CSCI 6907: Project #1

Samuel Dooley

February 15, 2016

Our goal is to answer the following questions about the Astronomy co-authorship dataset:

1. Who is the *central person(s)* in the graph?
2. What is the *longest path* in the graph?
3. What is the *largest clique* in the graph?
4. Given a particular node/person, what is its/his/hers *ego*?
5. Given a particular node/person, what is its/his/hers *power centrality*

In the process of answering these five questions, we have played with the following functions:

1. `degree(graph, v = V(graph), mode = c("all", "out", "in", "total"), loops = TRUE, normalized = FALSE)`

   This is a function that takes a graph as input and outputs an array of all the graph's vertices with each one's corresponding degree. The degree of a vertices is its number of 'neighbors,' i.e., how many places can one go from a given node while traversing this graph. For example:

   ```
   > degree(astrocollab)
   BIERMANN, PL     STANEV, TKGT       GOLDMAN, I        WANDEL, A
            36                1                5                4
   ....
   ```

   This array is ordered as the array of vertices is ordered. This BIERMANN, PL appears first in both this array and the array of vertices (this can be tested by `V(astrocollab)[1]`). To get the answer for the person with largest degree, or the central person, we can perform

   ```
   > sort( degree(astrocollab), decreasing = TRUE)[1]
   ```

2. `betweenness(graph,, ..)`

   This function was used to compute the power-centrality of a given person. Depending upon the reference that is used, power-centrality is equivalent to the betweenness-centrality of a given node. The betweenness centrality of a node is calculated as

   $$\sum_{x \neq y \neq z} \frac{\sigma_{xz}(y)}{\sigma_{xz}}$$

   where $x, y, z \in V$, σ_{xz} is defined to be the total number of shortest paths from x to z and $\sigma_{xz}(y)$ is the total number of those that pass through node y.

   This problem is incredibly hard to compute, and so we use the simplified version of the graph.

3. `betweenness.estimate(graph, cutoff, ...)`
   Since the above function is too computationally intensive for the entire graph, we can take this function to estimate the betweenness-centrality. This is done with a slight modification that cuts off the distance between nodes x and z in the above formula. This makes the calculation as follows:

$$\sum_{\substack{x \neq y \neq z \\ p(x,z) \leq 10}} \frac{\sigma_{xz}(y)}{\sigma_{xz}}$$

   where $x, y, z \in V$, $p(x, z)$ is the shortest path length from x to z, σ_{xz} is defined to be the total number of shortest paths from x to z and $\sigma_{xz}(y)$ is the total number of those that pass through node y.

4. `largest_cliques(graph, ...)`
   This function computes the largest fully connected subgraph of `graph`. This can be done with simple matrix manipulation and linear algebra, so it is rather efficient. Thus we do not need to use the simplified version, and the original astronomy collaboration graph of full vertices.

5. `ego(graph, order, ...)`
   This function will take the `graph` and will compute, for each vertex, v, of `graph`, the subgraph which has all vertices in `graph` that are of `order` distance away from v. This can be done easily for small order, but it harder when the order gets larger. So when `order` is greater than about 7 for this graph, it is best to use the simplified graph for this.

6. `diameter(graph, ...)`
   This function computes the longest geodesic path in `graph`. A geodesic path is one such that there are no repeating vertices in the path which eliminates all problems of the path containing cycles.

   In general, this problem is NP-hard which at the very least shows that there is not a known polynomial-time algorithm for this problem, even if P=NP. So in order for this calculation to be done efficiently, we use the simplified subgraph for this problem.

7. `shortest.paths(graph, from, ...)`
   This function takes the `graph` and computes the shortest path from the vertex `from` to all the other vertices in the graph. The returned value will the the vertices traversed on that particular path from `from` to the other vertex.

   One can also specify a `to` vertex and the function will return the list of vertices traversed from `from` to `to`.

8. `clusters(graph, ..)`
   This function will take `graph` and determine what the connected components are. If the graph is connected, then `clusters` will return one subgraph. If it is not connected, then the function will return an array of the connected components. This is thus useful for simplifying the graph. For this particular graph (astronomy collaborators), we can use the function `cluster.distribution` to determine that the majority of the clusters are of size $<<$ the maximum size of clusters.

Answers

1. "FRONTERA, F"

2. If we run this on the entire graph, we get an answer of
   17.482562

3. This can be computed easily with the function

```
> largest.cliques(astrocollab)
```

Code

```
astrocollab <- upgrade_graph(astrocollab)


# Central person:
# input: graph
# output: array of strings with
#         names of nodes with the most neighbors
central_person <- function(graph) {

  d <-sort( degree(graph), decreasing = TRUE)

  # Get the people with the most
  m <- max(d)
  n <- names(which( m == d ))
  return(n)
}

# Simplify the graph
# First easy thing to be done would be to remove all the
# nodes that are isolates
remove_isolates <- function(graph) {

  d <- degree(graph)

  # find those that have zero neighbors
  new <- delete.vertices( graph, which( d == 0 ) )
  return( new )
}

# Can simplify the graph more
# We notice that there are 369 connectected components
# with the largest connected component comprising of 88%
# of the graph. Let's only analyze that part
largest_connected <- function(graph) {

  cls <- clusters(new)
  # what is the index of the largest connected component?
  i <- which.max(cls$csize)
  largest_cc <- which(clusters(new)$membership != i)

  new <- delete.vertices(graph, largest_cc)
  return(new)
}

simple <- largest_connected( remove_isolates( astrocollab ) )
```