ERNM Vignette

Ian Fellows

November 5, 2012

1 Introduction

A graph is a collection of nodes, each of which may either be connected or not connected to each other node. For the purposes of this document, nodes may not be connected to themselves. In addition to the graph connections, each node may have characteristics which are of interest to the researcher. A network is defined as the union of a graph and the nodal characteristics. Random graphs, where connections between nodes are random but nodal characteristics are either fixed or missing, have a long history in the mathematical literature starting with the simple Erdos-Renyi model (**?**), and including the more general exponential-family random graph models (ERGM) for which inference requires modern Markov Chain Monte Carlo (MCMC) methods (**??**). On the other hand we have Gibbs/Markov random field models where nodal attributes are random but interconnections between nodes are fixed. A simple example is the Ising model of ferromagnetism (**?**) from the statistical physics literature which is exactly solvable under certain network configurations (**?**); however, most field models require more complex methodologies for inference (**?**).

In the social network literature, these two classes of models are conceptually defined as "social selection" and "social influence" models. In social selection models, the probability of social ties between individuals are determined by nodal characteristics such as age or sex (see **?** and references therein). In social influence models, individuals' nodal characteristics are determined by social ties (see **?** and references therein). **?** argues that the processes of tie selection and nodal variate influence are co-occurring phenomena, with ties affecting nodal variates and visa versa, and should therefore be considered together. This chapter presents a joint exponential-family model of connections between nodes (dyads), and nodal attributes, thus representing a unification of social selection and influence. We will refer to this model as an exponential-family random network model (ERNM).

Let the graph Y be an n by n matrix whose entries $Y_{i,j}$ indicate whether subject i and j are connected, where n is the size of the population. Further let X be an $n \times q$ matrix of nodal variates. We define the network to be the random variable (Y, X).

ERGM:

$$P(Y=y|\eta, X=x) = \frac{1}{c(\eta,x)} e^{\eta \cdot g((y,x))+o((y,x))}$$

Gibbs/Markov random field:

$$P(X=x|\eta, Y=y) = \frac{1}{c(\eta,y)} e^{\eta \cdot g((y,x))+o((y,x))}$$

(NEW!!) Exponential-Family Random Network Model (ERNM):

$$P(Y=y, X=x|\eta) = \frac{1}{c(\eta)} e^{\eta \cdot g((y,x))+o((y,x))}$$

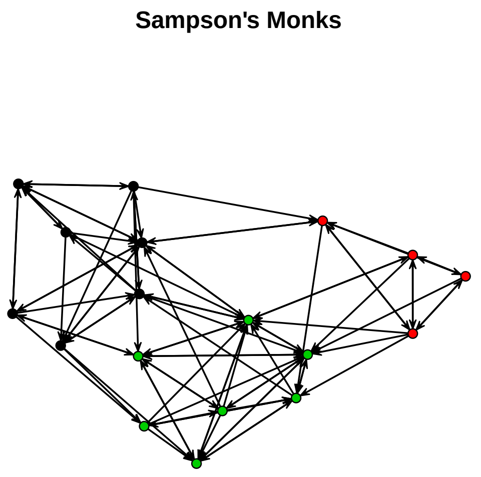
As ernm models subsume both ergm and Gibbs random field models, the ernm package can fit and simulate from both.

```
> set.seed(1)
> library(ernm)
> library(ergm)
```

2 ERGM

In this section we compare an ergm fit with the ergm package with an ergm model fit with ther ernm package.

```
> data(sampson)
> #fit a model with ergm
> ergmFit <- ergm(samplike ~ edges + asymmetric + match("group"))

Iteration 1 of at most 20:
Convergence test P-value: 9.7e-03
The log-likelihood improved by 0.001176
Iteration 2 of at most 20:
Convergence test P-value: 1.2e-01
The log-likelihood improved by 0.0005834
Iteration 3 of at most 20:
Convergence test P-value: 5.8e-01
Convergence detected. Stopping.
The log-likelihood improved by 0.0001227

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use t

> summary(ergmFit)

==========================
Summary of model fit
==========================
Formula:   samplike ~ edges + asymmetric + match("group")

Iterations:  20

Monte Carlo MLE Results:
                Estimate Std. Error MCMC % p-value
edges            -1.5493     0.2296      0 < 1e-04 ***
asymmetric       -0.7162     0.2404      0 0.00312 **
nodematch.group   2.0333     0.3125      0 < 1e-04 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

     Null Deviance: 424.2  on 306  degrees of freedom
 Residual Deviance: 692.5  on 303  degrees of freedom

AIC: 698.5    BIC: 709.7    (Smaller is better.)

> #fit a similar model with ernm
> net <- as.BinaryNet(samplike)
> plot(net,vertex.col=net[["group"]],main="Sampson's Monks")
> ernmFit <- ernm(net ~ edges() + reciprocity() + nodeMatch("group"),verbose=0)

calculating: .....

> summary(ernmFit)

                theta        se          z      p.value
edges       -2.218064 0.2215872 -10.009892 1.379027e-23
reciprocity  1.358206 0.4684402   2.899422 3.738514e-03
nodeMatch    2.000028 0.2995928   6.675820 2.458539e-11
```



We note a few things here. First, though the model formulas are similar, they are distinct, with ernm having its own statistics. The edges and match statistics are equivalent, but the ernm model counts symmetric ties, whereas the ergm package counts asymmetric ties. Which variates are considered random is specified after the | character. by default, only the graph is considered random. If the graph and a variable named ”var” is missing, you can use something like:

net $\sim$ edges() | var

if only var is random (fixed graph), then you can use noDyad

net $\sim$ edges() | noDyad + var

constraints and offsets are nested in a flag as in:

net $\sim$ edges() + constraint(boundedDegree(1,10))

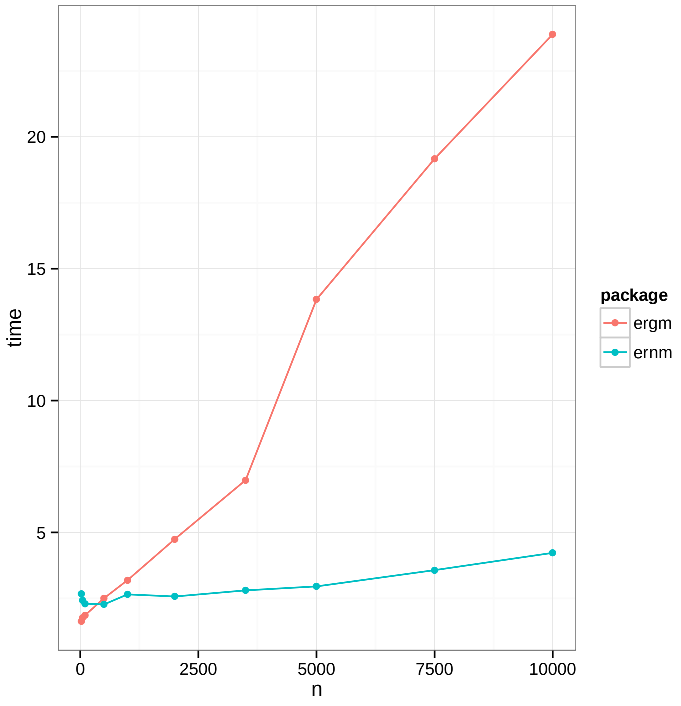
net $\sim$ edges() + offset(offsetName())

Secondly, a new network data structure is used (BinaryNet), which can be operated on both from C++ and R. More on this later.

2.1 ERGM Timings

It may be instructive to compare the speed of ergm v.s. ernm. To do this, we use a model with a term for edges and one for triangles. the parameter for the triangle term is set to 0, and the edges parameter is chosen such that the mean degree of the network is 10. Both packages are set to use Tie-Dyad toggling, with a burn in of 1000000, an interval of 1000, and a sample size of 1000.

```
> library(ggplot2)
> ns <- c(25,50,100,500,1000,2000,3500,5000,7500,10000)
> ergmTimings <- c()
> ernmTimings <- c()
> for(n in ns){
+   print(n)
+   p <- 10/(n-1)
+   theta <- c(log(p/(1-p)),0)
+
+   nw <- network.initialize(n, directed = FALSE)
+   net <- as.BinaryNet(nw)
+
+   sampler <- createCppSampler(net ~ edges() + triangles(),sampler="VertexMetropolis")
+   sampler$getModel()$setThetas(theta)
+
+   ernmTime <- system.time(sims1 <- sampler$generateSampleStatistics(1000000,1000,1000))
+
+   ergmTime <- system.time(sims <- simulate(nw ~ edges + triangle,nsim=1000,coef=theta,stat
+            control = control.simulate.formula(MCMC.burnin = 1000000, MCMC.interval = 1000)
+   ergmTimings <- c(ergmTimings,ergmTime[3])
+   ernmTimings <- c(ernmTimings,ernmTime[3])
+ }

[1] 25
[1] 50
[1] 100
[1] 500
[1] 1000
[1] 2000
[1] 3500
[1] 5000
[1] 7500
[1] 10000

> print(ergmTimings)

elapsed elapsed elapsed elapsed elapsed elapsed elapsed elapsed elapsed elapsed
  1.635   1.772   1.861   2.503   3.191   4.747   6.984  13.832  19.164  23.882

> print(ernmTimings)

elapsed elapsed elapsed elapsed elapsed elapsed elapsed elapsed elapsed elapsed
  2.681   2.426   2.303   2.270   2.652   2.572   2.804   2.955   3.571   4.226

> dat <- rbind(data.frame(time=ergmTimings,n=ns,package="ergm"),
+              data.frame(time=ernmTimings,n=ns,package="ernm"))
> p <- qplot(x=n,y=time,data=dat,color=package,geom="line")+geom_point()+ theme_bw()
> print(p)
```



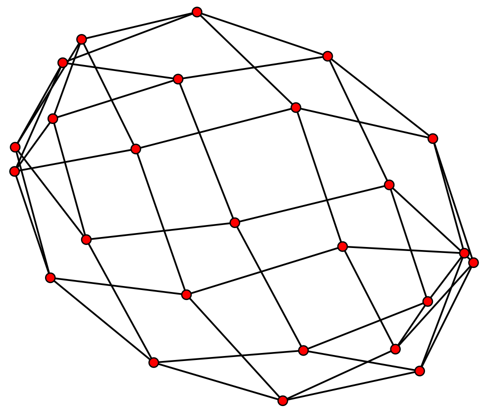
Note how the ergm simulation times explode up to around 30 as the number of nodes increases, whereas the ernm simulations remain under 5 seconds. This is because all of ernm's toggling / change statistics are constant time as n increases provided the average degree remains constant. These results represent a marked improvement for ergm, previous versions of ergm that I tested yielded results over 2 times as slow.

3 Gibbs-Markov Random Fields

If the graph is fixed and the node variable is binary, then we can write an ising model as

net ∼ nodeMatch(”var”) + nodeCount(”var”) | noDyad + var

```
> form <- latticeNet ~ nodeMatch("var") + nodeCount("var") | noDyad + var
> n <- 25
> nw <- network.initialize(n, directed = FALSE)
> #create a toroidal lattice
> latticeNet <- as.BinaryNet(nw)
> latticeNet[["var"]] <- sample(c("a","b"),n,replace=TRUE)
> nt <- function(i,j) sqrt(n)*(i-1) + (j-1) + 1
> for(node in 1:n){
+   i <- (node-1) %% sqrt(n) + 1
+   j <- floor((node-1)/sqrt(n)) + 1
+   latticeNet[node,nt(i,((j-2) %% sqrt(n)) + 1)] <- TRUE
+   latticeNet[node,nt(i,((j) %% sqrt(n)) + 1)] <- TRUE
+   latticeNet[nt( ((i-2) %% sqrt(n)) + 1 , j),node] <- TRUE
+   latticeNet[nt(((i) %% sqrt(n)) + 1 , j),node] <- TRUE
+ }
> plot(latticeNet)
```



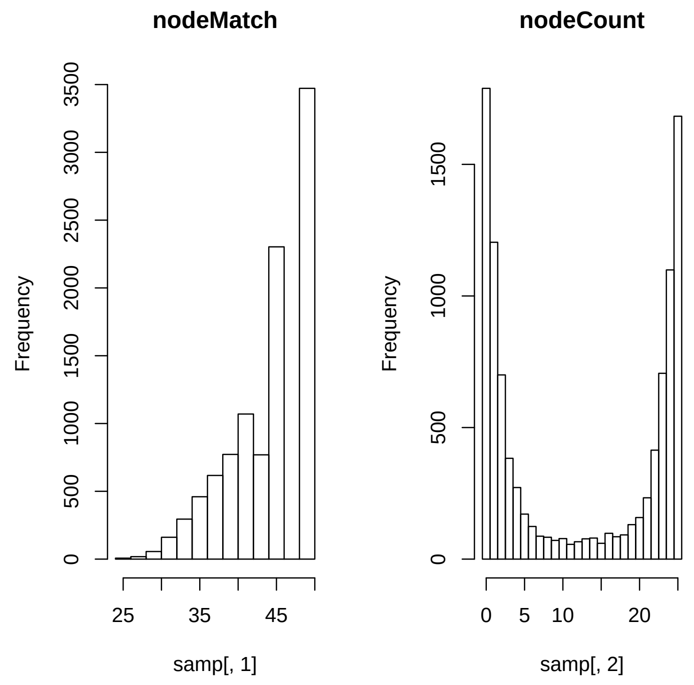
Here we will simulate a model right on the boarder of the phase transition. and fit a Gibbs field model to a point near the phase transition

```
> sampler <- createCppSampler(form,nodeSamplingPercentage=1)
> sampler$getModel()$setThetas(c(.9,0))
> samp <- sampler$generateSampleStatistics(1000,1000,10000)
> par(mfrow=c(1,2))
> hist(samp[,1],main="nodeMatch")
> hist(samp[,2],breaks=seq(from=-.5,to=25.5,length.out=27),main="nodeCount")
> colMeans(samp)

nodeMatch nodeCount
  44.6744   12.2569

> #fit the ernm
> fit <- ernm(form,meanStats=c(44.6216, 12.6480),mcmcInterval=1000)
```

```
sample statistics:
       means:
          nodeMatch nodeCount
simulated   25.0368   12.4504
observed    44.6216   12.6480
       std:
     nodeMatch nodeCount
[1,]   3.52759  2.514632



iteration: 1
log likelihood improved by:  9.351325
 maximum scaled gradient:  5.551893
 nodeMatch  nodeCount
0.58806027 0.02288115
sample statistics:
       means:
          nodeMatch nodeCount
simulated   34.5662   13.3854
observed    44.6216   12.6480
       std:
     nodeMatch nodeCount
[1,]    5.1472  6.086557



iteration: 2
log likelihood improved by:  1.530644
 maximum scaled gradient:  1.953567
   nodeMatch    nodeCount
 0.906488944 -0.002949878
sample statistics:
       means:
          nodeMatch nodeCount
simulated   44.7414   12.3713
observed    44.6216   12.6480
       std:
     nodeMatch nodeCount
[1,]  5.146394  10.76868



iteration: 3
log likelihood improved by:  0.0005988971
 maximum scaled gradient:  0.02569489
```

```
    nodeMatch     nodeCount
 0.9020091088 -0.0005553294
sample statistics:
       means:
          nodeMatch nodeCount
simulated   44.6294    12.253
observed    44.6216    12.648
       std:
     nodeMatch nodeCount
[1,]  5.227293  10.71441



iteration: 4
log likelihood improved by:  0.0006802448
 maximum scaled gradient:  0.03686622
```

```
> fit

                           ERNM Model
Domain:
 Random graph = FALSE, Random variables = var

             nodeMatch     nodeCount
Parameters   0.9020091 -0.0005553294
Mean Values 44.6294000 12.2530000000
```



4 Working with BinaryNets

BinaryNets are network data structures native to the ernm package. They are special in a couple ways. First, they have a sparse representation of missingness, such that a directed network where halve of the nodes have been egocentrically sampled takes the same amount of space a a simple fully observed network. Secondly, they may be passed up and down easily from R to C++ and visa versa. Finally, they are extensible on the C++ level, so that their implementation may be replaced. For example, it might be useful to put in a file backed storage system for large problems.

BinaryNets can be coerced to and from network objects.

```
> data(sampson)
> #coersion
> net <- as.BinaryNet(samplike)
> nw2 <- as.network(net)
> print(nw2)
```

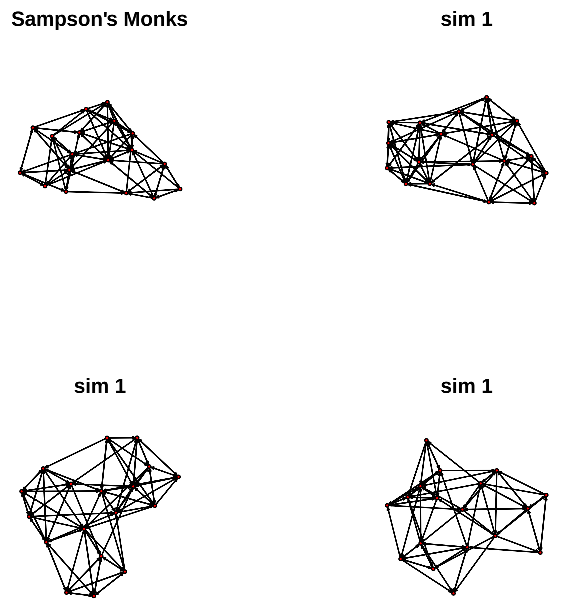
```
 Network attributes:
  vertices = 18
  directed = TRUE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 88
    missing edges= 0
    non-missing edges= 88

 Vertex attribute names:
    cloisterville group vertex.names

> #dyad Extraction
> net[1:2,1:5]

      [,1]  [,2] [,3]  [,4] [,5]
[1,] FALSE  TRUE TRUE FALSE TRUE
[2,] FALSE FALSE TRUE FALSE TRUE

> net$outNeighbors(c(1,2,3))

[[1]]
[1]  2  3  5  7 11 15

[[2]]
[1]  3  5  6  9 15

[[3]]
[1]  2  7  8 14

> #dyad assignment
> net[1,1:5] <- rep(NA,5)
> net[1:2,1:5]

      [,1]  [,2] [,3]  [,4] [,5]
[1,] FALSE    NA   NA    NA   NA
[2,] FALSE FALSE TRUE FALSE TRUE

> net[1:2,1:5,maskMissing=FALSE] #remove the mask over missing values and see nothing was re

      [,1]  [,2] [,3]  [,4] [,5]
[1,] FALSE  TRUE TRUE FALSE TRUE
[2,] FALSE FALSE TRUE FALSE TRUE

> #node variables
> net$variableNames()

$discrete
[1] "group"        "na"           "vertex.names"

$continuous
[1] "cloisterville"
```

```
> net[["group"]]

 [1] Loyal    Loyal    Loyal    Loyal    Loyal    Loyal    Loyal    Turks
 [9] Turks    Turks    Turks    Turks    Turks    Turks    Outcasts Outcasts
[17] Outcasts Outcasts
Levels: Loyal Outcasts Turks

> net[["rnorm"]] <- rnorm(18)
> net[["rnorm"]]

 [1]  0.14441451 -0.57949376 -3.47592196 -0.94085147  0.37996743 -0.11105589
 [7] -1.14843258  0.59404012  0.56828957 -1.20534168  0.57650561 -0.78246131
[13]  1.19329181  0.86643758 -1.08681315 -0.05110687 -0.74386960 -0.57519834

>
> #See available methods
> #print(DirectedNet)
> #print(UndirectedNet)
```

5 A simple ERNM for Sampson's Monks

Here we will create a model with both the graph and group being random.

```
> net <- as.BinaryNet(samplike)
> form <- net ~ edges() + reciprocity() + homophily("group") + nodeCount("group") | group
> fit <- ernm(form,verbose=0)

calculating: ......

> fit

                           ERNM Model
Domain:
 Random graph = TRUE, Random variables = group

                edges reciprocity homophily nodeCount1 nodeCount2
Parameters  -1.148926    1.365997  5.667398 0.02746373  -2.222304
Mean Values 88.208500   28.064900 15.311497 7.09270000   3.992500

> #generate some networks from the model
> sampler <- createCppSampler(form)
> sampler$getModel()$setThetas(fit$theta)
> netList <- sampler$generateSample(1000,1000,3)
> par(mfrow=c(2,2))
> plot(net,main="Sampson's Monks")
> plot(netList[[1]],main="sim 1")
> plot(netList[[2]],main="sim 1")
> plot(netList[[3]],main="sim 1")
```



6 A latent class model for Sampson's Monks

```
> net <- as.BinaryNet(samplike)
> net[["latent"]] <- factor(rep(NA,18),levels=c("a","b","c"))
> form <- net ~ edges() + reciprocity() + homophily("latent") + nodeCount("latent") | latent
> fit <- ernm(form,modelType=MissingErnmModel,verbose=0,theta0=c(-1.8,2.4,1,0,0))

calculating: ....

> fit

                           ERNM Model (with missing values)
Domain:
 Random graph = TRUE, Random variables = latent

                edges reciprocity homophily nodeCount1 nodeCount2
Parameters  -1.125041    1.323231  5.773252  0.2307555  -2.216451
Mean Values 88.607600   28.107400 15.414312  7.2210000   4.008200

> #simulate 5 networks conditional upon observed graph
> sampler <- createCppSampler(form,sampler="NTDNBRNonObservedMetropolis"
+                              ,nodeSamplingPercentage=1)
```

```
> sampler$getModel()$setThetas(fit$theta)
> netList <- sampler$generateSample(1000,1000,5)
> #all simulations yield identical clusters
> cbind(SampsonsClusters=as.character(net[["group"]]),
+       Simulations=sapply(netList,function(x) x$getVariable("latent",maskMissing=FALSE)))

      SampsonsClusters
 [1,] "Loyal"          "b" "b" "b" "b" "b"
 [2,] "Loyal"          "b" "b" "b" "b" "b"
 [3,] "Loyal"          "b" "b" "b" "b" "b"
 [4,] "Loyal"          "b" "b" "b" "b" "b"
 [5,] "Loyal"          "b" "b" "b" "b" "b"
 [6,] "Loyal"          "b" "b" "b" "b" "b"
 [7,] "Loyal"          "b" "b" "b" "b" "b"
 [8,] "Turks"          "a" "a" "a" "a" "a"
 [9,] "Turks"          "a" "a" "a" "a" "a"
[10,] "Turks"          "a" "a" "a" "a" "a"
[11,] "Turks"          "a" "a" "a" "a" "a"
[12,] "Turks"          "a" "a" "a" "a" "a"
[13,] "Turks"          "a" "a" "a" "a" "a"
[14,] "Turks"          "a" "a" "a" "a" "a"
[15,] "Outcasts"       "c" "c" "c" "c" "c"
[16,] "Outcasts"       "c" "c" "c" "c" "c"
[17,] "Outcasts"       "c" "c" "c" "c" "c"
[18,] "Outcasts"       "c" "c" "c" "c" "c"
```