Project 3

Logan Bolton

2025-02-23

Acknowledgement: This code was created through the repurposing of code found in the lecture notes and through collaboration with Claude 3.5 Sonnet and o3-mini. These AI tools were very helpful for me while fixing errors and determining the correct syntax to plot graphs.

Network Structure Analysis

Since the graph is NOT strongly connected, we have to find the diameter and radius using the largest strongly connected sub component.

The dataset also does not have duplicate sender -> receiver pairs, so we will analyze the dataset as an unweighted graph.

```
# Read the text file into a dataframe
df <- read.table("email-Eu-core.txt", header = FALSE, sep = " ")
colnames(df) <- c("From", "To")
# The dataset does not have weights so I abandoned this approach:
# df_weighted <- aggregate(weight ~ From + To, data = transform(df, weight = 1), FUN = sum)

g <- graph_from_data_frame(df, directed = TRUE)
# No need for weights
# g <- graph_from_data_frame(df_weighted, directed = TRUE)

num_nodes = vcount(g)
num_edges = ecount(g)

# Check strong connectivity
components <- components(g, mode="strong")

# Extract the vertices belonging to the largest strongly connected component
largest_comp_vertices <- V(g)[components$membership == which.max(components$csize)]
g_largest <- induced_subgraph(g, largest_comp_vertices)

# Calculate diameter and radius on the largest SCC
diameter_largest <- diameter(g_largest, directed = TRUE)
radius_largest <- radius(g_largest, mode = "out")


output <- paste(
  "Graph Summary",
  paste("Number of nodes:", num_nodes),
  paste("Number of edges:", num_edges),
  paste("Density:", round(edge_density(g), 6)),
  paste(""),
```

```
  "Connectivity:",
  paste("Number of strongly connected components:", components$no),
  paste("Size of largest strongly connected component:", max(components$csize)),
  paste("fraction of elements belonging to the largest strong subcomponent: ", max(components$csize)/nu
  "The graph is NOT strongly connected",
  paste(""),
  paste("Largest diameter:", diameter_largest),
  paste("Largest radius:", radius_largest),
  "-----------------------------------------------------------------------",

  sep = "\n"
)
cat(output)
```

```
## Graph Summary
## Number of nodes: 1005
## Number of edges: 25571
## Density: 0.025342
##
## Connectivity:
## Number of strongly connected components: 203
## Size of largest strongly connected component: 803
## fraction of elements belonging to the largest strong subcomponent:  0.799004975124378
## The graph is NOT strongly connected
##
## Largest diameter: 6
## Largest radius: 3
## -----------------------------------------------------------------------
```

Cluster coefficient computed through the ClustF function found in the DirectedClustering package:

```
adj_matrix <- as_adjacency_matrix(g, sparse = FALSE)

# Compute clustering coefficients (cycle type)
clustering_coeff <- ClustF(adj_matrix, type = "cycle")
```

```
## Warning in ClustF(adj_matrix, type = "cycle"): Loops have been removed
```

```
# Print the global cycle clustering coefficient
print(clustering_coeff$GlobalcycleCC)
```

```
## [1] 0.2934512
```

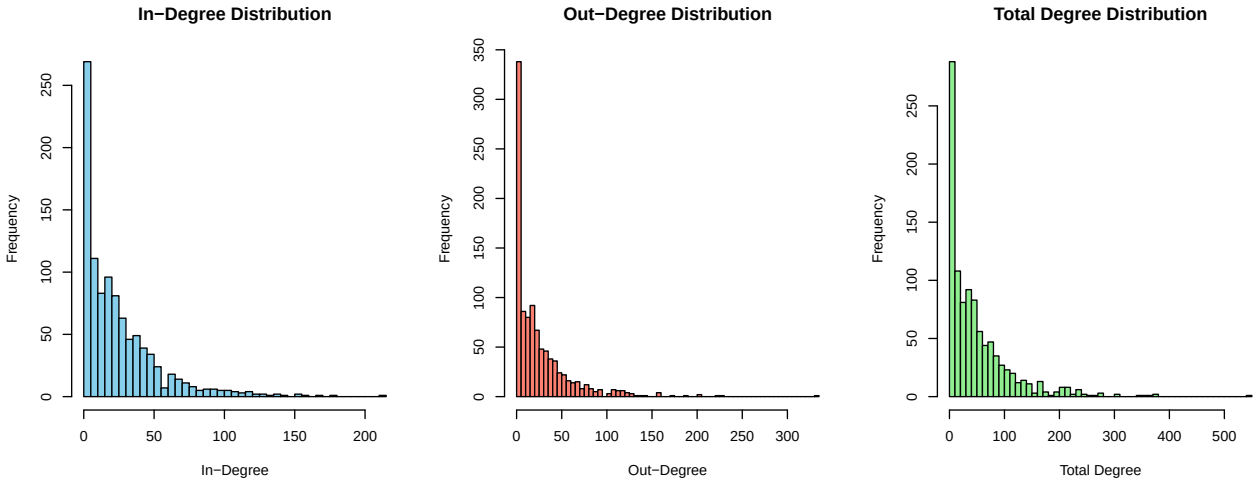
```
print(paste("Global out-clustering coefficient:", clustering_coeff$GlobaloutCC))
```

```
## [1] "Global out-clustering coefficient: 0.288824631612028"
```

Degree Distribution Analysis

```
# Calculate in-degree, out-degree, and total degree
in_degree <- degree(g, mode = "in")
out_degree <- degree(g, mode = "out")
total_degree <- degree(g, mode = "all")

# Set up a 1x3 plotting layout
```

```r
par(mfrow = c(1, 3))

# Plot histogram for in-degree
hist(in_degree, breaks = 50, main = "In-Degree Distribution",
     xlab = "In-Degree", ylab = "Frequency", col = "skyblue", border = "black")

# Plot histogram for out-degree
hist(out_degree, breaks = 50, main = "Out-Degree Distribution",
     xlab = "Out-Degree", ylab = "Frequency", col = "salmon", border = "black")

# Plot histogram for total degree
hist(total_degree, breaks = 50, main = "Total Degree Distribution",
     xlab = "Total Degree", ylab = "Frequency", col = "lightgreen", border = "black")
```




```r
# Reset plotting layout
par(mfrow = c(1, 1))

# Summary statistics
cat("In-Degree Summary:\n")
```

```
## In-Degree Summary:
```

```r
print(summary(in_degree))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    5.00   18.00   25.44   36.00  212.00
```

```r
cat("\nOut-Degree Summary:\n")
```

```
##
## Out-Degree Summary:
```

```r
print(summary(out_degree))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    2.00   15.00   25.44   35.00  334.00
```

```r
cat("\nTotal Degree Summary:\n")
```

```
##
## Total Degree Summary:
```

```r
print(summary(total_degree))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    8.00   32.00   50.89   71.00  546.00
```

Community Detection

I chose to utilize the Edge Betweenness algorithm and the Walktrap algorithm for community detection. I used these two algorithms because they both work for unweighted, directed graphs and because of how differently they determine communities.

```r
eb_communities <- cluster_edge_betweenness(g)
eb_membership <- membership(eb_communities)
eb_modularity <- modularity(eb_communities)
eb_sizes <- sizes(eb_communities)

# Walktrap Community Detection
wt_communities <- cluster_walktrap(g)
wt_membership <- membership(wt_communities)
wt_modularity <- modularity(wt_communities)
wt_sizes <- sizes(wt_communities)

# Print summary statistics for both algorithms
cat("Edge Betweenness Community Detection:\n")
```

```
## Edge Betweenness Community Detection:
```

```r
cat("Number of communities:", length(unique(eb_membership)), "\n")
```

```
## Number of communities: 710
```

```r
cat("Modularity:", round(eb_modularity, 4), "\n")
```

```
## Modularity: 0.0425
```

```r
cat("Community sizes:\n")
```

```
## Community sizes:
```

```r
print(sort(table(eb_membership), decreasing = TRUE)[1:10])
```

```
## eb_membership
##   1   3 146  60  91  97 103  29  56  96
## 233  17   6   5   4   4   4   3   3   3
```

```r
cat("\n")
```

```r
cat("Walktrap Community Detection:\n")
```

```
## Walktrap Community Detection:
```

```r
cat("Number of communities:", length(unique(wt_membership)), "\n")
```

```
## Number of communities: 136
```

```r
cat("Modularity:", round(wt_modularity, 4), "\n")
```

```
## Modularity: 0.3673
```

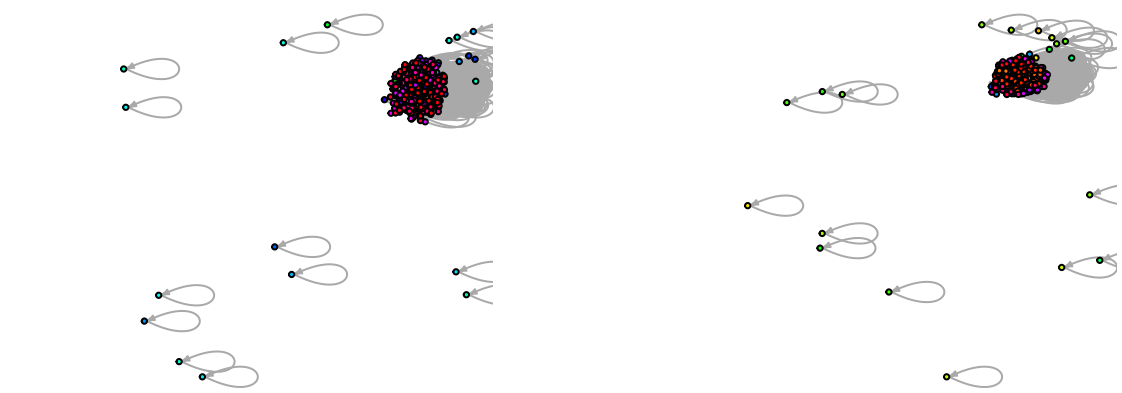
```r
cat("Community sizes:\n")
```

```
## Community sizes:
```

```r
print(sort(table(wt_membership), decreasing = TRUE)[1:10])
```

```
## wt_membership
##   4   1   5   9  12   7   8  11   6   3
## 352 109  91  85  85  84  30  22  12   4
```

```r
# Visualize the communities
# Function to plot communities with layout optimization
plot_communities <- function(graph, membership, title) {
  # Create color palette for communities
  num_communities <- length(unique(membership))
  colors <- rainbow(num_communities)

  # Set vertex colors based on community membership
  V(graph)$color <- colors[membership]

  # Calculate layout (using Fruchterman-Reingold algorithm)
  layout <- layout_with_fr(graph)

  # Plot the graph
  plot(graph,
       layout = layout,
       vertex.size = 3,
       vertex.label = NA,
       edge.arrow.size = 0.2,
       main = title)
}

# Set up plotting area for side-by-side comparison
par(mfrow = c(1, 2))

# Plot both community structures
plot_communities(g, eb_membership, "Edge Betweenness Communities")
plot_communities(g, wt_membership, "Walktrap Communities")
```




```
# Reset plotting parameters
par(mfrow = c(1, 1))
```

Triad Census Analysis

For the walktrap algorithm, these results indicate decent community detection. For the triads, over half are in the same community together.

However, for the edge betweenness algorithm, the low percentage of about 14% indicates that the algorithm may be breaking up communities into too small of groups. The large difference between both algorithms suggests that walktrap may be better suited than edge betweenness for detection.

Also unrelated note, but the 'triad_census' function here is fantastic. I tried to manually compute all the different triads using my own naive R code and my computer was not able to do that even after 8 hours of computation. I looked into random sampling every N triads per subcomponent of the graph because of these compute restraints, but the following approach was already very efficient and effective, so I abandoned that the N sampling strategy.

```
analyze_triads_communities <- function(g, communities) {
  # Get triad census
  triad_counts <- triad_census(g)

  # Get community memberships
  comm_mem <- membership(communities)

  # Function to check if three nodes form a triangle
  is_triangle <- function(nodes) {
    subg <- induced_subgraph(g, nodes)
    return(ecount(subg) >= 2)  # At least 2 edges for directed graph
  }

  # Get triangles using cliques
  # For directed graphs, we look at the underlying undirected graph for triangles
  g_undir <- as.undirected(g, mode="collapse")
  triangles <- cliques(g_undir, min=3, max=3)

  # Count triangles in same community
```

```
  same_comm_triangles <- 0
  total_triangles <- length(triangles)

  for(triangle in triangles) {
    if(length(unique(comm_mem[triangle])) == 1) {
      same_comm_triangles <- same_comm_triangles + 1
    }
  }

  # Create results
  results <- list(
    triad_census = triad_counts,
    triangle_stats = list(
      total_triangles = total_triangles,
      same_comm_triangles = same_comm_triangles,
      percent_same_comm = if(total_triangles > 0)
                            (same_comm_triangles/total_triangles) * 100
                          else 0
    )
  )

  return(results)
}

# Analyze both community detection methods
wt_triad_analysis <- analyze_triads_communities(g, wt_communities)
```

```
## Warning: `as.undirected()` was deprecated in igraph 2.1.0.
## i Please use `as_undirected()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
eb_triad_analysis <- analyze_triads_communities(g, eb_communities)

# Print results
cat("Triad Census Analysis Results:\n")
```

```
## Triad Census Analysis Results:
```

```
cat("----------------------------------------\n")
```

```
## ----------------------------------------
```

```
cat("Total triads by type:\n")
```

```
## Total triads by type:
```

```
print(wt_triad_analysis$triad_census)
```

```
##  [1] 153640073   6345756   7716387     81896     38347     58745    145903
##  [8]    262008      5639       419    279934      6984     11123      7455
## [15]     39656     34185
```

```
cat("\nTriangle Statistics:\n")
```

```
##
```

```
## Triangle Statistics:
```

```
cat("Walktrap Communities:\n")
```

```
## Walktrap Communities:
```

```
cat("  Total triangles:", wt_triad_analysis$triangle_stats$total_triangles, "\n")
```

```
##   Total triangles: 105461
```

```
cat("  Triangles in same community:", wt_triad_analysis$triangle_stats$same_comm_triangles, "\n")
```

```
##   Triangles in same community: 60034
```

```
cat("  Percentage in same community:",
    round(wt_triad_analysis$triangle_stats$percent_same_comm, 2), "%\n")
```

```
##   Percentage in same community: 56.93 %
```

```
cat("\nEdge Betweenness Communities:\n")
```

```
##
## Edge Betweenness Communities:
```

```
cat("  Total triangles:", eb_triad_analysis$triangle_stats$total_triangles, "\n")
```

```
##   Total triangles: 105461
```

```
cat("  Triangles in same community:", eb_triad_analysis$triangle_stats$same_comm_triangles, "\n")
```

```
##   Triangles in same community: 14697
```

```
cat("  Percentage in same community:",
    round(eb_triad_analysis$triangle_stats$percent_same_comm, 2), "%\n")
```

```
##   Percentage in same community: 13.94 %
```

Modularity Analysis

```
wt_mod_value <- modularity(g, wt_membership)
eb_mod_value <- modularity(g, eb_membership)

cat("Walktrap Modularity: ", wt_mod_value, "\n")
```

```
## Walktrap Modularity:  0.3821017
```

```
cat("Edge Betweenness Modularity: ", eb_mod_value, "\n")
```

```
## Edge Betweenness Modularity:  0.04254974
```

Evaluation

Based off this analysis, neither algorithm did very well at determining the correct communities. The edge betweenness algorithm did better than the walktrap algorithm in terms of NMI (0.444 vs 0.181), however both algorithms scored close to zero in the ARI score. In fact, the edge betweenness ARI score of (-0.002) indicates that the algorithm may actually be worse than random.

```
gt_communities <- read.table("email-Eu-core-department-labels.txt", header=FALSE)
colnames(gt_communities) <- c("Node", "Community")
```

```r
# Convert memberships to vectors ensuring they use the same node ordering
gt_membership <- gt_communities$Community
wt_membership_ordered <- membership(wt_communities)[order(as.numeric(V(g)))]
eb_membership_ordered <- membership(eb_communities)[order(as.numeric(V(g)))]

# Calculate NMI
nmi_wt <- NMI(gt_membership, wt_membership_ordered)
nmi_eb <- NMI(gt_membership, eb_membership_ordered)

# Calculate ARI
ari_wt <- ARI(gt_membership, wt_membership_ordered)
ari_eb <- ARI(gt_membership, eb_membership_ordered)

# Print results
cat("Comparison with Ground Truth:\n")
```

```
## Comparison with Ground Truth:
```

```r
cat("Walktrap Algorithm:\n")
```

```
## Walktrap Algorithm:
```

```r
cat("  NMI:", nmi_wt, "\n")
```

```
##   NMI: 0.1806671
```

```r
cat("  ARI:", ari_wt, "\n")
```

```
##   ARI: 0.001814829
```

```r
cat("\nEdge Betweenness Algorithm:\n")
```

```
##
## Edge Betweenness Algorithm:
```

```r
cat("  NMI:", nmi_eb, "\n")
```

```
##   NMI: 0.4440126
```

```r
cat("  ARI:", ari_eb, "\n")
```

```
##   ARI: -0.001998724
```

```r
# Create visualization comparing community sizes
par(mfrow=c(1,3))

# Ground Truth distribution
barplot(table(gt_membership),
        main="Ground Truth Community Sizes",
        xlab="Community ID",
        ylab="Size")

# Walktrap distribution
barplot(table(wt_membership),
        main="Walktrap Community Sizes",
        xlab="Community ID",
        ylab="Size")

# Edge Betweenness distribution
```

```
barplot(table(eb_membership),
        main="Edge Betweenness Community Sizes",
        xlab="Community ID",
        ylab="Size")
```