Forum

Start page
Recent posts

Links

The igraph homepage
igraph at launchpad
igraph-help mailing list
Projects that use igraph

Tutorials

R tutorial
Python tutorial

Cookbooks

Quick and easy R
R traps
R recipes
Community Detection In R
Python recipes
Community Detection In Python

About this wiki

What is a Wiki Site?
How to edit pages?
How to join this site?
Site members
Recent changes
List all pages
Page Tags

Page tags

assortativity bioinformatics community detection latex python r tikz tutorial visualization

Add a new page

new page

Community Detection In R

Fold

Table of Contents

1 Clique percolation

Clique percolation is a community detection method developed by Gergely Palla and his co-workers, see Palla, Gergely, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435(7043):814-8. Pubmed Arxiv.
This algorithm is not implemented in igraph, but here is a quick (and rather inefficient) version to do it:

```
clique.community <- function(graph, k) {
    clq <- cliques(graph, min=k, max=k)
    edges <- c()
    for (i in seq_along(clq)) {
      for (j in seq_along(clq)) {
        if ( length(unique(c(clq[[i]], clq[[j]]))) == k+
          edges <- c(edges, c(i,j)-1)
        }
      }
    }
    clq.graph <- simplify(graph(edges))
    V(clq.graph)$name <- seq_len(vcount(clq.graph))
    comps <- decompose.graph(clq.graph)

    lapply(comps, function(x) {
      unique(unlist(clq[ V(x)$name ]))
    })
}
```

2 Label propagation algorithm by Raghavan et al.

Usha Nandini Raghavan, Réka Albert and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks, Phys. Rev. E 76, 036106 Arxiv

A quick implementation by Peter McMahan, he sent this to the igraph-help mailing list.

```
largeScaleCommunity <- function(g,mode="all"){
  V(g)$group <- as.character(V(g))
  thisOrder <- sample(vcount(g),vcount(g))-1
  t <- 0
  done <- FALSE
  while(!done){
    t <- t+1
    cat("\rtick:",t)
    done <- TRUE ## change to FALSE whenever a node cha
    for(i in thisOrder){
      ## get the neighbor group frequencies:
      groupFreq <- table(V(g)[neighbors(g,i,mode=mode)]
      ## pick one of the most frequent:
      newGroup <- sample(names(groupFreq) [groupFreq==m
      if(done){done <- newGroup==V(g)[i]$group}
      V(g)[i]$group <- newGroup
    }
  }
  ## now fix any distinct groups with same labels:
  for(i in unique(V(g)$group)){
    ## only bother for connected groups
    if(!is.connected(subgraph(g,V(g)[group==i]))){
      theseNodes <- V(g)[group==i]
      theseClusters <- clusters(subgraph(g,theseNodes))
      ## iterate through the clusters and append their
      for(j in unique(theseClusters$membership)){
        V(g)[theseNodes[theseClusters$membership==j]]$g
      }
    }
  }
  return(g)
}
```

3 How to use the community detection algorithms?

3.1 Code

Gábor wrote this in the mailing-list.

```
memberships <- list()

### edge.betweenness.community
ebc <- edge.betweenness.community(G)
mods <- sapply(0:ecount(G), function(i) {
 g2 <- delete.edges(G, ebc$removed.edges[seq(length=i)]
 cl <- clusters(g2)$membership
 modularity(G, cl)
})

g2 <- delete.edges(G, ebc$removed.edges[1:(which.max(mo
memberships$`Edge betweenness` <- clusters(g2)$membersh

### fastgreedy.community
fc <- fastgreedy.community(G)
memb <- community.to.membership(G, fc$merges,
                               steps=which.max(fc$modul
memberships$`Fast greedy` <- memb$membership

### leading.eigenvector.community
lec <- leading.eigenvector.community(G)
memberships$`Leading eigenvector` <- lec$membership

### spinglass.community
sc <- spinglass.community(G, spins=10)
memberships$`Spinglass` <- sc$membership

### walktrap.community
wt <- walktrap.community(G, modularity=TRUE)
```

```
wmemb <- community.to.membership(G, wt$merges,
                                steps=which.max(wt$modu
memberships$`Walktrap` <- wmemb$membership

### label.propagation.community
memberships$`Label propagation` <- label.propagation.co
```

3.2 Example

```
G <- graph.disjoint.union(graph.atlas(1000),graph.atlas
G <- add.edges(G,c(2,10,11,15,16,0))
G$layout <- layout.kamada.kawai
V(G)$color <- rainbow(3)[memberships$'Edge betweenness'
plot(G)
```

4 Testing the significance of a community

The following code snippet performs a Wilcoxon rank-sum test on the "internal" and "external" degrees of a community in order to quantify its significance. Let us call the edges within a community "internal" and the edges connecting the vertices of a community with the rest of the graph "external". The null hypothesis of the test is that there is no difference between the number of "internal" and "external" edges incident to a vertex of the community. More internal than external edges show that the community is significant; less internal than external edges show that the community is in fact an "anti-community". The p-value of the test performed by this function will be close to zero in both cases; the value of the test statistic tells us whether we have a community or an anti-community.

```
community.significance.test <- function(graph, vs, ...)
    if (is.directed(graph)) stop("This method requires
    subgraph <- induced.subgraph(graph, vs)
    in.degrees <- degree(subgraph)
    out.degrees <- degree(graph, vs) - in.degrees
    wilcox.test(in.degrees, out.degrees, ...)
}
```

community detection r

page revision: 24, last edited: 6 May 2015, 23:04 (94 days ago)

Edit Rate (0) Tags Discuss (0) History Files Print

Site tools + Options

Powered by Wikidot.com Help | Terms of Service | Privacy | Report a bug | Flag as objectionable

Unless otherwise stated, the content of this page is licensed under GNU Free Documentation License.

Other interesting sites

Nopalla otsaan

UCSD Lab Medicine
Spring 2010

AviationKnowledge

Khai's personal knowledge vault.
Materials on this site are not original. When possible, references to original articles are listed on each page.