Network Structure

Kyra Bankhead

2023-03-02

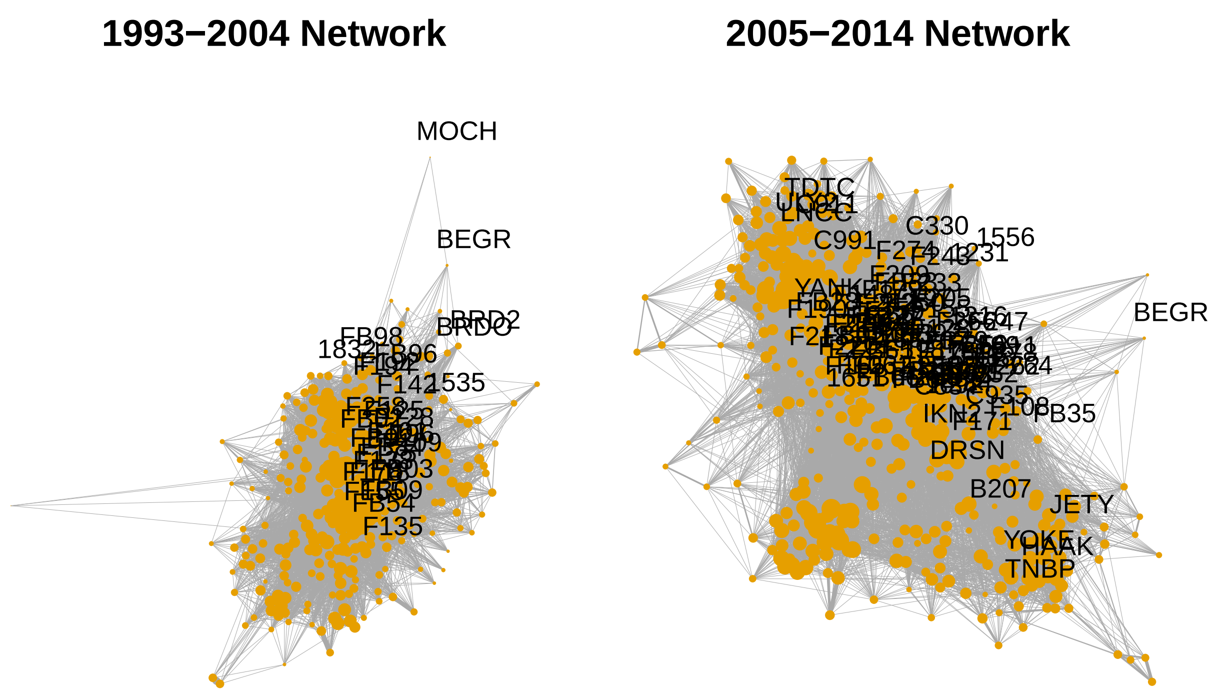
In this markdown I will:

1. Create the network structure from the association matrix.
2. Evaluate local and global network metrics.
3. Permutate the link weights using the WalkTrap algorithm.
4. Evaluate and plot modularity.

PART 1: ***Network Structure***

```
## Create social network
ig <- lapply(nxn, function (df) {
  graph_from_adjacency_matrix(
  df,
  mode = "undirected",
  weighted = TRUE,
  diag = FALSE)})

# Set the node names based on row names
row_names <- lapply(nxn, function (df) {rownames(df)})

for (i in seq_along(ig)) {
  V(ig[[i]])$name <- row_names[[i]]
}

## Only show IDs of HI dolphins
### subset_HI in "GLMM.R"
HI_data <-  diff_raw(subset_HI(list_years))
row_names_HI <- lapply(HI_data, function (df) {
  as.vector(df$Code[(df$DiffHI == "BG" | df$DiffHI == "SD" |
                                    df$DiffHI == "FG") & df$Freq > 0])})

# Plot network
# Set up the plotting area with 1 row and 2 columns for side-by-side plots
par(mfrow=c(1, 2), mar = c(0.5, 0.5, 0.5, 0.5))

main_labels <- c("1993-2004 Network", "2005-2014 Network")
```

```r
# Loop through the list of graphs and plot them side by side
for (i in 1:length(ig)) {
  plot(ig[[i]],
       layout = layout_with_fr(ig[[i]]),
       edge.width = E(ig[[i]])$weight * 4, # edge thickness
       vertex.size = sqrt(igraph::strength(ig[[i]], vids = V(ig[[i]]), mode = c("all"), loops = TRUE) *
       vertex.frame.color = NA,
       vertex.label.family = "Helvetica",
       vertex.label = ifelse(V(ig[[i]])$name %in% row_names_HI[[i]], V(ig[[i]])$name, NA),
       vertex.label.color = "black",
       vertex.label.cex = 0.8,
       vertex.label.dist = 2,
       vertex.frame.width = 0.01)
  # Add the main label above the plot
  title(main = main_labels[i], line = -1)
}
```




```r
# Reset the plotting area to its default configuration
par(mfrow=c(1, 1))
```

PART 2: *Network Metrics*

Local Network Metrics

```
# Edgelist: Nodes (i & j) and edge (or link) weight
el <- readRDS("../data/el_years.RData")

# Set the node names based on row names
get_names <- function (matrix, metric) {
  row_names <- lapply(matrix, function (df) {rownames(df)})
for (i in seq_along(metric)) {
  metric[[i]][,1] <- row_names[[i]]
}
  return(metric)
  }

# Weighted clustering coefficients
cluster <- readRDS("../data/cluster.RData")
cluster_diffs <- get_names(nxn, cluster)
cluster_diffs_HI <- lapply(seq_along(cluster_diffs), function(i) {
  df <- cluster_diffs[[i]]
  df_new <- as.data.frame(df[df[, 1] %in% row_names_HI[[i]], , drop = FALSE])
  return(df_new)
})
compare_cluster <- merge(
  cluster_diffs_HI[[1]][, c(1, 2)],
  cluster_diffs_HI[[2]][, c(1, 2)],
  by.x = "node",
  by.y = "node"
)
colnames(compare_cluster) <- c("ID", "Period.1", "Period.2")
compare_cluster[, c(2, 3)] <- sapply(compare_cluster[, c(2, 3)], as.numeric)
# Calculate differences
compare_cluster$Difference <- compare_cluster$Period.2 - compare_cluster$Period.1


# Betweenness centrality
between <- lapply(el, function (df) {betweenness_w(df, alpha=1)})
between_diffs <- get_names(nxn, between)
between_diffs_HI <- lapply(seq_along(between_diffs), function(i) {
  df <- between_diffs[[i]]
  df_new <- as.data.frame(df[df[, 1] %in% row_names_HI[[i]], , drop = FALSE])
  return(df_new)
})
compare_between <- merge(
  between_diffs_HI[[1]],
  between_diffs_HI[[2]],
  by.x = "node",
  by.y = "node"
)
colnames(compare_between) <- c("ID", "Period.1", "Period.2")
compare_between[, c(2, 3)] <- sapply(compare_between[, c(2, 3)], as.numeric)
```

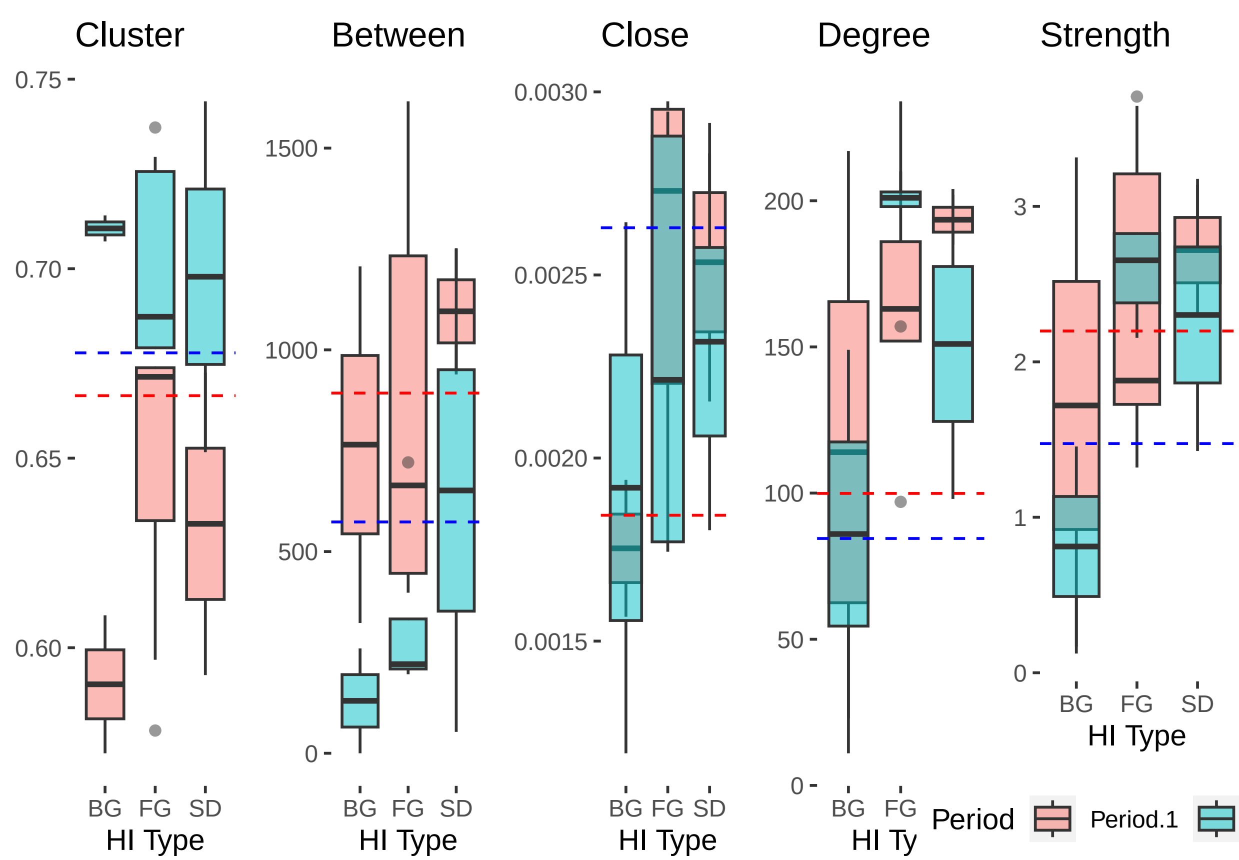
```
# Calculate differences
compare_between$Difference <- compare_between$Period.2 - compare_between$Period.1


# Closeness centrality
close <- lapply(el, function (df) {closeness_w(df, alpha=1)})
close_diffs <- get_names(nxn, close)
close_diffs_HI <- lapply(seq_along(close_diffs), function(i) {
  df <- close_diffs[[i]]
  df_new <- as.data.frame(df[df[, 1] %in% row_names_HI[[i]], , drop = FALSE])
  return(df_new)
})
compare_close <- merge(
  close_diffs_HI[[1]][, c(1, 2)],
  close_diffs_HI[[2]][, c(1, 2)],
  by.x = "node",
  by.y = "node"
)
colnames(compare_close) <- c("ID", "Period.1", "Period.2")
compare_close[, c(2, 3)] <- sapply(compare_close[, c(2, 3)], as.numeric)
# Calculate differences
compare_close$Difference <- compare_close$Period.2 - compare_close$Period.1


# Degree and strength centrality
strength <- lapply(el, function (df) {degree_w(df, measure=c("degree","output"), type="out", alpha=1)})
strength_diffs <- get_names(nxn, strength)
strength_diffs_HI <- lapply(seq_along(strength_diffs), function(i) {
  df <- strength_diffs[[i]]
  df_new <- as.data.frame(df[df[, 1] %in% row_names_HI[[i]], , drop = FALSE])
  return(df_new)
})
compare_strength <- merge(
  strength_diffs_HI[[1]],
  strength_diffs_HI[[2]],
  by.x = "node",
  by.y = "node"
)
colnames(compare_strength) <- c("ID", "Period.1_degree", "Period.1_strength", "Period.2_degree", "Perio
compare_strength[, c(2:5)] <- sapply(compare_strength[, c(2:5)], as.numeric)
# Calculate differences
compare_strength$Difference_degree <- compare_strength$Period.2_degree - compare_strength$Period.1_degre
compare_strength$Difference_strength <- compare_strength$Period.2_strength - compare_strength$Period.1_s

# Look at all of the local metrics together
## Add a column containing HI type
names_BG <- unlist(lapply(HI_data, function (df) {
  as.vector(df$Code[df$DiffHI == "BG" & df$Freq > 0])}))
names_SD <- unlist(lapply(HI_data, function (df) {
  as.vector(df$Code[df$DiffHI == "SD" & df$Freq > 0])}))
names_FG <- unlist(lapply(HI_data, function (df) {
  as.vector(df$Code[df$DiffHI == "FG" & df$Freq > 0])}))
```

```
HI_type <- ifelse(compare_cluster$ID %in% names_BG, "BG",
                  ifelse(compare_cluster$ID %in% names_SD, "SD",
                         ifelse(compare_cluster$ID %in% names_FG, "FG", "NA")))

# Combine the data
local_metrics_HI <- data.frame(ID = compare_cluster$ID, HI_type = HI_type,
  Period = c("Period.1", "Period.2"),
  Cluster = c(compare_cluster$Period.1, compare_cluster$Period.2),
  Between = c(compare_between$Period.1, compare_between$Period.2),
  Close = c(compare_close$Period.1, compare_close$Period.2),
  Degree = c(compare_strength$Period.1_degree, compare_strength$Period.2_degree),
  Strength = c(compare_strength$Period.1_strength, compare_strength$Period.2_strength))

## Add a rown to compare the averages of each metric with HI IDs
avg_metrics <- data.frame(ID = "Average", HI_type = "NA",
                          Period = c("Period.1", "Period.2"),
                          Cluster = c(mean(cluster[[2]][, 2]), mean(cluster[[1]][, 2])),
                          Between = c(mean(between[[2]][, 2]), mean(between[[1]][, 2])),
                          Close = c(mean(close[[2]][, 2]), mean(close[[1]][, 2])),
                          Degree = c(mean(strength[[2]][, 2]), mean(strength[[1]][, 2])),
                          Strength = c(mean(strength[[2]][, 3]), mean(strength[[1]][, 3])))

local_metrics_HI <- rbind(local_metrics_HI, avg_metrics)

# Reshape the data from wide to long format
local_metrics_HI <- melt(local_metrics_HI, id.vars = c("ID", "HI_type", "Period"), variable.name = "Met
colnames(local_metrics_HI) <- c("ID", "HI_type", "Period", "Metric", "value")

# Make sure metric is in character
local_metrics_HI$Metric <- as.character(local_metrics_HI$Metric)

# Get rid of the average values
local_met_HI <- local_metrics_HI[local_metrics_HI$HI_type != "NA", ]
```

Plot different metrics within HI types

```
# Plot for each Metric
plot_list <- list()
unique_metrics <- unique(local_met_HI$Metric)

for (i in seq_along(unique_metrics)) {
  metric <- unique_metrics[i]

  # Filter data for the current metric
  metric_data <- local_met_HI[local_met_HI$Metric == metric,]

  # Get the corresponding value for NA, Period.1 and the current metric
  value_na_period1 <- local_metrics_HI$value[local_metrics_HI$HI_type == "NA" &
                                             local_metrics_HI$Period == "Period.1" &
                                             local_metrics_HI$Metric == metric]
```

```
  # Get the corresponding value for NA, Period.2 and the current metric
  value_na_period2 <- local_metrics_HI$value[local_metrics_HI$HI_type == "NA" &
                                              local_metrics_HI$Period == "Period.2" &
                                              local_metrics_HI$Metric == metric]

  # Create the plot
  current_plot <- ggplot(metric_data, aes(x = HI_type, y = value, fill = Period)) +
    geom_boxplot(position = "identity", alpha = 0.5) +
    labs(x = "HI Type", y = NULL, fill = "Period") +
    ggtitle(paste(metric)) +
    theme(panel.background = element_blank()) +
    geom_hline(yintercept = value_na_period1, col = "red", linetype = "dashed") +
    geom_hline(yintercept = value_na_period2, col = "blue", linetype = "dashed")

  # Store the legend on the last plot
  if (i == length(unique_metrics)) {
    plot_list[[i]] <- current_plot + theme(legend.position = "bottom")
  } else {
    plot_list[[i]] <- current_plot + theme(legend.position = "none")
  }
}

# Arrange plots side by side
grid.arrange(grobs = plot_list, ncol = 5)
```



Results of local metrics after HAB event

- Local clustering coefficient: Measure of the prevalence of node clusters in a network (i.e. if dolphin associates all know each other, the clustering coefficient will be high).
- Average: The clustering coefficient has stayed roughly the same before and after HAB.
- Beggars: Large difference in clustering before and after the HAB event. Before the HAB beggars had a much lower than average coefficient and after it was much higher.
- Fixed Gear Foragers: Same clustering as average before and after HAB, but it increased after event.
- Scavengers/Depredators: Same result as the beggars but much closer to average clustering.
- Betweeness: A high betweeness means that the individual is in the communication path of other individuals, therefore, the individuals it interacts with, depend on its presence.
- Average: Node betweeness has decreased after HAB, therefore there is less connections between different clusters.
- Beggars: Equally in the communication path of others as average before HAB. Then much lower betweeness after HAB.
- Fixed Gear Foragers: Same results as beggars with less of a difference after HAB.
- Scavengers/Depredators: These individuals are in the communication path of other individuals before HAB, then less so after HAB.
- Closeness: The larger the closeness centrality is for an individual, the more rapidly and easily it can influence the behavior of others.
- Average: Individuals are much more rapidly and easily influencing the behavior of others.
- Beggars: Before the HAB, these individuals are within the average range of closeness. After HAB the individuals have less of an influence on others.
- Fixed Gear Foragers: These individuals have a high influence on others before HAB. Then they range in closeness more and are roughly equal to the average.
- Scavengers/Depredators: Same result as the FG.
- Degree: # Individual's associates
- Average: Individuals' number of associates remain the same while slightly decreasing after HAB.
- Beggars: These individuals have the same number of associates as the average population for before and after HAB.
- Fixed Gear Foragers: These individuals have many more associated than the average population before and after HAB, with a larger gap after HAB.
- Scavengers/Depredators: Same result as FG, but the # of associates decreases after HAB.
- Strength: Total strength of an individuals' associations
- Average: The strength of associations decrease after HAB.
- Beggars: These individuals' association strengths are consistently lower than average.
- Fixed Gear Foragers: The strength of individuals' associates is average before HAB. Then after HAB, the strength of associates increase.
- Scavengers/Depredators: Before and after HAB, the strengths of association is consistently higher than average.

Global Network Metrics

- Size: Number of nodes.
- Density/Connectance: Proportion of realized links (observed/possible links).
- Average Path Length (geodesic): Measures the shortest distance between two random nodes then average shortest pathways between all pairs of nodes. Shows how far apart any pair of individuals will be on average.
- Geodesic path: the shortest path through the network from one node to another (l).
- Diameter: Length of the longest geodesic path (d).
- Clustering coefficient: Tendency of nodes to cluster in the network (Are the friends' friends also friends?).

```
#' Breakdown: connectance = length(which(as.dist(orca_hwi)!=0))/(N*(N-1)/2)
## Calculate connectance for each matrix
calculate_connectance <- function(matrix) {
  N <- nrow(matrix)
  total <- N * (N - 1) / 2
  real <- sum(matrix != 0)  # Count non-zero elements
  connectance <- real / total
  return(connectance)
}

connectance_list <- lapply(nxn, calculate_connectance)
connectance_list
```

```
## [[1]]
## [1] 0.5375485
##
## [[2]]
## [1] 0.4575163
```

```
# Shortest path lengths (geodesics) and diameter
# # mean shortest path
dist <- lapply(ig, function(df) {mean_distance(df)})
dist
```

```
## [[1]]
## [1] 0.004317689
##
## [[2]]
## [1] 0.005595755
```

PART 3: *Permutate Link Weights*

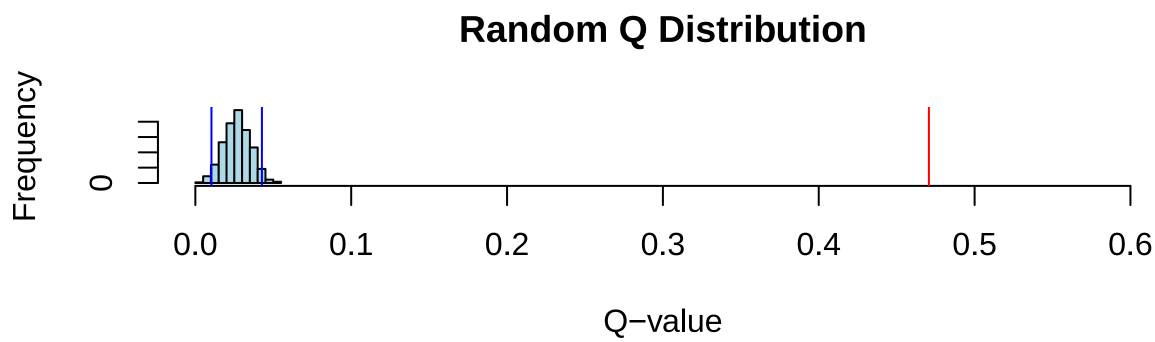
Walktrap algorithm breakdown with one interation

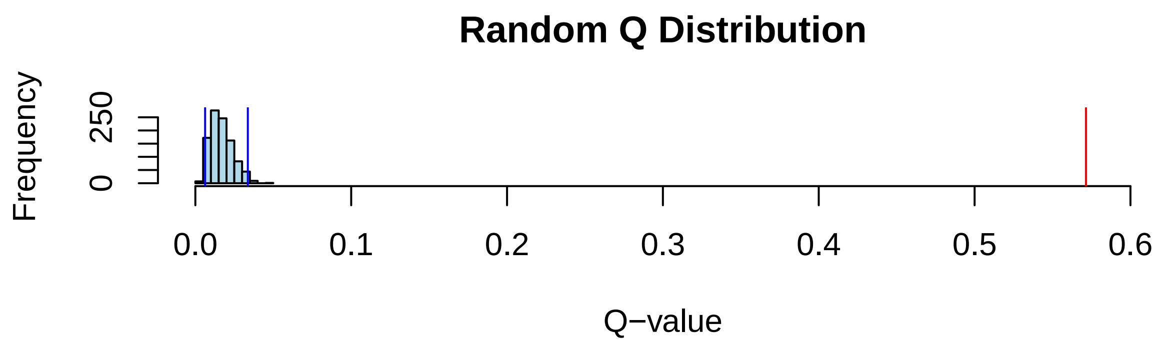
Permutate with multiple interations

```r
# Run modularity permutations 1000 times for each matrix
run_mod <- function(el_list, dolphin_walk_list) {
  iter <- 1000
  randmod <- numeric(iter)  # Initialize a numeric vector to store Q-values

  for (i in 1:iter) {
    # Save the edgelist into a new object and permutate the link weights
    auxrand <- el_list
    auxrand[, 2] <- sample(auxrand[, 2])

    # Create an igraph graph from the permuted edgelist
    igrand <- graph.edgelist(as.matrix(auxrand[, 1:2]), directed = FALSE)
    E(igrand)$weight <- auxrand[, 2]  # Assign link weights

    # Calculate modularity using walktrap community detection
    rand_walk <- walktrap.community(igrand)
    randmod[i] <- modularity(rand_walk)  # Save Q-value into the vector
  }

  # Calculate the 95% confidence interval (two-tailed test)
  ci <- quantile(randmod, probs = c(0.025, 0.975), type = 2)

  # Visualization of the random Q distribution
  hist(randmod, xlim = c(0, 0.6), main = "Random Q Distribution", xlab = "Q-value", ylab = "Frequency",

  # Empirical Q-value
  abline(v = modularity(dolphin_walk_list), col = "red")

  # 2.5% CI
  abline(v = ci[1], col = "blue")

  # 97.5% CI
  abline(v = ci[2], col = "blue")

  # Return a data frame with Q-value and confidence intervals
  result <- data.frame(Q = modularity(dolphin_walk_list), LowCI = ci[1], HighCI = ci[2])
  return(result)
}

# Plot null distributions for modularity
# Set up the plotting area with 2 rows and 2 column
par(mfrow=c(2, 1))
run_mod(el_list = el[[1]], dolphin_walk_list = dolphin_walk[[1]])
```

```
##              Q      LowCI     HighCI
## 2.5% 0.4706832 0.01033202 0.04271868
```

```r
run_mod(el_list = el[[2]], dolphin_walk_list = dolphin_walk[[2]])
```






```
##                Q      LowCI     HighCI
## 2.5% 0.5714733 0.00627028 0.03367213
```

We can reject the null hypothesis that individuals cluster at random and conclude that there is evidence that modularity is higher than what we would expect by chance.

PART 4: *Modularity*

- Newman's Q modularity: Stopping parameter Q removes links according to the betweenness.

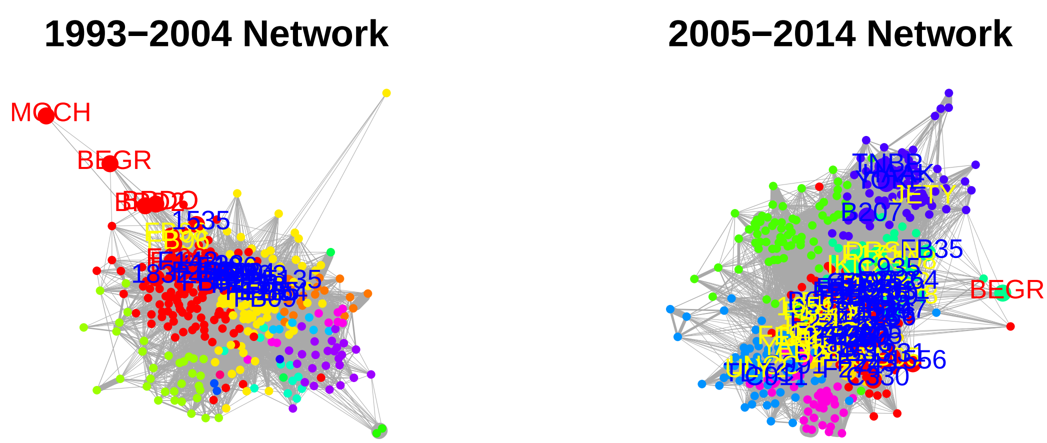
```r
# Create an unweighted network
system.time({
  registerDoParallel(n.cores)
  dolp_ig <- list()
  for (l in seq_along(list_years)) {
    dolp_ig[[l]] <- graph.edgelist(el[[l]][,1:2])
    # Add the edge weights to this network
    E(dolp_ig[[l]])$weight <- as.numeric(el[[l]][,3])
    # Create undirected network
    dolp_ig[[l]] <- as.undirected(dolp_ig[[l]])
  }
```

```
  ### End parallel processing
  stopImplicitCluster()
})
```

```
##    user  system elapsed
##    0.11    0.04    1.11
```

```
# Newman's Q modularity
newman <- lapply(dolp_ig, function (df) {cluster_leading_eigen(df, steps = -1, weights = E(df)$weight,
                                start = NULL, options = arpack_defaults, callback = NULL,
                                extra = NULL, env = parent.frame())})


# Set the node names based on row names
BG <- SD <- FG <- vector("list", length = length(dolp_ig))

for (i in seq_along(dolp_ig)) {
  # Set the node names
  V(dolp_ig[[i]])$name <- rownames(nxn[[i]])

  ## Parse out what HI behavior they engage in
  BG[[i]] <- as.vector(HI_data[[i]]$Code[HI_data[[i]]$DiffHI == "BG" & HI_data[[i]]$Freq > 0])
  SD[[i]] <- as.vector(HI_data[[i]]$Code[HI_data[[i]]$DiffHI == "SD" & HI_data[[i]]$Freq > 0])
  FG[[i]] <- as.vector(HI_data[[i]]$Code[HI_data[[i]]$DiffHI == "FG" & HI_data[[i]]$Freq > 0])

  ## Initialize label_color attribute for each node
  V(dolp_ig[[i]])$label_color <- "black"

  ## Make a different text color for each category
  V(dolp_ig[[i]])$label_color[V(dolp_ig[[i]])$name %in% BG[[i]]] <- "red"
  V(dolp_ig[[i]])$label_color[V(dolp_ig[[i]])$name %in% SD[[i]]] <- "yellow"
  V(dolp_ig[[i]])$label_color[V(dolp_ig[[i]])$name %in% FG[[i]]] <- "blue"
}

# Generate a vector of colors based on the number of unique memberships
for (i in seq_along(dolp_ig)) {
  V(dolp_ig[[i]])$color <- NA
  col <- rainbow(max(newman[[i]]$membership))

  for (j in 1:max(newman[[i]]$membership)){
    V(dolp_ig[[i]])$color[which(newman[[i]]$membership==j)] <- col[j]
  }
}

# Make sure the HI dolphins stand out
for (i in seq_along(dolp_ig)) {
  V(dolp_ig[[i]])$size <- ifelse(V(dolp_ig[[i]])$name %in% row_names_HI[[i]], 10, 5)
}

# Set up the plotting area with 1 row and 2 columns for side-by-side plots
par(mfrow=c(1, 2))
# Main labels for the plots
main_labels <- c("1993-2004 Network", "2005-2014 Network")  # Replace with appropriate main labels
```

```r
# Plot the graph with individual IDs as labels
for (i in seq_along(dolp_ig)) {
plot(dolp_ig[[i]],
     layout = layout_with_fr(dolp_ig[[i]]),
     # link weight, rescaled for better visualization
     edge.width= E(dolp_ig[[i]])$weight*4,
     # node size as degree (rescaled)
     vertex.size= V(dolp_ig[[i]])$size,
     vertex.frame.color= NA, #"black",
     vertex.label.family = "Helvetica",
     vertex.label=ifelse(V(dolp_ig[[i]])$name %in% row_names_HI[[i]], as.character(V(dolp_ig[[i]])$name
     vertex.label.color = V(dolp_ig[[i]])$label_color,
     vertex.label.cex=0.8,
     vertex.label.dist=0.5,
     # edge.curved=0,
     vertex.frame.width=0.01)
  # Add the main label above the plot
  title(main = main_labels[i], line = -1)
  }
```



Since these modules can represent functional units, I need to test which mechanisms drive the modular topology by creating null models.