is "SwendsenWang".
n	number of iterations.
burn	number of burn-in.

Details

Use the thermodynamic integration approach to calculate the normalizing constant from a simple Potts model. See rPotts1 for more information on the simple Potts model.

By the thermodynamic integration method,

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

where N is the total number of vertices (nvertex), k is the number of colors (ncolor), and $U(\mathbf{z}) = \sum_{i \sim j} \mathrm{I}(z_i = z_j)$. Calculate $E(U(\mathbf{z})$ for subbetas based on samples, and then compute the integral by numerical integration.

Value

The corresponding normalizing constant.

References

Peter J. Green and Sylvia Richardson (2002) Hidden Markov Models and Disease Mapping *Journal of the American Statistical Association* **vol. 97, no. 460**, 1055-1070

See Also

BlocksGibbs, SW, Wolff

Examples

```
## Not run:
#Example 1: Calculate the normalizing constant of a simple Potts model
#           with the neighborhood structure corresponding to a
#           first-order Markov random field defined on a
#           3*3 2D graph. The number of colors is 2 and beta=2.
#           Use 11 subbetas evenly distributed between 0 and 2.
#           The sampling algorithm is Swendsen-Wang with 10000
#           iterations and 1000 burn-in.

edges <- getEdges(mask=matrix(1,3,3), neiStruc=c(2,2,0,0))
getNC(beta=2, subbetas=seq(0,2,by=0.2), nvertex=3*3, ncolor=2,
      edges, algorithm="S", n=10000, burn=1000)

## End(Not run)
```

---

getNeighbors *Get Neighbors of All Vertices of a Graph*

Description

Obtain neighbors of vertices of a 1D, 2D, or 3D graph.

Usage

```
getNeighbors(mask, neiStruc)
```

Arguments

mask	a vector, matrix, or 3D array specifying vertices within a graph. Vertices of value 1 are within the graph and 0 are not.
neiStruc	a scalar, vector of four components, or 3×4 matrix corresponding to 1D, 2D, or 3D graphs. It gives the definition of neighbors of a graph. All components of neiStruc should be positive (≥ 0) even numbers. For 1D graphs, neiStruc gives the number of neighbors of each vertex. For 2D graphs, neiStruc[1] specifies the number of neighbors on vertical direction, neiStruc[2] horizontal direction, neiStruc[3] north-west (NW) to south-east (SE) diagonal direction, and neiStruc[4] south-west (SW) to north-east (NE) diagonal direction. For 3D graphs, the first row of neiStruc specifies the number of neighbors on vertical direction, horizontal direction and two diagonal directions from the 1-2 perspective, the second row the 1-3 perspective, and the third row the 2-3 perspective. The index to perspectives is represented with the leftmost subscript of the array being the smallest.

Details

There could be more than one way to define the same 3D neighborhood structure for a graph (see Example 3 for illustration).

Value

A matrix with each row giving the neighbors of a vertex. The number of the rows is equal to the number of vertices within the graph and the number or columns is the number of neighbors of each vertex.

For a 1D graph, if each vertex has two neighbors, The first column are the neighbors on the left-hand side of corresponding vertices and the second column the right-hand side. For the vertices on boundaries, missing neighbors are represented by the number of vertices within a graph plus 1. When `neiStruc` is bigger than 2, The first two columns are the same as when `neiStruc` is equal to 2; the third column are the neighbors on the left-hand side of the vertices on the first column; the forth column are the neighbors on the right-hand side of the vertices on the second column, and so on and so forth. And again for the vertices on boundaries, their missing neighbors are represented by the number of vertices within a graph plus 1.

For a 2D graph, the index to vertices is column-wised. For each vertex, the order of neighbors are as follows. First are those on the vertical direction, second the horizontal direction, third the NW to SE diagonal direction, and forth the SW to NE diagonal direction. For each direction, the neighbors of every vertex are arranged in the same way as in a 1D graph.

For a 3D graph, the index to vertices is that the leftmost subscript of the array moves the fastest. For each vertex, the neighbors from the 1-2 perspective appear first and then the 1-3 perspective and finally the 2-3 perspective. For each perspective, the neighbors are arranged in the same way as in a 2D graph.

References

Gerhard Winkler (1995) Image Analysis, Random Fields and Dynamic Monte Carlo Methods *Springer-Verlag*

Dai Feng (2008) Bayesian Hidden Markov Normal Mixture Models with Application to MRI Tissue Classification *Ph. D. Dissertation, The University of Iowa*

Examples

```
#Example 1: get all neighbors of a 1D graph.
mask <- c(0,0,rep(1,4),0,1,1,0,0,1,1,1)
getNeighbors(mask, neiStruc=2)

#Example 2: get all neighbors of a 2D graph based on neighborhood structure
#           corresponding to the second-order Markov random field.
mask <- matrix(1, nrow=2, ncol=3)
getNeighbors(mask, neiStruc=c(2,2,2,2))

#Example 3: get all neighbors of a 3D graph based on 6 neighbors structure
#           where the neighbors of a vertex comprise its available
#           N,S,E,W, upper and lower adjacencies. To achieve it, there
#           are several ways, including the two below.
```

```
mask <- array(1, dim=rep(3,3))
n61 <- matrix(c(2,2,0,0,
                0,2,0,0,
                0,0,0,0), nrow=3, byrow=TRUE)
n62 <- matrix(c(2,0,0,0,
                0,2,0,0,
                2,0,0,0), nrow=3, byrow=TRUE)
n1 <- getNeighbors(mask, neiStruc=n61)
n2 <- getNeighbors(mask, neiStruc=n62)
n1 <- apply(n1, 1, sort)
n2 <- apply(n2, 1, sort)
all(n1==n2)

#Example 4: get all neighbors of a 3D graph based on 18 neighbors structure
#           where the neighbors of a vertex comprise its available
#           adjacencies sharing the same edges or faces.
#           To achieve it, there are several ways, including the one below.

n18 <- matrix(c(2,2,2,2,
                0,2,2,2,
                0,0,2,2), nrow=3, byrow=TRUE)
mask <- array(1, dim=rep(3,3))
getNeighbors(mask, neiStruc=n18)
```

`getPatches` *Get Patches of a Graph*

Description

Obtain patches of a graph by Rem's algorithm.

Usage

```
getPatches(bonds, nvertex)
```

Arguments

`bonds`	a matrix of bonds in a graph, with one bond per row.
`nvertex`	number of vertices in a graph.

Details

Given all bonds and the number of vertices in a graph, this function provides all patches.

Value

A list comprises all patches in a graph. Each component of the list consists of vertices within one patch.

References

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

Examples

```
#Example 1: Find patches of a 3*3 2D graph with 6 bonds.

bonds <- matrix(c(1,2,2,5,5,6,3,6,5,8,7,8), ncol=2, byrow=TRUE)
getPatches(bonds, 9)
```

---

getWeights *Get All Weights of a Graph*

Description

Obtain weights of edges of a 1D, 2D, or 3D graph based on the neighborhood structure.

Usage

```
getWeights(mask, neiStruc, format=1)
```

Arguments

mask	a vector, matrix, or 3D array specifying vertices within a graph. Vertices of value 1 are within the graph and 0 are not.
neiStruc	a scalar, vector of four components, or 3×4 matrix corresponding to 1D, 2D, or 3D graphs. It specifies the neighborhood structure. See `getNeighbors` for details.
format	If it is 1, then the output is a vector of weights, one for two vertices in the corresponding output from `getEdges`. If it is 2, then the output is a matrix, one for two vertices in the corresponding output from `getNeighbors`. The default value is 1.

Details

The weights are equal to the reciprocals of the distance between neighboring vertices.

Value

A vector of weights, one component corresponding to an edge of a graph. Or a matrix of weights, one component corresponding to two vertices in neighbor.

Examples

```
#Example 1: get all weights of a 2D graph based on neighborhood structure
#           corresponding to the first-order Markov random field.
mask <- matrix(1 ,nrow=2, ncol=3)
getWeights(mask, neiStruc=c(2,2,0,0))

#Example 2: get all weights of a 2D graph based on neighborhood structure
#           corresponding to the second-order Markov random field.
#           Put the weights in a matrix form corresponding to
#           neighbors of vertices.
mask <- matrix(1 ,nrow=3, ncol=3)
getWeights(mask, neiStruc=c(2,2,2,2), format=2)

#Example 3: get all weights of a 3D graph based on 6 neighbors structure
#           where the neighbors of a vertex comprise its available
#           N,S,E,W, upper and lower adjacencies.
mask <- array(1, dim=rep(3,3))
n61 <- matrix(c(2,2,0,0,
                0,2,0,0,
                0,0,0,0), nrow=3, byrow=TRUE)
getWeights(mask, neiStruc=n61)
```

---

rPotts1 *Generate One Random Sample from a Potts Model*

Description

Generate one random sample from a Potts model with external field by Gibbs Sampling that takes advantage of conditional independence, or the partial decoupling method.

Usage

```
rPotts1(nvertex, ncolor, neighbors, blocks, edges=NULL, weights=1,
        spatialMat=NULL, beta, external, colors,
        algorithm=c("Gibbs", "PartialDecoupling"))
```

Arguments

nvertex	number of vertices in a graph.
ncolor	number of colors each vertex can take.
neighbors	all neighbors in a graph. It is not required when using the partial decoupling method.
blocks	the blocks of vertices in a graph. It is not required when using the partial decoupling method.
edges	all edges in a graph. The default value is NULL. It is not required when using Gibbs sampling.

`weights` weights between neighbors or δ_{ij}s in the partial decoupling method. When using Gibbs sampling, there is one for each corresponding component in `neighbors`. When using partial decoupling, there is one for each corresponding component in `edges`. The default values are 1s for all.

`spatialMat` a matrix that describes the relationship among vertices in neighbor. It is not required when using the partial decoupling method. The default value is `NULL` corresponding to the simple or compound Potts model.

`beta` the parameter inverse temperature of the Potts model.

`external` a matrix giving values of external field. The number of rows equal to `nvertex` and number of columns equal to `ncolor`.

`colors` the current colors of vertices.

`algorithm` a character string specifying the algorithm used to generate samples. It must be either "Gibbs", or "PartialDecoupling", and may be abbreviated. The default is "Gibbs".

Details

This function generates random samples from a Potts model as follows:

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

where $C(\beta)$ is a normalizing constant and $i \sim j$ indicates neighboring vertices. The parameter β is called the "inverse temperature", which determines the level of spatial homogeneity between neighboring vertices in the graph. We assume $\beta > 0$. The set $\mathbf{z} = \{z_1, z_2, \ldots, \}$ comprises the indices to the colors of all vertices. Function $f(z_i, z_j)$ determines the relationship among vertices in neighbor. Parameter w_{ij} is the weight between vertex i and j. The term $\sum_i \alpha_i(z_i)$ is called the "external field".

For the simple, the compound, and the simple repulsion Potts models, the external field is equal to 0. For the simple and the compound Potts model $f(z_i, z_j) = I(z_i = z_j)$. Parameters w_{ij} are all equal for the simple Potts model but not so for the compound model.

For the repulsion Potts model $f(z_i, z_j) = \beta_1$ if $z_i = z_j$; $f(z_i, z_j) = \beta_2$ if $|z_i - z_j| = 1$; $f(z_i, z_j) = \beta_3$ otherwise.

The argument `spatialMat` is used to specify the relationship among vertices in neighbor. The default value is `NULL` corresponding to the simple or the compound Potts model. The component at the ith row and jth column defining the relationship when the color of a vertex is i and the color of its neighbors is j. Besides the default setup, for the simple and the compound Potts models `spatailMat` could be an identity matrix also. For the repulsion Potts model, it is

$$\begin{pmatrix} a_1 & a_2 & a_3 & \ldots & a_3 \\ a_2 & a_1 & a_2 & \ldots & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_3 & a_3 & a_3 & \ldots & a_1 \end{pmatrix}$$

Other relationships among neighboring vertices can be specified through it as well.

Gibbs sampling can be used to generate samples from all kinds of Potts models. We use the method that takes advantage of conditional independence to speed up the simulation. See BlocksGibbs for details.

The partial decoupling method could be used to generate samples from the simple Potts model plus the external field. The δ_{ij}s are specified through the argument weights.

Value

The output is a vector with the kth component being the new color of vertex k.

References

Dai Feng (2008) Bayesian Hidden Markov Normal Mixture Models with Application to MRI Tissue Classification *Ph. D. Dissertation, The University of Iowa*

David M. Higdon (1998) Auxiliary variable methods for Markov Chain Monte Carlo with applications *Journal of the American Statistical Association* **vol. 93** 585-595

See Also

BlocksGibbs, Wolff SW

Examples

```
## Not run:
neighbors <- getNeighbors(matrix(1, 16, 16), c(2,2,0,0))
blocks <- getBlocks(matrix(1, 16, 16), 2)
spatialMat <- matrix(c(2, 0, -1, 0, 2, 0, -1, 0, 2), ncol=3)
mu <- c(22, 70 ,102)
sigma <- c(17, 16, 19)
count <- c(40, 140, 76)
y <- unlist(lapply(1:3, function(i) rnorm(count[i], mu[i], sigma[i])))
external <- do.call(cbind,
                    lapply(1:3, function(i) dnorm(y, mu[i],sigma[i])))
current.colors <- rep(1:3, count)
rPotts1(nvertex=16^2, ncolor=3, neighbors=neighbors, blocks=blocks,
        spatialMat=spatialMat, beta=0.3, external=external,
        colors=current.colors, algorithm="G")
edges <- getEdges(matrix(1, 16, 16), c(2,2,0,0))
rPotts1(nvertex=16^2, ncolor=3, edges=edges, beta=0.3,
        external=external, colors=current.colors, algorithm="P")

## End(Not run)
```

SW *Generate Random Samples from a Compound Potts Model by the Swendsen-Wang Algorithm*

Description

Generate random samples from a compound Potts model using the Swendsen-Wang algorithm.

Usage

```
SW(n, nvertex, ncolor, edges, weights, beta)
```

Arguments

n	number of samples.
nvertex	number of vertices of a graph.
ncolor	number of colors each vertex can take.
edges	edges of a graph.
weights	weights of edges. One for each corresponding component in edges. The default values are 1s for all.
beta	the parameter inverse temperature of the Potts model.

Details

We use the Swendsen-Wang algorithm to generate random samples from a compound Potts model. See rPotts1 for more information on the compound Potts model.

Value

The output is a nvertex by n matrix with the kth column being the kth sample.

References

Robert H. Swendsen and Jian-Sheng Wang (1987) Nonuniversal Critical Dynamics in Monte Carlo Simulations *Physical Review Letters* **vol. 58, no. 2**, 86-88

Dai Feng (2008) Bayesian Hidden Markov Normal Mixture Models with Application to MRI Tissue Classification *Ph. D. Dissertation, The University of Iowa*

See Also

Wolff, BlocksGibbs

Examples

```
#Example 1: Generate 100 samples from a Potts model with the
#           neighborhood structure corresponding to a
#           second-order Markov random field defined on a
#           3*3 2D graph. The number of colors is 2.
#           beta=0.1. All weights are equal to 1.

edges <- getEdges(mask=matrix(1, 2, 2), neiStruc=rep(2,4))
set.seed(100)
SW(n=500, nvertex=4, ncolor=2, edges, beta=0.8)
```

`Wolff` *Generate Random Samples from a Compound Potts Model by the Wolff Algorithm*

Description

Generate random samples from a compound Potts model using the Wolff Algorithm.

Usage

```
Wolff(n, nvertex, ncolor, neighbors, weights, beta)
```

Arguments

`n`	number of samples.
`nvertex`	number of vertices of a graph.
`ncolor`	number of colors each vertex can take.
`neighbors`	neighbors of a graph.
`weights`	weights between neighbors. One for each corresponding component in `neighbors`. The default values are 1s for all.
`beta`	the parameter inverse temperature of the Potts model.

Details

We use the Wolff algorithm to generate random samples from a compound Potts model. See `rPotts1` for more information on the compound Potts model.

Value

A `nvertex` by `n` matrix with the kth column being the kth sample.

References

Ulli Wolff (1989) Collective Monte Carlo Updating for Spin Systems *Physical Review Letters* **vol. 62, no. 4**, 361-364

Dai Feng (2008) Bayesian Hidden Markov Normal Mixture Models with Application to MRI Tissue Classification *Ph. D. Dissertation, The University of Iowa*

See Also

SW, BlocksGibbs

Examples

```
#Example 1: Generate 100 samples from a Potts model with the
#           neighborhood structure corresponding to a
#           second-order Markov random field defined on a
#           3*3 2D graph. The number of colors is 2.
#           beta=0.7. All weights are equal to 1.

neighbors <- getNeighbors(mask=matrix(1, 3, 3), neiStruc=rep(2,4))
Wolff(n=100, nvertex=9, ncolor=2, neighbors, beta=0.7)
```

Index

∗ **distribution**
BlocksGibbs, 2
getNC, 6
rPotts1, 12
SW, 15
Wolff, 16
∗ **math**
getConfs, 4
∗ **spatial**
getBlocks, 3
getEdges, 5
getNeighbors, 8
getPatches, 10
getWeights, 11

BlocksGibbs, 2, *8*, *14, 15*, *17*

getBlocks, 3
getConfs, 4
getEdges, 5
getNC, 6
getNeighbors, 8
getPatches, 10
getWeights, 11

rPotts1, *2*, *7*, 12, *15, 16*

SW, *3*, *8*, *14*, 15, *17*

Wolff, *3*, *8*, *14, 15*, 16