Networking Project

William D. Gizzi

11/7/2018

Load Data

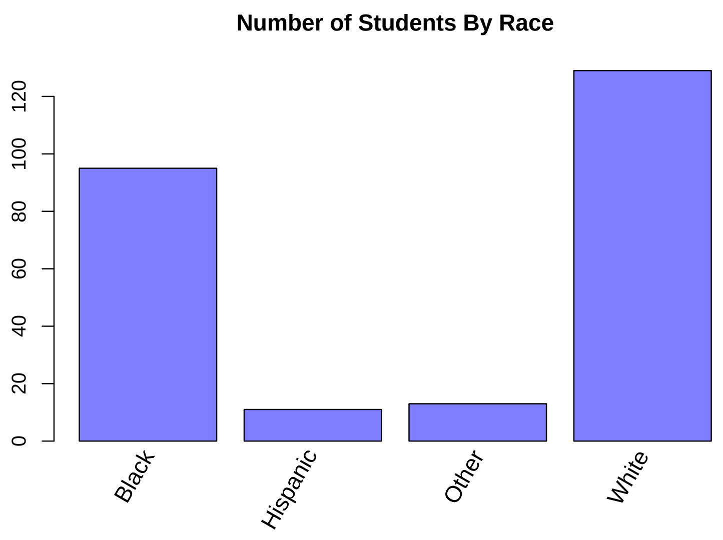
I will start by loading the data and visualizing the network.

```
library(statnet)
library(UserNetR)
library(igraph)
library(igraphdata)
library(intergraph)
library(ergm)

par()

set.seed(123)

data("faux.dixon.high")
dixon <- asIgraph(faux.dixon.high)
dixon.v.df <- as_data_frame(dixon, what = "vertices")
dixon.e.df <- as_data_frame(dixon, what = "edges")
write.csv(dixon.v.df, file = "dixon_V.csv")  #Vertices
write.csv(dixon.e.df, file = "dixon_e.csv")  #Edges
```

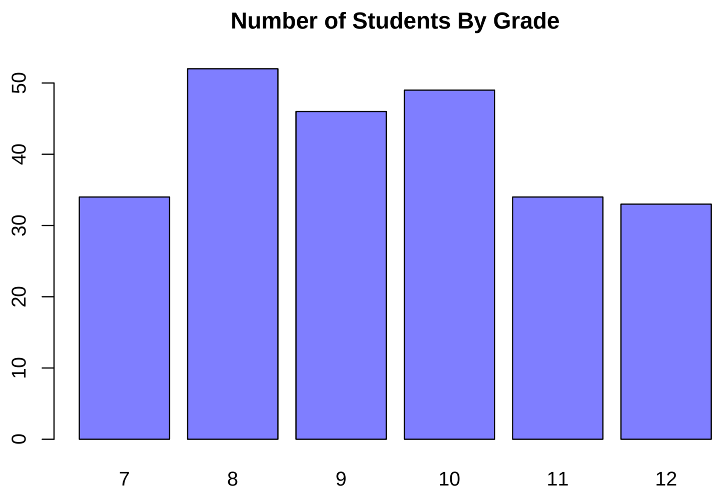
This network is a directed network of Dixon School. There are 248 nodes, 1197 edges, and node attributes of grade, race, and sex. Let's do a little exploration just to get an idea of what the data is like.

Exploration

```
# add some colors
blue <- "#7F7EFF"
green <- "#B8E0D2"
red <- "#E63462"

prep <- table(V(dixon)$race)
names = c("Black", "Hispanic", "Other", "White")
a = barplot(prep, col = blue, names.arg = "",
    main = "Number of Students By Race")
text(a[, 1], -3.7, srt = 60, adj = 1, xpd = TRUE,
    labels = names, cex = 1.2)
```



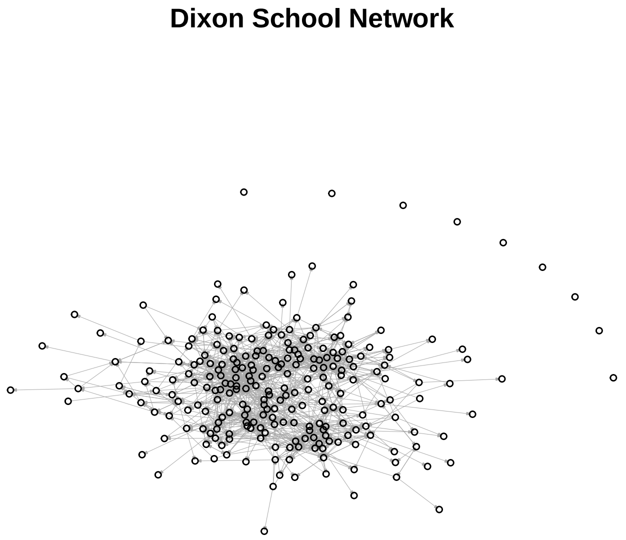
```
prep <- table(V(dixon)$grade)
b = barplot(prep, col = blue, main = "Number of Students By Grade")
```



The first graph suggests that there is an equal amount of non-minority students vs. minority students. There are slightly more students in grades 8-10, but not by an excessive amount. Let's plot the network so we can visualize the connections. I hypothesize that students will group together based on whether or not they are a minority, by whether they are in high school or middle school, and by sex.

Network Plots

```
par(mfrow = c(1, 1))
l <- layout.auto
plot(dixon, edge.width = 0.1, edge.arrow.size = 0.1,
    main = "Dixon School Network", vertex.label = NA,
    vertex.label.color = "black", vertex.label.cex = 0.3,
    vertex.color = NA, vertex.size = 2, vertex.label.dist = 1,
    rescale = T, layout = l, asp = 9/16)
```

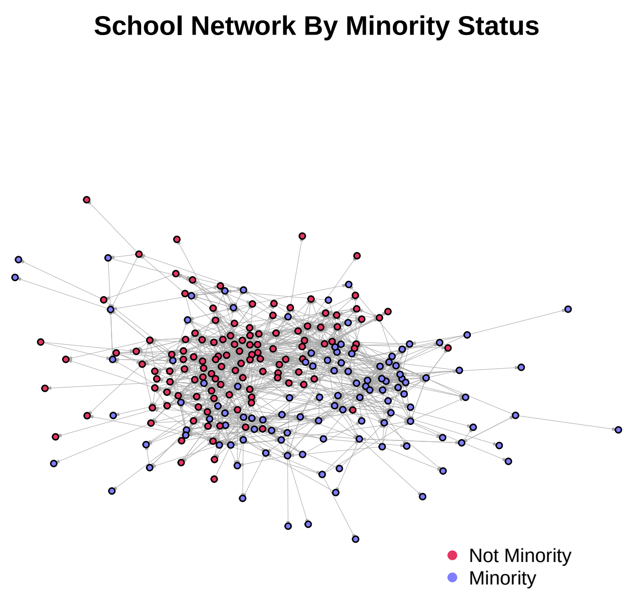


So, at first glance it looks like there are some nodes with 0 degree. Let's verify that and remove them.

```
degree(dixon)
```

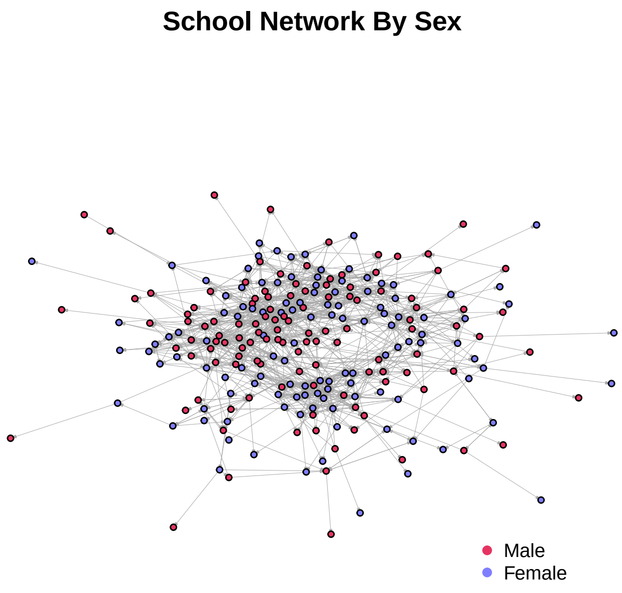
```
##   [1] 11 13  4 10  2  5  7 10  7  8 10 16  2
##  [14]  1 13  2  7  5 10  3 11  1  8 18  8 19
##  [27]  1 15  4 21  5 20  6  1  7 14  6 25  9
##  [40] 18 18 29  3  0  2 29 14 16 17  9  0  5
##  [53]  4  5 17 29 17 14  3 27  3  5  6 27  5
##  [66]  9  6  1 21  6  3  1  7  7  1 19  8  2
##  [79]  6 15 16 11 17 16 14  1 13  7 21  0  1
##  [92]  6  4  5  0  4 28 15 12 17 11  8  1  8
## [105]  8  5  2  2  1  4 14 10 25  7  5  7 29
## [118]  5 10 17  1 10  8 22 15 14  4 22  7  2
## [131] 18  2  2 12  9  7  0  9 18  1  3  2 10
## [144]  1 13 12  6 20 18  4  6  8  1 20  7 14
## [157]  5 14 18  4  6 10  0 12 19  6  6  1 13
## [170]  9  7  7  8 35 10  9 10 15 11  0 20  0
## [183]  1 12  9 16  6  4 21 14  4  5 10  8 16
## [196]  2 23 15  3  7  6  8  5  2  1 10 16 16
## [209] 12 17  8 10 23  1 11  3  3 12 10 10  5
## [222] 19 16 21 14 10  6 13  5 10  8 10  9  1
## [235] 12 12  4  1 15 18  7 15  9  0  2  6 22
## [248] 15
```

```
dixon <- delete_vertices(dixon, which(degree(dixon) <
    1))  #delete nodes with 0 degree

V(dixon)$color = ifelse(V(dixon)$race == "W",
    red, blue)  # set color by minority status, 'other' is counted as a minority
plot(dixon, edge.width = 0.1, main = "School Network By Minority Status",
    edge.arrow.size = 0.1, vertex.label = NA,
    vertex.label.color = "black", vertex.label.cex = 0.3,
    vertex.size = 2, vertex.label.dist = 1, rescale = T,
    layout = l, asp = 9/16)

legend("bottomright", legend = c("Not Minority",
    "Minority"), col = c(red, blue), pch = c(19,
    19), bty = "n", pt.cex = 0.8, cex = 0.8, text.col = "black",
    horiz = F, inset = c(0.1, 0.1))
```



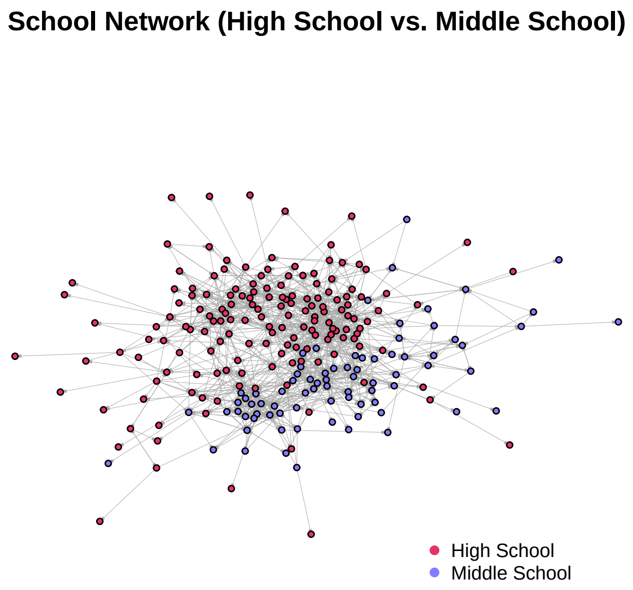
It is easy to observe two distinct clusters. Perhaps the race attribute will prove to be a useful attribute for our model. Let's look at a couple more visualizations.

```
l <- layout.auto
V(dixon)$color = ifelse(V(dixon)$sex == 1, red,
    blue)  # set color by sex
plot(dixon, edge.width = 0.1, main = "School Network By Sex",
    edge.arrow.size = 0.1, vertex.label = NA,
    vertex.label.color = "black", vertex.label.cex = 0.3,
    vertex.size = 2, vertex.label.dist = 1, rescale = T,
    layout = l, asp = 9/16)

legend("bottomright", legend = c("Male", "Female"),
    col = c(red, blue), pch = c(19, 19), bty = "n",
    pt.cex = 0.8, cex = 0.8, text.col = "black",
    horiz = F, inset = c(0.1, 0.1))
```

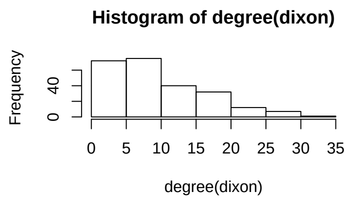


There doesn't seem to be much of a distinction between male and female.

```
l <- layout.auto
V(dixon)$color = ifelse(V(dixon)$grade >= 9, red,
    blue)
plot(dixon, edge.width = 0.1, main = "School Network (High School vs. Middle School)",
    edge.arrow.size = 0.1, vertex.label = NA,
    vertex.label.color = "black", vertex.label.cex = 0.3,
    vertex.size = 2, vertex.label.dist = 1, rescale = T,
    layout = l, asp = 9/16)

legend("bottomright", legend = c("High School",
    "Middle School"), col = c(red, blue), pch = c(19,
    19), bty = "n", pt.cex = 0.8, cex = 0.8, text.col = "black",
    horiz = F, inset = c(0.1, 0.1))
```



While it is expected to have each grade level cluster together, I was curious to see if there was an easily identifiable difference between high school and middle school students (also expected). In a future analysis, it might be interesting to see which middle schoolers and high schoolers act as "bridges" between middle school and high school students or communities. How does the degree distribution look? This could help us get an idea of "popularity."



```
hist(degree(dixon))
```

Here, the histogram of degree shows that most students have 0 - 10 connections. It looks like only a couple of students have the highest degree in the range 30 - 35. Let's look at some centrality measures.

Centrality

```
central <- centralization.degree(dixon)$res
between <- betweenness(dixon, normalized = TRUE)
close <- closeness(dixon, normalized = TRUE)
eigen <- eigen_centrality(dixon)$vector

info <- data.frame(central, round(between, 3),
    round(close, 3), round(eigen, 3), row.names = V(dixon)$vertex.names)
```

```
colnames(info) <- c("Degree Centrality", "Betweenness",
    "Closeness", "Eigenvector Centrality")

ordered <- info[order(-info$‘Degree Centrality‘),
    ]  # order by degree centrality
ordered[1:30, ]
```

```
##     Degree Centrality Betweenness Closeness
## 178                35       0.050     0.071
## 44                 29       0.040     0.072
## 48                 29       0.034     0.071
## 58                 29       0.061     0.073
## 121                29       0.033     0.072
## 101                28       0.032     0.071
## 63                 27       0.022     0.070
## 68                 27       0.028     0.072
## 40                 25       0.024     0.071
## 117                25       0.038     0.072
## 202                23       0.028     0.071
## 218                23       0.025     0.070
## 128                22       0.026     0.070
## 132                22       0.014     0.070
## 254                22       0.028     0.071
## 31                 21       0.032     0.071
## 73                 21       0.018     0.071
## 93                 21       0.025     0.072
## 193                21       0.013     0.071
## 229                21       0.025     0.072
## 33                 20       0.022     0.071
## 152                20       0.030     0.073
## 158                20       0.013     0.072
## 185                20       0.018     0.071
## 27                 19       0.022     0.070
## 80                 19       0.010     0.071
## 169                19       0.029     0.071
## 227                19       0.024     0.069
## 25                 18       0.014     0.070
## 42                 18       0.020     0.071
##     Eigenvector Centrality
## 178                  1.000
## 44                   0.919
## 48                   0.810
## 58                   0.639
## 121                  0.837
## 101                  0.750
## 63                   0.447
## 68                   0.656
## 40                   0.465
## 117                  0.731
## 202                  0.257
## 218                  0.435
## 128                  0.723
## 132                  0.704
## 254                  0.346
## 31                   0.582
## 73                   0.704
## 93                   0.238
## 193                  0.700
## 229                  0.589
## 33                   0.476
## 152                  0.507
## 158                  0.625
## 185                  0.402
## 27                   0.361
## 80                   0.607
## 169                  0.460
## 227                  0.262
## 25                   0.438
## 42                   0.608
```

Here, I calcuated different centrality metrics to get an idea of which students are the most popular. I've only shown the first 30 for the sake of saving space. It looks like student 178 is the most popular. Let's take a closer look at the students with the top 5 degree centrality and see if we can find anything interesting.

```
ordered[1:5, ]
```

```
##     Degree Centrality Betweenness Closeness
## 178                35       0.050     0.071
## 44                 29       0.040     0.072
## 48                 29       0.034     0.071
## 58                 29       0.061     0.073
## 121                29       0.033     0.072
##     Eigenvector Centrality
## 178                  1.000
## 44                   0.919
## 48                   0.810
## 58                    0.639
## 121                   0.837
```

The only thing that jumps out is that student 58 has a similar degree to the others, but a relatively larger betweenness than the others. This could mean that student 58 is crucial for information flow. Let's take a closer look at the betweenness values.

```
ordered2 <- info[order(-info$Betweenness), ]
ordered2[1:5, 1:2]
```

```
##     Degree Centrality Betweenness
## 58                 29       0.061
## 178                35       0.050
## 44                 29       0.040
## 117                25       0.038
## 228                16       0.036
```

So, student 228 has a relatively large betweenness with a lower degree than expected. Thus, student 228 may also be crucial for information flow. Next, let's see what communities we can identify from the network.

Communities

I've decicded to test out the edge betweenness community algorithm because there does seem to be significantly varying betweenness within the network, and the walktrap community algorithm because it works on both undirected and directed networks and its original publication suggests that it's a bit more accurate than fastgreedy.

```
edge.betweenness.community(dixon)
```

```
## IGRAPH clustering edge betweenness, groups: 124, mod: 0.18
## + groups:
##   $`1`
##   [1] 1
##
##   $`2`
##    [1]   2   4   6   7   8  10  12  13  45
##   [10]  47  50  56  72  79  86  89  93  96
##   [19] 107 108 120 123 124 127 141 147 150
##   [28] 165 170 177 181 182 186 189 198 201
##   [37] 204 210 216 218 221 223 224 227 228
##   [46] 230 232 234 235 237
##   + ... omitted several groups/vertices
```

It looks like the edge betweenness community algorithm detected 124 groups. This seems like a bit much. Let's see how the walktrap algorithm does.

```
walktrap <- walktrap.community(dixon)
walktrap
```

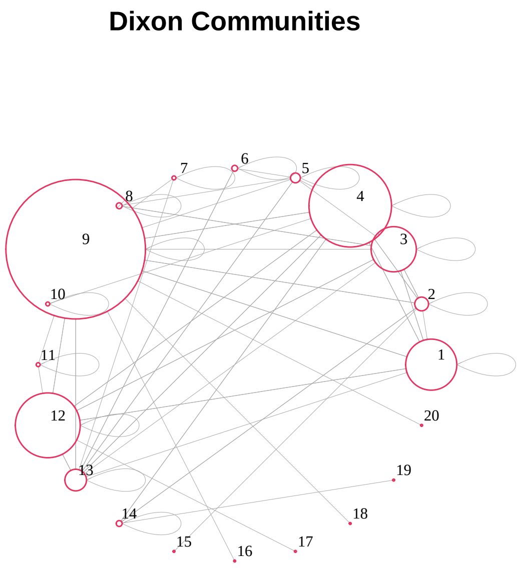
```
## IGRAPH clustering walktrap, groups: 20, mod: 0.51
## + groups:
##   $`1`
##    [1]  11  14  19  20  21  26  39  46  72
##   [10]  76  77  92 114 128 134 140 144 145
##   [19] 158 164 184 207 211 218 220 233
##
##   $`2`
##   [1]  13 104 193 195 219 224 230
##
##   $`3`
##    [1]   8  10  28  29  31  55  84  86  87
##   + ... omitted several groups/vertices
```

```
dixon <- set_vertex_attr(dixon, "community", V(dixon),
    walktrap$membership)  # add community as attribute
```

20 groups seems much more reasonable. Let's visualize it.

Community Visualization

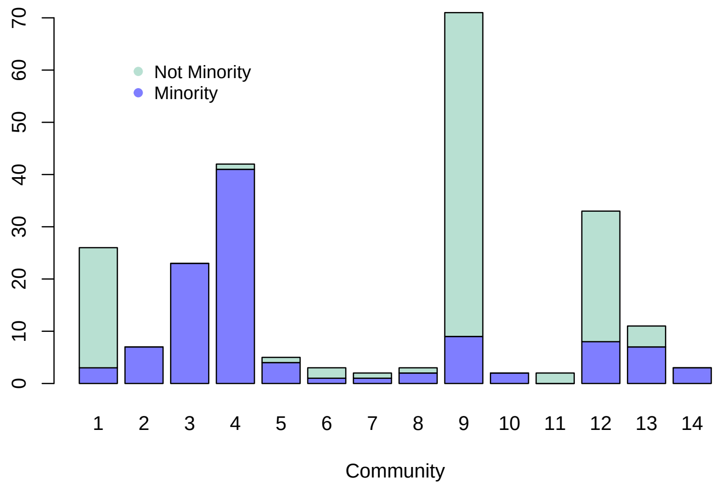
```
V(dixon)$size = 1
E(dixon)$count = 1
comm.graph <- contract.vertices(dixon, walktrap$membership,
    vertex.attr.comb = list(size = "sum", "ignore"))
comm.graph <- simplify(comm.graph, remove.loops = FALSE,
    edge.attr.comb = list(count = "sum", "ignore"))
plot(comm.graph, main = "Dixon Communities", edge.arrow.size = 0.01,
    edge.width = 0.3, vertex.frame.color = red,
    vertex.color = NA, vertex.label.color = "black",
    vertex.label.cex = 0.7, vertex.label.dist = 1,
    rescale = T, layout = layout_in_circle(comm.graph))
```



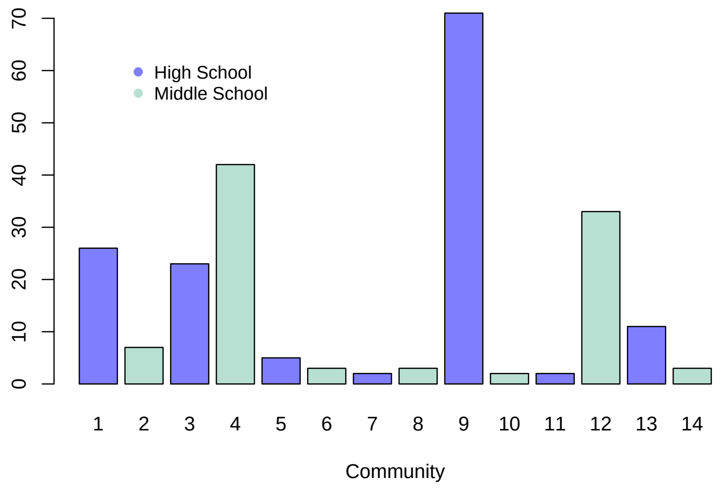
Communities 9, 4, and 12 seem to be the largest. Let's dive a little deeper into the specifics of each community.

Community Analysis

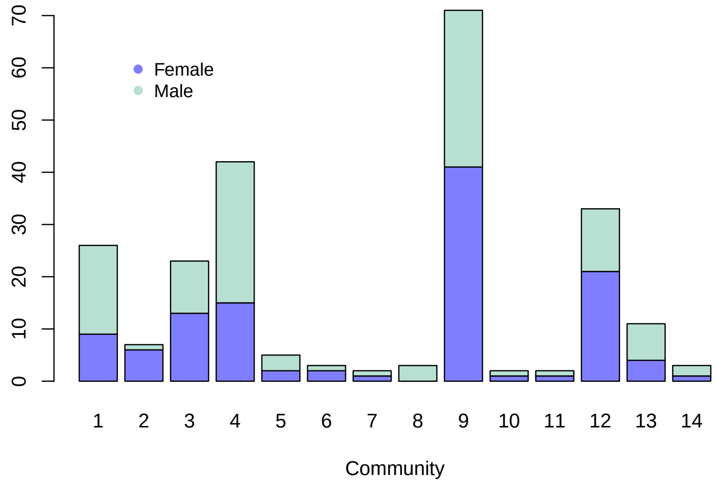
```
V(dixon)$color = ifelse(V(dixon)$race == "W",
    green, blue)
prep <- table(V(dixon)$color, V(dixon)$community)
prep <- prep[, 1:14]  # limit to groups with more than 1 member
barplot(prep, col = c(blue, green), xlab = "Community")

legend("topleft", legend = c("Not Minority", "Minority"),
    col = c(green, blue), pch = c(19, 19), bty = "n",
    pt.cex = 0.9, cex = 0.9, text.col = "black",
    horiz = F, inset = c(0.1, 0.1))
```



```
V(dixon)$color = ifelse(V(dixon)$race >= 9, green,
    blue)
prep <- table(V(dixon)$color, V(dixon)$community)
prep <- prep[, 1:14]  # limit to groups with more than 1 member
barplot(prep, col = c(blue, green), xlab = "Community")

legend("topleft", legend = c("High School", "Middle School"),
    col = c(blue, green), pch = c(19, 19), bty = "n",
    pt.cex = 0.9, cex = 0.9, text.col = "black",
    horiz = F, inset = c(0.1, 0.1))
```



```r
V(dixon)$color = ifelse(V(dixon)$sex == 2, green,
    blue)
prep <- table(V(dixon)$color, V(dixon)$community)
prep <- prep[, 1:14]  # limit to groups with more than 1 member
barplot(prep, col = c(blue, green), xlab = "Community")

legend("topleft", legend = c("Female", "Male"),
    col = c(blue, green), pch = c(19, 19), bty = "n",
    pt.cex = 0.9, cex = 0.9, text.col = "black",
    horiz = F, inset = c(0.1, 0.1))
```



The first barplot does a decent job of showing the distribution between non-minority and minority students for each community. While most communities aren't too diverse; communities 6, 12, and 13 have a decent amount of both non-minority and minority students. The second graph shows, as expected, that high schoolers and middle schoolers don't mix in communities. Other than communities 2 and 8, most communities seem to be evenly split between male and female. These findings suggest that the walktrap method is identifying communities in-line with what we observed in our earlier visualizations.

ERGM

First, let's estimate the Dixon network with the null model, the simplest possible model.

Null Model

```
set.seed(123)
dixon.net <- asNetwork(dixon)  # convert back in order to use communities in ERGM (we removed nodes wit
null <- ergm(dixon.net ~ edges)
```

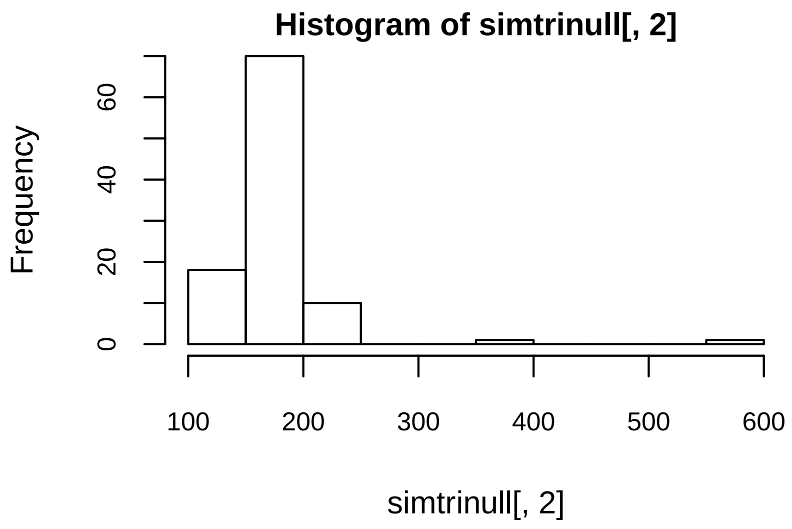
```
## Starting maximum pseudolikelihood estimation (MPLE):

## Evaluating the predictor and response matrix.

## Maximizing the pseudolikelihood.

## Finished MPLE.

## Stopping at the initial estimate.

## Evaluating log-likelihood at the estimate.
```

```
summary(null)
```

```
##
## ==========================
## Summary of model fit
## ==========================
##
## Formula:   dixon.net ~ edges
##
## Iterations:  7 out of 20
##
## Monte Carlo MLE Results:
##       Estimate Std. Error MCMC % z value
## edges -3.83989    0.02921      0  -131.4
##       Pr(>|z|)
## edges   <1e-04 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##      Null Deviance: 78855  on 56882  degrees of freedom
##  Residual Deviance: 11612  on 56881  degrees of freedom
##
## AIC: 11614    BIC: 11623    (Smaller is better.)
```

First, the negative edges coefficient shows that the density of the network is less than 50%, which is normal. The coefficient can also be used to show that the probability of creating an additional edge by adding another node is 0.02104361

```
plogis(-3.83989)

## [1] 0.02104361

simtrinull <- simulate(null, nsim = 100, monitor = ~triangles,
    statsonly = TRUE, control = control.simulate.ergm(MCMC.burnin = 1000,
        MCMC.interval = 1000), seed = 567)

## Warning: You appear to be calling
## simulate.formula() directly.
## simulate.formula() is a method, and will not
## be exported in a future version of 'ergm'.
## Use simulate() instead, or getS3method() if
## absolutely necessary.

dixon.tri <- summary(faux.dixon.high ~ triangle)
par(mar = c(4, 4, 1, 1), cex.main = 0.9, cex.lab = 0.9,
    cex.axis = 0.75)
hist(simtrinull[, 2])
points(dixon.tri, 3, pch = "X", cex = 2)
```



```
"Actual number of triangles in Dixon network"

## [1] "Actual number of triangles in Dixon network"

sum(count_triangles(dixon))

## [1] 1878
```

The histogram shows that the null model does not get anywhere near all of the formed triangles (1878 total). Let's add some attributes to our model and see if we can make it better.

Full Model

```
set.seed(123)
model <- ergm(dixon.net ~ edges + mutual + nodematch("grade") +
    nodematch("race") + nodematch("sex") + gwesp(0.25,
    fixed = TRUE), control = control.ergm(MCMC.samplesize = 40000,
    MCMC.interval = 1000))
```

```
## Starting maximum pseudolikelihood estimation (MPLE):

## Evaluating the predictor and response matrix.

## Maximizing the pseudolikelihood.

## Finished MPLE.

## Starting Monte Carlo maximum likelihood estimation (MCMLE):

## Iteration 1 of at most 20:

## Optimizing with step length 0.51607199443621.

## The log-likelihood improved by 14.75.

## Iteration 2 of at most 20:

## Optimizing with step length 0.679052872040695.

## The log-likelihood improved by 4.866.

## Iteration 3 of at most 20:

## Optimizing with step length 0.555095831559625.

## The log-likelihood improved by 4.204.

## Iteration 4 of at most 20:

## Optimizing with step length 1.

## The log-likelihood improved by 3.74.

## Step length converged once. Increasing MCMC sample size.

## Iteration 5 of at most 20:

## Optimizing with step length 1.

## The log-likelihood improved by 0.5393.

## Step length converged twice. Stopping.
## Finished MCMLE.

## Evaluating log-likelihood at the estimate. Using 20 bridges: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1
## This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the mcmc.d
```

```
summary(model)
```

```
##
## ==========================
## Summary of model fit
## ==========================
##
## Formula:   dixon.net ~ edges + mutual + nodematch("grade") + nodematch("race") +
##     nodematch("sex") + gwesp(0.25, fixed = TRUE)
##
## Iterations:  5 out of 20
##
## Monte Carlo MLE Results:
##                  Estimate Std. Error MCMC %
## edges            -5.81822    0.06220      0
## mutual            1.86679    0.12051      0
## nodematch.grade   1.14544    0.04622      0
## nodematch.race    0.75576    0.04843      0
## nodematch.sex     0.22154    0.04784      0
## gwesp.fixed.0.25  1.20139    0.04797      0
##                  z value Pr(>|z|)
## edges            -93.539   <1e-04 ***
## mutual            15.491   <1e-04 ***
## nodematch.grade   24.784   <1e-04 ***
## nodematch.race    15.604   <1e-04 ***
## nodematch.sex      4.631   <1e-04 ***
## gwesp.fixed.0.25  25.046   <1e-04 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##      Null Deviance: 78855  on 56882  degrees of freedom
##  Residual Deviance:  8374  on 56876  degrees of freedom
##
## AIC: 8386    BIC: 8440    (Smaller is better.)
```

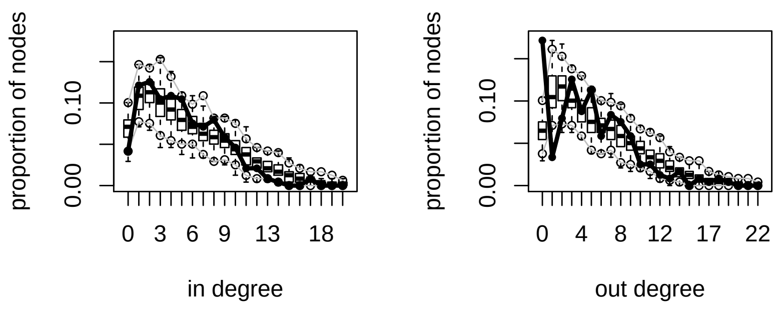
Ok, so the AIC value here is much lower (8448 vs. 11793) which suggests a better fit. Furthermore, all of the selected features are significant (p-value < 0.05). The hypothesis is that ties involving homophily of grade, race, and sex are significantly more than what would be expected in a simple random graph. Furthermore I added a term for mutuality (tendency for ties to be reciprocated) and the gwesp term to account for the social preference to be friends with your friends' friends.

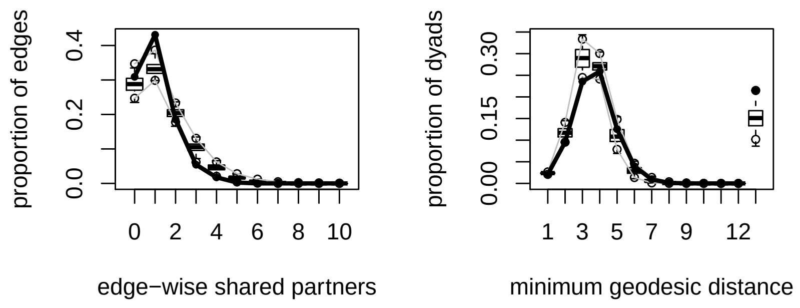
GOF

```
nullsim <- simulate(null, verbose = TRUE, seed = 123)

## Starting MCMC iterations to generate  1  network

## Finished simulation 1 of 1.

mainsim <- simulate(model, verbose = TRUE, seed = 123)  #Simulations based on full model

## Starting MCMC iterations to generate  1  network
## Finished simulation 1 of 1.

rowgof <- rbind(summary(faux.dixon.high ~ edges +
    triangle), summary(nullsim ~ edges + triangle),
    summary(mainsim ~ edges + triangle))
rownames(rowgof) <- c("Dixon", "Null", "Full Model")
rowgof

##            edges triangle
## Dixon       1197     1595
## Null        1195      158
## Full Model  1262     1681
```

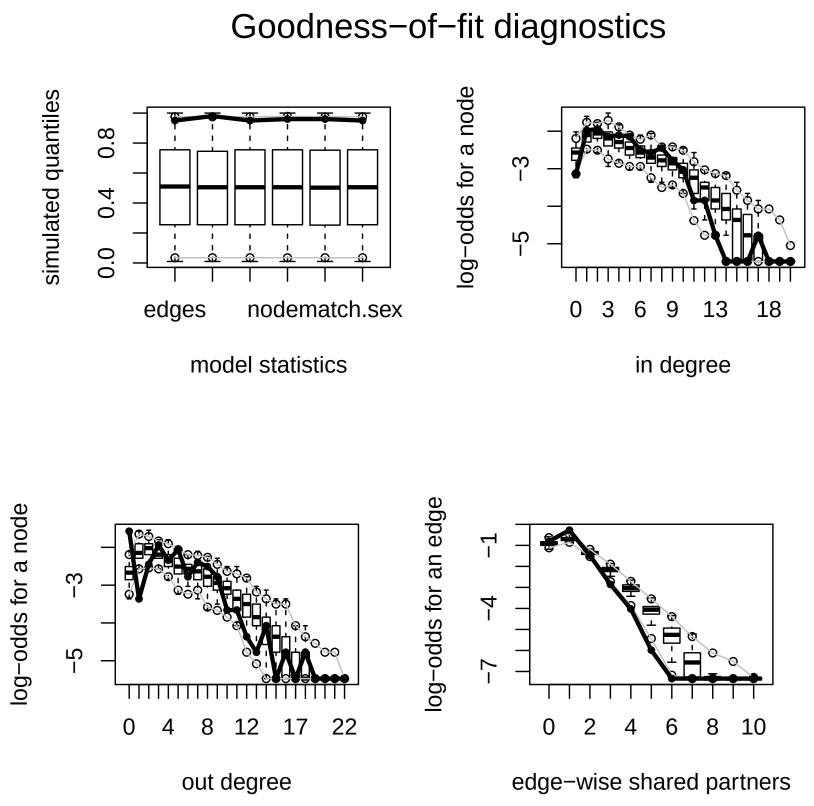
Here, the full model is almost perfect. It does a much better job of getting all of the edges and triangles than the null model.

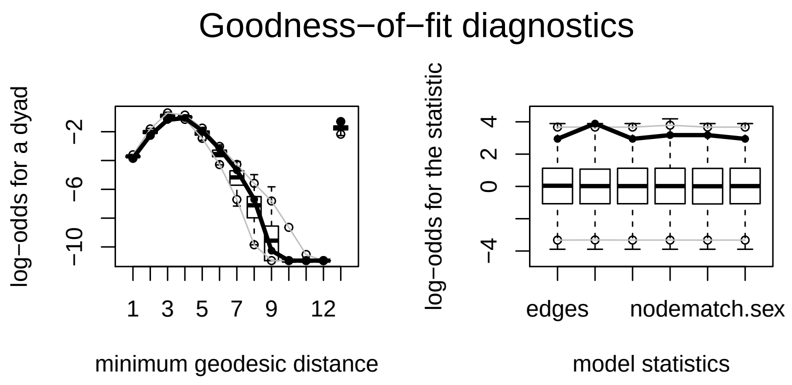
```
gof <- gof(model)
par(mfrow = c(1, 2))
plot(gof, cex.lab = 1, cex.axis = 1)
```





```
plot(gof, cex.lab = 1, cex.axis = 1, plotlogodds = T)
```





The goodness of fit graphs show that most of the terms have been modeled well. There may be some minor issue with the out degree, but nothing jumps out as problematic.

```
gof
```

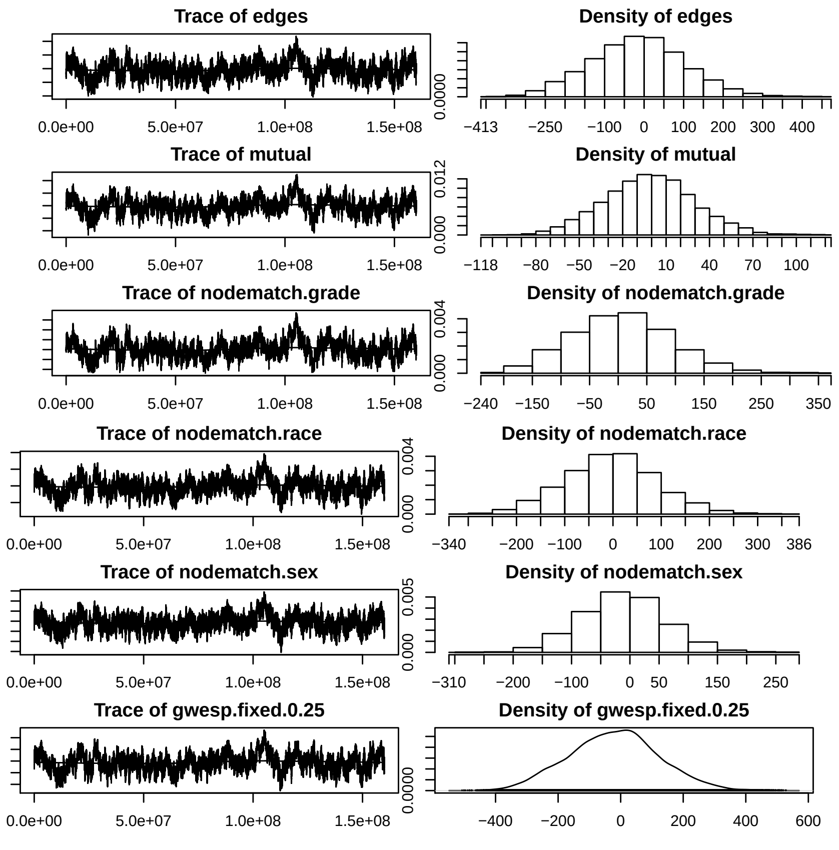
```
##
## Goodness-of-fit for in-degree
##
##    obs min  mean max MC p-value
## 0   10   7 16.55  28       0.06
## 1   29  17 26.06  43       0.50
## 2   30  16 26.58  35       0.52
## 3   25  11 24.13  43       0.88
## 4   26  11 21.77  33       0.38
## 5   25   9 18.88  27       0.14
## 6   18   8 17.38  26       0.96
## 7   17   8 15.76  27       0.76
## 8   19   5 13.68  20       0.12
## 9   14   5 12.85  20       0.76
## 10  11   3 10.94  19       1.00
## 11   5   1  8.64  17       0.28
## 12   5   1  6.60  11       0.60
## 13   2   1  5.28  10       0.20
## 14   1   0  4.24  10       0.20
## 15   0   0  2.87   8       0.16
## 16   0   0  2.30   6       0.26
## 17   2   0  1.54   6       1.00
## 18   0   0  1.01   4       0.76
## 19   0   0  0.74   3       1.00
## 20   0   0  0.37   2       1.00
## 21   0   0  0.39   3       1.00
## 22   0   0  0.19   2       1.00
## 23   0   0  0.10   2       1.00
## 24   0   0  0.05   2       1.00
## 25   0   0  0.05   1       1.00
## 26   0   0  0.01   1       1.00
## 27   0   0  0.01   1       1.00
## 28   0   0  0.01   1       1.00
## 29   0   0  0.01   1       1.00
## 30   0   0  0.01   1       1.00
##
## Goodness-of-fit for out-degree
##
##    obs min  mean max MC p-value
## 0   41   7 15.81  29       0.00
## 1    8  12 26.58  41       0.00
## 2   19  12 27.42  42       0.14
## 3   30  15 24.55  34       0.32
## 4   21  13 21.12  34       1.00
## 5   27   9 18.72  28       0.08
## 6   14   8 16.72  29       0.46
## 7   20   8 15.98  26       0.44
## 8   18   5 14.22  23       0.46
## 9   14   4 12.40  22       0.62
## 10   6   4 10.63  18       0.16
## 11   6   2  8.43  18       0.54
## 12   3   1  7.05  14       0.14
## 13   2   0  5.45  11       0.20
## 14   4   0  4.08  10       1.00
## 15   0   0  3.12   8       0.14
## 16   2   0  2.43   8       1.00
## 17   1   0  1.32   6       1.00
## 18   2   0  1.06   4       0.56
## 19   1   0  0.78   5       1.00
## 20   0   0  0.45   3       1.00
## 21   0   0  0.26   2       1.00
## 22   0   0  0.18   2       1.00
## 23   0   0  0.02   1       1.00
## 24   0   0  0.14   2       1.00
## 25   0   0  0.05   1       1.00
## 26   0   0  0.01   1       1.00
## 27   0   0  0.01   1       1.00
## 28   0   0  0.01   1       1.00
##
## Goodness-of-fit for edgewise shared partner
##
##       obs min   mean max MC p-value
## esp0  370 340 387.91 441       0.46
## esp1  516 369 448.28 526       0.02
## esp2  222 194 276.00 342       0.10
## esp3   65  63 141.15 207       0.02
## esp4   21  19  61.90  96       0.02
## esp5    3   1  23.09  45       0.04
## esp6    0   0   7.45  18       0.04
## esp7    0   0   2.04   8       0.36
## esp8    0   0   0.57   6       1.00
## esp9    0   0   0.17   3       1.00
## esp10   0   0   0.04   1       1.00
## esp11   0   0   0.01   1       1.00
##
## Goodness-of-fit for minimum geodesic distance
##
##       obs   min     mean   max MC p-value
## 1    1197  1165  1348.61  1538       0.10
## 2    5419  5210  6660.47  8249       0.06
## 3   13393 13406 16420.02 19547       0.00
## 4   14733 13119 15406.38 17284       0.40
## 5    7132  3939  6311.73  8764       0.44
## 6    2262   618  1650.86  2924       0.30
## 7     524    44   350.97   954       0.44
## 8      70     0    65.26   389       0.74
## 9       2     0    11.23   167       0.78
## 10      0     0     1.98    77       1.00
## 11      0     0     0.29    17       1.00
## 12      0     0     0.02     1       1.00
## Inf 12150  4898  8654.18 14001       0.06
##
## Goodness-of-fit for model statistics
##
##                         obs       min
## edges             1197.0000 1165.0000
## mutual             219.0000  213.0000
## nodematch.grade    785.0000  738.0000
## nodematch.race     912.0000  828.0000
## nodematch.sex      681.0000  652.0000
## gwesp.fixed.0.25   900.4146  802.8701
##                       mean      max
## edges             1348.610 1538.000
## mutual             260.820  304.000
## nodematch.grade    898.820 1009.000
## nodematch.race    1033.110 1200.000
## nodematch.sex     766.190  893.000
## gwesp.fixed.0.25 1086.732 1344.895
##                  MC p-value
## edges                  0.10
## mutual                 0.04
## nodematch.grade        0.10
## nodematch.race         0.10
## nodematch.sex          0.08
## gwesp.fixed.0.25       0.10
```

For the in-degree nodes there doesn't appear to be any significant difference between the simulated network and original network for all of the nodes. The same can be said for the out-degree nodes except for those with a low p-value (nodes 0, 1)

```
par(mfrow = c(6, 2))
par(mar = c(2, 1, 2, 1))
mcmc <- mcmc.diagnostics(model)
```

```
## Sample statistics summary:
##
## Iterations = 16000:160015000
## Thinning interval = 1000
## Number of chains = 1
## Sample size per chain = 160000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                      Mean     SD Naive SE
## edges            -14.7699 119.09  0.29772
## mutual            -0.9498  31.18  0.07795
## nodematch.grade    7.1042  87.26  0.21815
## nodematch.race    -3.7486  95.40  0.23850
## nodematch.sex    -10.8529  71.18  0.17794
## gwesp.fixed.0.25 -20.7888 144.91  0.36228
##                  Time-series SE
## edges                     8.426
## mutual                    2.253
## nodematch.grade           6.936
## nodematch.race            6.781
## nodematch.sex             4.660
## gwesp.fixed.0.25         10.817
##
## 2. Quantiles for each variable:
```

```
##
##                     2.5%    25%   50%   75%
## edges            -244.0  -95.0 -14.0 63.00
## mutual            -61.0  -22.0  -1.0 19.00
## nodematch.grade  -153.0  -54.0   6.0 64.00
## nodematch.race   -190.0  -67.0  -3.0 57.00
## nodematch.sex    -148.0  -59.0 -11.0 35.00
## gwesp.fixed.0.25 -298.9 -119.1 -19.5 73.78
##                  97.5%
## edges            223.0
## mutual            62.0
## nodematch.grade  184.0
## nodematch.race   187.0
## nodematch.sex    131.0
## gwesp.fixed.0.25 267.8
##
##
## Sample statistics cross-correlations:
##                      edges    mutual
## edges            1.0000000 0.9476478
## mutual           0.9476478 1.0000000
## nodematch.grade  0.9571137 0.9416504
## nodematch.race   0.9731662 0.9363469
## nodematch.sex    0.9557053 0.9101989
## gwesp.fixed.0.25 0.9839804 0.9612074
##                  nodematch.grade
## edges                  0.9571137
## mutual                 0.9416504
## nodematch.grade        1.0000000
## nodematch.race         0.9239881
## nodematch.sex          0.9133904
## gwesp.fixed.0.25       0.9637902
##                  nodematch.race
## edges                 0.9731662
## mutual                0.9363469
## nodematch.grade       0.9239881
## nodematch.race        1.0000000
## nodematch.sex         0.9318973
## gwesp.fixed.0.25      0.9661288
##                  nodematch.sex
## edges                0.9557053
## mutual               0.9101989
## nodematch.grade      0.9133904
## nodematch.race       0.9318973
## nodematch.sex        1.0000000
## gwesp.fixed.0.25     0.9413272
##                  gwesp.fixed.0.25
## edges                   0.9839804
## mutual                  0.9612074
## nodematch.grade         0.9637902
## nodematch.race          0.9661288
## nodematch.sex           0.9413272
## gwesp.fixed.0.25        1.0000000
##
## Sample statistics auto-correlation:
## Chain 1
##              edges    mutual nodematch.grade
## Lag 0    1.0000000 1.0000000       1.0000000
## Lag 1000 0.9880979 0.9936961       0.9930185
## Lag 2000 0.9784609 0.9883399       0.9872677
## Lag 3000 0.9704768 0.9836334       0.9823703
## Lag 4000 0.9636753 0.9793548       0.9780716
## Lag 5000 0.9576978 0.9754390       0.9741891
##          nodematch.race nodematch.sex
## Lag 0         1.0000000     1.0000000
## Lag 1000      0.9892165     0.9826341
## Lag 2000      0.9804473     0.9690313
## Lag 3000      0.9730991     0.9579830
## Lag 4000      0.9667423     0.9488101
## Lag 5000      0.9611299     0.9409799
##          gwesp.fixed.0.25
## Lag 0           1.0000000
## Lag 1000        0.9939829
## Lag 2000        0.9887945
## Lag 3000        0.9841143
## Lag 4000        0.9797846
## Lag 5000        0.9757317
##
## Sample statistics burn-in diagnostic (Geweke):
## Chain 1
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##            edges           mutual
##           -1.941           -1.891
##  nodematch.grade   nodematch.race
##           -1.671           -2.011
##    nodematch.sex gwesp.fixed.0.25
##           -2.278           -1.994
##
## Individual P-values (lower = worse):
##            edges           mutual
##       0.05222609       0.05860359
##  nodematch.grade   nodematch.race
##       0.09464680       0.04435770
##    nodematch.sex gwesp.fixed.0.25
##       0.02274023       0.04610303
## Joint P-value (lower = worse):  0.178928 .
```



```
##
## MCMC diagnostics shown here are from the last round of simulation, prior to computation of final para
```

The MCMC diagnostic plots show that the process converged for each term (no skewness and generally centered close to 0)