condor Vignette

September 24, 2015

1 Introduction

COmplex Network Description Of Regulators (CONDOR) implements methods for clustering bipartite networks and estimating the contribution of each node to its community's modularity. For an application of this method to identify diesease-associated single nucleotide polymorphisms, see `http://arxiv.org/abs/1509.02816`.

2 Implementing the Bipartite Modularity Maximization

The code in **condor.modularity.max** is an implementation of the method described in Michael Barber's paper **Modularity and community detection in bipartite networks** (Phys. Rev. E 76, 066102 (2007)). A few general comments:

- Maximizing bipartite modularity is an NP-hard problem
- This method is heuristic and can depend on initial assignments of the nodes to communities
- For the implementation in **condor.cluster**, I use a non-bipartite community detection method from the **igraph** package to use as initial assignments of nodes to communities, which are then used in **condor.modularity.max**.
- Community structure is designed to cluster networks that form a giant connected component. All of the analysis in this package uses the giant connected component.

3 Workflow

```
> library(condor)
> library(igraph)
```

condor works with an edgelist (**elist** in the code below) as its input.

```
> r = c(1,1,1,2,2,2,3,3,3,4,4);
> b = c(1,2,3,1,2,4,2,3,4,3,4);
> reds <- c("Alice","Sue","Janine","Mary")
> blues <- c("Bob","John","Ed","Hank")
> elist <- data.frame(red=reds[r],blue=blues[b])
```

In **elist**, notice all nodes of the same type–women and men in this case–appear in the same column together. This is a requirement. **create.condor.object** will throw an error if a node appears in both columns.

```
> condor.object <- create.condor.object(elist)
```

A condor.object is just a list. You can look at the different items using **names**

```
> names(condor.object)
```

```
[1] "G"          "edges"      "Qcoms"      "modularity" "red.memb"
[6] "blue.memb"  "qscores"
```

condor.cluster will cluster the nodes and produce the overall modularity along with two community membership **data.frames**:

```
> condor.object <- condor.cluster(condor.object)

[1] "modularity of projected graph 0"
[1] "making new comm"
[1] "Q = 0.140495867768595"
[1] "Q = 0.231404958677686"
[1] "Q = 0.231404958677686"

> print(condor.object$red.memb)

  red.names com
1     Alice   2
2    Janine   1
3      Mary   1
4       Sue   2

> print(condor.object$blue.memb)

  blue.names com
1        Bob   2
2         Ed   1
3       Hank   1
4       John   2
```

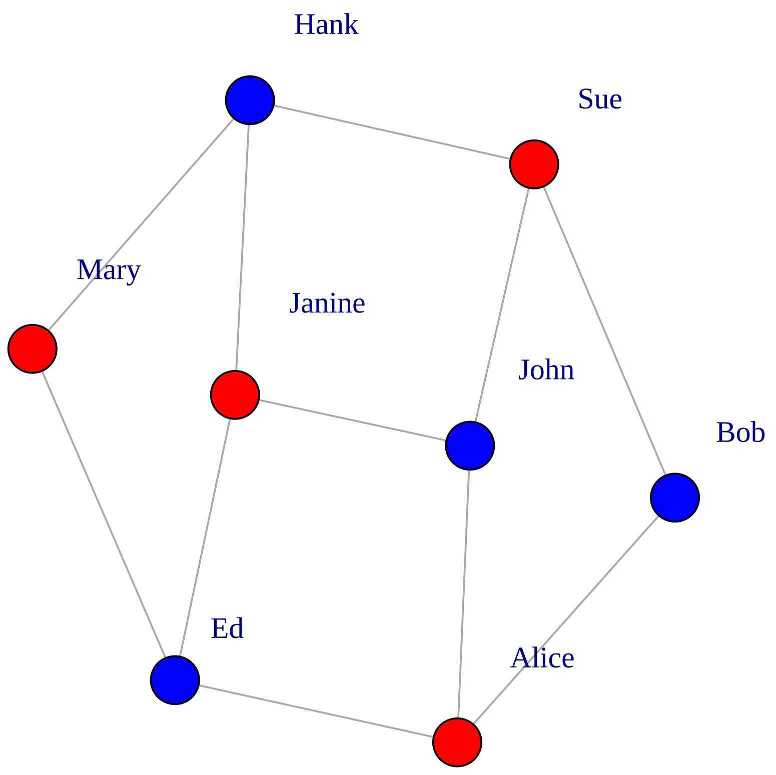
Nodes in first community are Alice, John, Bob, Sue, nodes in second community are Ed, Janine, Hank, Mary based on the modularity maximization. Here's a picture:

```
> gtoy = graph.edgelist(as.matrix(elist),directed=FALSE)
> set.graph.attribute(gtoy, "layout", layout.kamada.kawai(gtoy))

IGRAPH UN-- 8 11 --
+ attr: layout (g/n), name (v/c)

> V(gtoy)[c(reds,blues)]$color <- c(rep("red",4),rep("blue",4))

> plot(gtoy,vertex.label.dist=2)
```

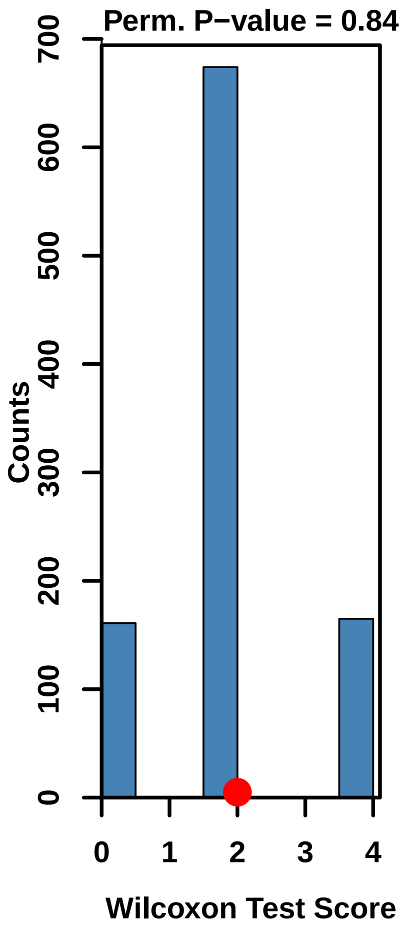


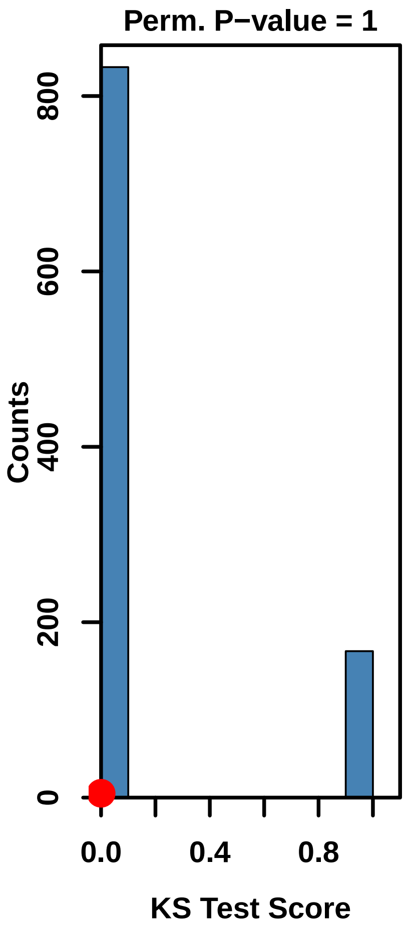
To get each node's modularity contribution (as a fraction of the community's modularity), run

```
> condor.object <- condor.qscore(condor.object)
```

If you have a subset of nodes that you think are more likely to lie at the cores of your communities, you can test this using **condor.core.enrich**:

```
> q_women <- condor.object$qscores$red.qscore
> core_stats <- condor.core.enrich(test_nodes=c("Alice","Mary"),
+                                  q=q_women,perm=TRUE,plot.hist=TRUE)
```





If you have questions, contact John Platig at jplatig@jimmy.harvard.edu