Amazon Product Co-Purchasing Network Analysis

Mu Niu, Kai Wa Ho, Jingtang

2024-06-06

Introduction

In this project, we conduct a comprehensive social network analysis on a dataset derived from Amazon's product co-purchasing records. The raw edge data, which was collected on June 1, 2003, from the Amazon website, includes 403,394 nodes representing individual products and 3,387,388 unweighted directed edges indicating frequently co-purchased product pairs. The direction of an edge signifies the order of purchase, with an outward edge from product A to product B indicating that product B is frequently purchased after product A. This dataset is available at **Amazon Product Co-Purchasing Network.**

Additionally, the raw nodes data was collected by crawling Amazon's website and encompasses metadata and review information for 548,552 different products, including books, music, DVDs, and video tapes. This dataset provides detailed information such as the product title, sales rank, list of similar products, detailed product categorization, and reviews. This data was collected in the summer of 2006, and it can be accessed at **Amazon Product Metadata.**

After a thorough data cleaning process, we integrated the edge data and node information data, resulting in an igraph object containing 398,688 nodes. Each node has four attributes: ID (unique product ID), Title (product name), Group (class the product belongs to), and Category (subcategory of the class).

The primary objectives of this analysis are to understand the relationships between products, identify co-purchasing patterns and customer shopping behaviors through network data visualization, and explore various network, node, and edge metrics. We aim to interpret these metrics within the network context, apply community detection algorithms, and analyze the resulting communities. Additionally, we will examine the network's adjacency matrix, reorder vertices to highlight patterns, and observe changes in the matrix to identify clearer patterns based on community or connected components.

Methodology

In this project, we followed a structured approach to process the data, analyze the network, and visualize the results.The data processing procedures began with mapping the nodes information dataset, which was a text file with issues of missing data and inconsistent formats, into a dataframe with four columns: ID, Title, Group, and Category. Subsequently, we filtered out all edges in the edge dataset that connected nodes with missing IDs, product titles, or product groups. This ensured that only valid and complete data was included in the analysis. The mapped nodes information data and filtered edge data was then integrated into an igraph object containing 398,688 nodes with four vertex attributes: ID, Title, Group, and Category. The data cleaning process employed packages such as `dplyr` for data manipulation and `igraph` for constructing and managing the igraph object in R.

For sampling methodology, we generated 4 induced subgraphs by using a node-centric approach. For each sub-network, we randomly selected one node as the starting point and retrieved all nodes connected to it. We then iteratively expanded our sample by including nodes connected to the current set of nodes, continuing this process until the specified number of nodes was reached. This sampling method was designed to capture

densely connected nodes, facilitating better visualization and understanding of the relationships between products.

This comprehensive methodology enabled us to effectively analyze and interpret the Amazon product co-purchasing network, uncovering valuable insights into customer shopping behaviors and product relationships.

Analysis

After sampling four sub-networks, each formed by using our previously described sampling method and starting with nodes from the book, music, video, and DVD groups respectively, we visualized the subgraphs. This visualization was performed using the base plot and Plotly package in R to generate both static and interactive plots.

The visualizations revealed several interesting patterns in co-purchasing behavior. For instance, we observed that individuals who purchase education books also tend to listen to R&B, rap, and hip-hop music. This correlation might suggest a demographic overlap between students or educators and fans of these music genres. Additionally, we found that people who buy nonfiction books also favor history books, which is logical as both are grounded in factual content. Travel book enthusiasts were also seen to purchase audiobooks, likely due to the convenience of not carrying physical copies while traveling. Similarly, customers who buy children's books often buy audiobooks, possibly because audiobooks provide a convenient way for babysitters to engage children. In the DVD group, those who enjoy comedy DVDs also show a diverse taste in music, purchasing DJ music, pop, rock, and metal. An intriguing finding was that several memoir nodes were linked to R&B and DJ music, which might suggest that people prefer listening to these genres while reading memoirs. Additionally, we found that individuals interested in business and investment books also showed an interest in books on arts and photography.

These visualizations enhanced our understanding of the relationships between products and allowed us to identify distinct co-purchasing patterns. The interactive plots, containing more detailed information, can be accessed **here.** By analyzing these subgraphs, we gained valuable insights into customer behavior and product affinities, which can be leveraged for better marketing strategies and product recommendations.

Results

Conclusion

We analyzed the Amazon music product co-purchasing network to understand its structure and dynamics. Initially, we looked at the basic characteristics of the network, which showed moderate density with significant co-purchasing connections among music products. This setup allows us to see clear patterns and clusters, indicating that while many products are connected, they're not all directly linked.

Our study used community detection algorithms like Walktrap, Infomap, and FastGreedy to identify community structures within the network. Walktrap and Infomap were particularly good at finding smaller, closely-knit groups, showing specific buying behaviors among certain products. On the other hand, FastGreedy helped us see larger community structures, providing a broader view of how the network is organized.

We also used visualizations like reordered adjacency matrices to illustrate the network's dense connections and the formation of communities around frequently bought items. These visual tools confirmed that our community detection methods were effective and that the network structure was both robust and stable.

Our analysis of the Amazon music product co-purchasing network provides valuable insights for enhancing marketing strategies. We identified clusters and key products that illustrate how recommendations and purchases propagate through the network. This information is crucial for targeting marketing efforts effectively and optimizing product recommendations. The study underscores the importance of understanding community dynamics within these networks, offering essential guidance for developing targeted marketing approaches and improving recommendation systems in e-commerce and marketing.

References

Dataset:

- **Amazon Product Co-Purchasing Network:** The edge dataset representing products and their co-purchasing patterns on Amazon.
- **Amazon Product Metadata:** Metadata for products listed in the Amazon Product Co-Purchasing Network.

Academic Reference:

- J. Leskovec, L. Adamic, and B. Adamic. The Dynamics of Viral Marketing. ACM Transactions on the Web (ACM TWEB), 1(1), 2007.

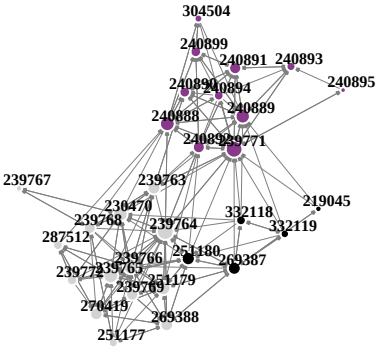
Acknowledgments:

We extend our heartfelt thanks to Isaiah Katz and Dr. Uma Ravat. Their assistance was invaluable, and we couldn't have completed this project without their support.

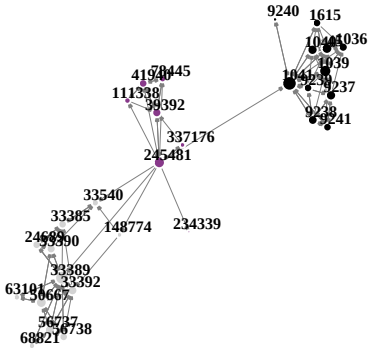
Appendices

- Figures

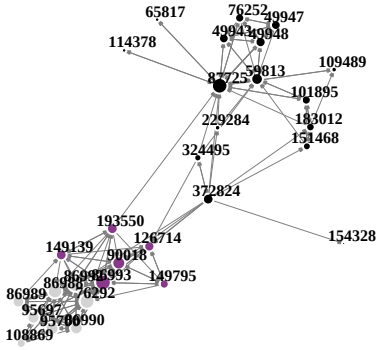
Music Products Community



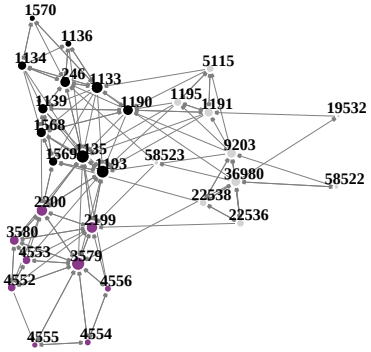
Book Products Community



Video Products Community



DVD Products Community



- Code

```
# Loading Library
library(readr)
library(igraph)
library(rsample)
library(dplyr)
library(stringr)
library(plotly)

#### Data Cleaning
# read data
meta <- readLines('../data/amazon-meta.txt')

# map into df with 5 columns: Id, Title, Group, Categories, Subcategory
meta <- data.frame(line = meta, stringsAsFactors = FALSE)

filtered_meta <- meta %>%
  mutate(keep = grepl("Id:|title:|group:|categories:", line) |
                lag(grepl("categories:", line), default = FALSE)) %>%
  filter(keep) %>%
  select(-keep)

# map to df
result <- data.frame(Id = character(),
                     Title = character(),
                     Group = character(),
                     Categories = character(),
                     Subcategory = character(),
                     stringsAsFactors = FALSE)


# Process the data
i <- 1
while (i <= nrow(filtered_meta)) {
  current_row <- filtered_meta$line[i]
  if (grepl("Id:   ", current_row)) {
    # Check if there are at least three more rows and they match the expected titles
    if (i + 4 <= nrow(filtered_meta) &&
        grepl("  title: ", filtered_meta$line[i + 1]) &&
        grepl("  group: ", filtered_meta$line[i + 2]) &&
        grepl("  categories: ", filtered_meta$line[i + 3])) {

      # Extract and clean the data
      id <- as.integer(sub("Id:   ", "", filtered_meta$line[i]))
      title <- sub("  title: ", "", filtered_meta$line[i + 1])
      group <- sub("  group: ", "", filtered_meta$line[i + 2])
      categories <- as.integer(sub("  categories: ", "", filtered_meta$line[i + 3]))
      if (categories == 0) {
        subcategory <- NA  # Return NA if categories are 0
      }
      else if (categories > 0){
        if (group %in% c("Music", "Book", "Video")){
          sub <- filtered_meta$line[i + 4]
```

```
          subcategory <- str_split(sub, "\\|")[[1]][4] %>% str_replace_all("\\[.*?\\]", "")
          if (subcategory %in% c('Reference', 'Genres')){
            subcategory <- str_split(sub, "\\|")[[1]][5] %>% str_replace_all("\\[.*?\\]", "")
          }
        }
        else if (group == "DVD"){
          sub <- filtered_meta$line[i + 4]
          subcategory <- str_split(sub, "\\|")[[1]][5] %>% str_replace_all("\\[.*?\\]", "")
        }
      }
      # Append to result dataframe
      result <- rbind(result, data.frame(Id = id,
                                         Title = title,
                                         Group = group,
                                         Categories = categories,
                                         Subcategory = subcategory,
                                         stringsAsFactors = FALSE))
      # Move index forward by 4
      i <- i + 5
    }
    else {
      # Skip the current Id and all following rows until next Id or end of data
      i <- i + 1
      while(i <= nrow(filtered_meta) && !grepl("Id:  ", filtered_meta$line[i])) {
        i <- i + 1
      }
    }
  }
}

# write to csv
result %>% write.csv('../data/meta_data.csv')

# read edges data
meta = read.csv('../data/meta_data.csv')
data = read.table("../data/amazon0601.txt")
colnames(data) = c("From", "To")

# keep edges that link nodes we have info on
filtered_data = data[(data$From %in% meta$Id) & (data$To %in% meta$Id), ]

# write to csv
filtered_data %>% write.csv('../data/filtered_data.csv')

# convert to igraph
amz <- graph_from_data_frame(filtered_data, directed = TRUE)

# create attributes for igraph object from meta data
title = c()
group = c()
sub = c()

for (i in V(amz)$name){
```

```
  if (as.integer(i) %in% meta$Id){
    title = append(title, meta[meta$Id == as.integer(i), ]$Title)
    group = append(group, meta[meta$Id == as.integer(i), ]$Group)
    sub = append(sub, meta[meta$Id == as.integer(i), ]$Subcategory)
  }
}

# handle missing value
sub[is.na(sub)] <- "missing"

V(amz)$title <- title
V(amz)$group <- group
V(amz)$sub <- sub

# save igraph object
saveRDS(amz, file = '../data/amz_igraph.rds')

amz <- readRDS(file = '../data/amz_igraph.rds')

#### Sampling Method
# Function to retrieve connected nodes up to a given count
retrieve_connected_nodes <- function(graph, start_node, count = 30) {
  # Initialize the list with the start node
  nodes_to_explore <- list(start_node)
  connected_nodes <- c(start_node)

  # Keep a list to avoid revisiting nodes
  visited_nodes <- numeric(0)

  # Explore the graph until we reach the desired number of nodes
  while (length(nodes_to_explore) > 0 && length(connected_nodes) < count) {
    current_node <- nodes_to_explore[[1]]
    nodes_to_explore <- nodes_to_explore[-1]  # Remove the explored node

    # Skip if already visited
    if (current_node %in% visited_nodes) next

    # Mark as visited
    visited_nodes <- c(visited_nodes, current_node)

    # Get neighbors and add to nodes to explore
    neighbors <- neighbors(graph, current_node)
    new_neighbors <- neighbors[!neighbors %in% connected_nodes]
    nodes_to_explore <- c(nodes_to_explore, as.list(new_neighbors))
    connected_nodes <- c(connected_nodes, new_neighbors)

    # Limit the collection if it exceeds the desired count
    if (length(connected_nodes) > count) {
      connected_nodes <- connected_nodes[1:count]
      break
    }
  }
```

```
  # Return the vertex sequence of connected nodes
  return(connected_nodes)
}

#### Visualization
# Define a function to generate and plot the community for a given product group
plot_product_community <- function(group_name, main_title) {
  set.seed(194)
  # Randomly select one node from the specified group
  random_node <- sample(V(amz)[V(amz)$group == group_name], 1)

  # Apply function to get 200 related nodes
  nodes <- retrieve_connected_nodes(amz, random_node)
  network <- induced_subgraph(amz, nodes)

  # Choose a layout that spreads out the nodes more effectively
  layout <- layout_with_fr(network)

  # Set graph margins to zero
  par(mar = c(0, 0, 2, 0))

  # Base plot
  plot(network, layout = layout,
       vertex.color = "#88398A",
       vertex.frame.color = "#FFFFFF",
       vertex.size = 5,
       vertex.label = V(network)$name,
       vertex.label.dist = 1,
       vertex.label.cex = 0.8,
       vertex.label.color = "black",
       vertex.label.font = 2,
       edge.color = "gray50",
       edge.width = 0.2,
       edge.arrow.size = 0.1,
       main = paste(main_title, "Base Plot"),
       bg = "white"
  )

  # Community plot
  cluster <- cluster_optimal(network)
  mycomcols <- c("black", "#D3D3D3", "#88398A")

  plot(network, layout = layout,
       vertex.color = mycomcols[cluster$membership],
       vertex.frame.color = "#FFFFFF",
       vertex.size = sqrt(degree(network)) * 2,
       vertex.label = V(network)$name,
       vertex.label.dist = 1,
       vertex.label.cex = 0.6,
       vertex.label.color = "black",
       vertex.label.font = 2,
       edge.color = "gray50",
       edge.width = 0.2,
```

```
      edge.arrow.size = 0.1,
      main = paste(main_title, "Community"),
      bg = "white"
)

# Get vertex data including the degree for size scaling
vertex_data <- data.frame(
  Id = V(network)$name,
  x = layout[, 1],
  y = layout[, 2],
  degree = degree(network),
  Title = V(network)$title,
  Group = V(network)$group,
  Category = V(network)$sub
)

# Enhance hover info by including all attributes except x, y coordinates
vertex_data$hoverinfo <- apply(vertex_data[, -c(2, 3)], 1, function(row) {
  paste(names(row), row, sep=": ", collapse="<br>")
})

# Get edge data
edge_data <- get.data.frame(network, what = "edges")

# Join edge data with vertex data to get coordinates for 'from' and 'to'
edge_data <- merge(edge_data, vertex_data, by.x = "from", by.y = "Id", all.x = TRUE)
edge_data <- merge(edge_data, vertex_data, by.x = "to", by.y = "Id", all.x = TRUE, suffixes = c(".fro

# Prepare data for Plotly plot
edges <- list(
  x = c(rbind(edge_data$x.from, edge_data$x.to, NA)),
  y = c(rbind(edge_data$y.from, edge_data$y.to, NA)),
  type = "scatter",
  mode = "lines",
  line = list(color = "grey", width = 0.5)
)

nodes <- list(
  x = vertex_data$x,
  y = vertex_data$y,
  hovertext = vertex_data$hoverinfo,
  mode = "markers",
  marker = list(size = vertex_data$degree * 2,
                color = mycomcols[cluster$membership]),
  type = "scatter",
  hoverinfo = "text"
)

# Create the plot
plot_ly() %>%
  add_trace(x = edges$x, y = edges$y, mode = edges$mode, type = edges$type, line = edges$line) %>%
  add_trace(x = nodes$x, y = nodes$y, hovertext = nodes$hovertext, mode = nodes$mode, type = nodes$ty
  layout(
```

```
      title = paste("Network Visualization of", main_title),
      xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE),
      yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE),
      hovermode = 'closest'
    )
}

# Music group
plot_product_community("Music", "Music Products")

# Book group
plot_product_community("Book", "Book Products")

# Video group
plot_product_community("Video", "Video Products")

# DVD group
plot_product_community("DVD", "DVD Products")

#### Network Metrics
## Music Products
# Density
network_density <- edge_density(music.network)
cat("Network Density:", network_density, "\n")

# Average Path Length
avg_path_length <- average.path.length(music.network, directed = FALSE)
cat("Average Path Length:", avg_path_length, "\n")

# Diameter
network_diameter <- diameter(music.network, directed = FALSE)
cat("Network Diameter:", network_diameter, "\n")

# Node Metrics
# Degree
node_degree <- degree(music.network)
cat("Node Degree:\n")
print(summary(node_degree))

# Betweenness Centrality
node_betweenness <- betweenness(music.network)
cat("Node Betweenness Centrality:\n")
print(summary(node_betweenness))

# Closeness Centrality
node_closeness <- closeness(music.network)
cat("Node Closeness Centrality:\n")
print(summary(node_closeness))

# Eigenvector Centrality
node_eigenvector <- evcent(music.network)$vector
cat("Node Eigenvector Centrality:\n")
print(summary(node_eigenvector))
```

```r
# Edge Metrics
# Edge Betweenness
edge_betweenness <- edge.betweenness(music.network)
cat("Edge Betweenness:\n")
print(summary(edge_betweenness))

## Book Products
# Density
network_density <- edge_density(book.network)
cat("Network Density:", network_density, "\n")

# Average Path Length
avg_path_length <- average.path.length(book.network, directed = FALSE)
cat("Average Path Length:", avg_path_length, "\n")

# Diameter
network_diameter <- diameter(book.network, directed = FALSE)
cat("Network Diameter:", network_diameter, "\n")

# Node Metrics
# Degree
node_degree <- degree(book.network)
cat("Node Degree:\n")
print(summary(node_degree))

# Betweenness Centrality
node_betweenness <- betweenness(book.network)
cat("Node Betweenness Centrality:\n")
print(summary(node_betweenness))

# Closeness Centrality
node_closeness <- closeness(book.network)
cat("Node Closeness Centrality:\n")
print(summary(node_closeness))

# Eigenvector Centrality
node_eigenvector <- evcent(book.network)$vector
cat("Node Eigenvector Centrality:\n")
print(summary(node_eigenvector))

# Edge Metrics
# Edge Betweenness
edge_betweenness <- edge.betweenness(book.network)
cat("Edge Betweenness:\n")
print(summary(edge_betweenness))

## Video Products
# Network Metrics
# Density
network_density <- edge_density(video.network)
cat("Network Density:", network_density, "\n")

# Average Path Length
```

```r
avg_path_length <- average.path.length(video.network, directed = FALSE)
cat("Average Path Length:", avg_path_length, "\n")

# Diameter
network_diameter <- diameter(video.network, directed = FALSE)
cat("Network Diameter:", network_diameter, "\n")

# Node Metrics
# Degree
node_degree <- degree(video.network)
cat("Node Degree:\n")
print(summary(node_degree))

# Betweenness Centrality
node_betweenness <- betweenness(video.network)
cat("Node Betweenness Centrality:\n")
print(summary(node_betweenness))

# Closeness Centrality
node_closeness <- closeness(video.network)
cat("Node Closeness Centrality:\n")
print(summary(node_closeness))

# Eigenvector Centrality
node_eigenvector <- evcent(video.network)$vector
cat("Node Eigenvector Centrality:\n")
print(summary(node_eigenvector))

# Edge Metrics
# Edge Betweenness
edge_betweenness <- edge.betweenness(video.network)
cat("Edge Betweenness:\n")
print(summary(edge_betweenness))

## DVD Products
network_density <- edge_density(dvd.network)
cat("Network Density:", network_density, "\n")

# Average Path Length
avg_path_length <- average.path.length(dvd.network, directed = FALSE)
cat("Average Path Length:", avg_path_length, "\n")

# Diameter
network_diameter <- diameter(dvd.network, directed = FALSE)
cat("Network Diameter:", network_diameter, "\n")

# Node Metrics
# Degree
node_degree <- degree(dvd.network)
cat("Node Degree:\n")
print(summary(node_degree))

# Betweenness Centrality
```

```r
node_betweenness <- betweenness(dvd.network)
cat("Node Betweenness Centrality:\n")
print(summary(node_betweenness))

# Closeness Centrality
node_closeness <- closeness(dvd.network)
cat("Node Closeness Centrality:\n")
print(summary(node_closeness))

# Eigenvector Centrality
node_eigenvector <- evcent(dvd.network)$vector
cat("Node Eigenvector Centrality:\n")
print(summary(node_eigenvector))

# Edge Metrics
# Edge Betweenness
edge_betweenness <- edge.betweenness(dvd.network)
cat("Edge Betweenness:\n")
print(summary(edge_betweenness))

#### Community Detection
## Music
# Walktrap Algorithm
walktrap_communities <- cluster_walktrap(music.network)
cat("Walktrap Algorithm:\n")
print(membership(walktrap_communities))
cat("Modularity:", modularity(walktrap_communities), "\n")

# Infomap Algorithm
infomap_communities <- cluster_infomap(music.network)
cat("Infomap Algorithm:\n")
print(membership(infomap_communities))
cat("Modularity:", modularity(infomap_communities), "\n")

# Visualize communities
par(mfrow=c(1,3))
plot(walktrap_communities, music.network, main="Walktrap Algorithm", vertex.size=10, edge.arrow.size=0.
plot(infomap_communities, music.network, main="Infomap Algorithm", vertex.size=10, edge.arrow.size=0.05

## Book
# Walktrap Algorithm
walktrap_communities <- cluster_walktrap(book.network)
cat("Walktrap Algorithm:\n")
print(membership(walktrap_communities))
cat("Modularity:", modularity(walktrap_communities), "\n")

# Infomap Algorithm
infomap_communities <- cluster_infomap(book.network)
cat("Infomap Algorithm:\n")
print(membership(infomap_communities))
cat("Modularity:", modularity(infomap_communities), "\n")

# Visualize communities
```

```r
par(mfrow=c(1,2))
plot(walktrap_communities, book.network, main="Walktrap Algorithm", vertex.size=10, edge.arrow.size=0.05
plot(infomap_communities, book.network, main="Infomap Algorithm", vertex.size=10, edge.arrow.size=0.05,

## Video
# Walktrap Algorithm
walktrap_communities <- cluster_walktrap(video.network)
cat("Walktrap Algorithm:\n")
print(membership(walktrap_communities))
cat("Modularity:", modularity(walktrap_communities), "\n")

# Infomap Algorithm
infomap_communities <- cluster_infomap(video.network)
cat("Infomap Algorithm:\n")
print(membership(infomap_communities))
cat("Modularity:", modularity(infomap_communities), "\n")

# Visualize communities
par(mfrow=c(1,3))
plot(walktrap_communities, video.network, main="Walktrap Algorithm", vertex.size=10, edge.arrow.size=0.0
plot(infomap_communities, video.network, main="Infomap Algorithm", vertex.size=10, edge.arrow.size=0.05

## DVD
# Walktrap Algorithm
walktrap_communities_dvd <- cluster_walktrap(dvd.network)
cat("Walktrap Algorithm:\n")
print(membership(walktrap_communities_dvd))
cat("Modularity:", modularity(walktrap_communities_dvd), "\n")

# Infomap Algorithm
infomap_communities_dvd <- cluster_infomap(dvd.network)
cat("Infomap Algorithm:\n")
print(membership(infomap_communities_dvd))
cat("Modularity:", modularity(infomap_communities_dvd), "\n")

# Visualize communities
plot(walktrap_communities_dvd, dvd.network, main="Walktrap Algorithm", vertex.size=10, edge.arrow.size=0
plot(infomap_communities_dvd, dvd.network, main="Infomap Algorithm", vertex.size=10, edge.arrow.size=0.0

#### Adjacency Matrix
# Create the adjacency matrix
adj_matrix <- as_adjacency_matrix(music.network, sparse = FALSE)

# Transpose the adjacency matrix to swap rows and columns
adj_matrix_transposed <- t(adj_matrix)

# Set up the plot with customizations
par(mar = c(5, 5, 4, 2) + 0.1)  # Increase margins for axis labels

# Visualize the transposed adjacency matrix with gridlines and node labels
image(1:nrow(adj_matrix_transposed), 1:ncol(adj_matrix_transposed), adj_matrix_transposed,
      main = "Adjacency Matrix For Music",
      xlab = "Nodes", ylab = "Nodes",
```

```
      axes = FALSE, col = c("white", "black"))

# Add gridlines
grid(nx = nrow(adj_matrix_transposed), ny = ncol(adj_matrix_transposed), col = "gray", lty = "dotted")

# Add axis labels with the correct node names
axis(1, at = 1:nrow(adj_matrix_transposed), labels = rownames(adj_matrix_transposed), las = 2, cex.axis
axis(2, at = 1:ncol(adj_matrix_transposed), labels = colnames(adj_matrix_transposed), las = 2, cex.axis

#### Re-ordering
# Hierarchical clustering to reorder the adjacency matrix
d <- dist(adj_matrix) # Distance matrix
hc <- hclust(d)       # Hierarchical clustering
order <- hc$order     # Order of the nodes

# Reorder the adjacency matrix
adj_matrix_reordered <- adj_matrix[order, order]

# Set up the plot with customizations
par(mar = c(5, 5, 4, 2) + 0.1)  # Increase margins for axis labels

# Plot the reordered adjacency matrix with gridlines and node labels
image(1:nrow(adj_matrix_reordered), 1:ncol(adj_matrix_reordered), adj_matrix_reordered,
      main = "Reordered Adjacency Matrix For Music",
      xlab = "Nodes", ylab = "Nodes",
      axes = FALSE, col = c("lightpink", "darkblue"))

# Add gridlines
grid(nx = nrow(adj_matrix_reordered), ny = ncol(adj_matrix_reordered), col = "gray", lty = "dotted")

# Add axis labels
axis(1, at = 1:nrow(adj_matrix_reordered), labels = rownames(adj_matrix_reordered), las = 2, cex.axis =
axis(2, at = 1:ncol(adj_matrix_reordered), labels = colnames(adj_matrix_reordered), las = 2, cex.axis =

## Optimal
modularity_value <- modularity(music.cluster)
modularity_value

membership <- membership(music.cluster)

# Reorder the nodes based on community membership
ordered_indices <- order(membership)
ordered_adjacency_matrix <- as.matrix(get.adjacency(music.network))[ordered_indices, ordered_indices]

# Plot the reordered adjacency matrix
image(t(ordered_adjacency_matrix), main="Reordered Adjacency Matrix for Music Network(Optimal)", xlab="
axis(1, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(music.network)$name[ordered_indices]
axis(2, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(music.network)$name[ordered_indices]

## Walktrap
communities <- cluster_walktrap(music.network)

# Get the membership vector
```

```r
membership <- membership(communities)

# Calculate modularity to quantify the strength of the detected communities
modularity_value <- modularity(communities)
print(paste("Modularity:", modularity_value))

# Identify the number of communities
num_communities <- length(unique(membership))
print(paste("Number of Communities:", num_communities))

# List the members of each community
community_list <- split(V(music.network)$name, membership)
community_list

# Reorder the nodes based on community membership
ordered_indices <- order(membership)
ordered_adjacency_matrix <- as.matrix(get.adjacency(music.network))[ordered_indices, ordered_indices]

# Plot the reordered adjacency matrix
image(t(ordered_adjacency_matrix), main="Reordered Adjacency Matrix for Music Network(Walktrap)", xlab=
axis(1, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(music.network)$name[ordered_indices]
axis(2, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(music.network)$name[ordered_indices]

## Undirected
# change the network to undirected
undirected_music_network <- as.undirected(music.network, mode="collapse")

# Detect communities using the walktrap algorithm
communities <- cluster_fast_greedy(undirected_music_network)

membership <- membership(communities)

# Calculate modularity to quantify the strength of the detected communities
modularity_value <- modularity(communities)
print(paste("Modularity:", modularity_value))

# Identify the number of communities
num_communities <- length(unique(membership))
print(paste("Number of Communities:", num_communities))

# List the members of each community
community_list <- split(V(undirected_music_network)$name, membership)
community_list



# Reorder nodes based on community membership
ordered_indices <- order(membership)
ordered_adjacency_matrix <- t(as.matrix(get.adjacency(undirected_music_network))[ordered_indices, ordere

# Plot reordered adjacency matrix with community colors
image(1:nrow((ordered_adjacency_matrix)), 1:ncol(ordered_adjacency_matrix), ordered_adjacency_matrix, c
axis(1, at=1:nrow(ordered_adjacency_matrix), labels=V(undirected_music_network)$name[ordered_indices],
```

```r
axis(2, at=1:ncol(ordered_adjacency_matrix), labels=V(undirected_music_network)$name[ordered_indices], l


## Book Original Matrix
edges_list_B <- as_edgelist(book.network, names = TRUE)


igraph_network_B <- graph.edgelist(edges_list_B, directed = TRUE)

network.adjacency_B <- as_adj(igraph_network_B)

network.adjacency_B

image(Matrix(network.adjacency_B))

# Create the adjacency matrix
adj_matrix <- as_adjacency_matrix(book.network, sparse = FALSE)

# Set up the plot with customizations
par(mar = c(5, 5, 4, 2) + 0.1)  # Increase margins for axis labels

# Visualize the adjacency matrix with gridlines and node labels
image(1:nrow(adj_matrix), 1:ncol(adj_matrix), adj_matrix,
      main = "Adjacency Matrix For Books",
      xlab = "Nodes", ylab = "Nodes",
      axes = FALSE, col = c("white", "black"))


# Add axis labels
axis(1, at = 1:nrow(adj_matrix), labels = rownames(adj_matrix), las = 2, cex.axis = 0.7, tick = FALSE)
axis(2, at = 1:ncol(adj_matrix), labels = colnames(adj_matrix), las = 2, cex.axis = 0.7, tick = FALSE)

### Reorder vertices to highlight patterns.
# Hierarchical clustering to reorder the adjacency matrix
d <- dist(adj_matrix) # Distance matrix
hc <- hclust(d)       # Hierarchical clustering
order <- hc$order     # Order of the nodes

# Reorder the adjacency matrix
adj_matrix_reordered <- adj_matrix[order, order]

# Set up the plot with customizations
par(mar = c(5, 5, 4, 2) + 0.1)  # Increase margins for axis labels

# Plot the reordered adjacency matrix with gridlines and node labels
image(1:nrow(adj_matrix_reordered), 1:ncol(adj_matrix_reordered), adj_matrix_reordered,
      main = "Reordered Adjacency Matrix For Books",
      xlab = "Nodes", ylab = "Nodes",
      axes = FALSE, col = c("lightpink", "darkblue"))

# Add gridlines
grid(nx = nrow(adj_matrix_reordered), ny = ncol(adj_matrix_reordered), col = "gray", lty = "dotted")
```

```r
# Add axis labels
axis(1, at = 1:nrow(adj_matrix_reordered), labels = rownames(adj_matrix_reordered), las = 2, cex.axis =
axis(2, at = 1:ncol(adj_matrix_reordered), labels = colnames(adj_matrix_reordered), las = 2, cex.axis =

# Detect communities
communities <- cluster_walktrap(book.network)

# Get the membership vector
membership <- membership(communities)

# Calculate modularity to quantify the strength of the detected communities
modularity_value <- modularity(communities)
print(paste("Modularity:", modularity_value))

# Identify the number of communities
num_communities <- length(unique(membership))
print(paste("Number of Communities:", num_communities))

# List the members of each community
community_list <- split(V(book.network)$name, membership)
community_list

# Reorder the nodes based on community membership
ordered_indices <- order(membership)
ordered_adjacency_matrix <- as.matrix(get.adjacency(book.network))[ordered_indices, ordered_indices]

# Plot the reordered adjacency matrix
image(ordered_adjacency_matrix, main="Reordered Adjacency Matrix Based on Community for Book Network", 
axis(1, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(book.network)$name[ordered_indices],

# change the network to undirected
undirected_book_network <- as.undirected(book.network, mode="collapse")

# Detect communities using the fastgreedy algorithm
communities <- cluster_fast_greedy(undirected_book_network)

# Get the membership vector
membership <- membership(communities)

# Calculate modularity to quantify the strength of the detected communities
modularity_value <- modularity(communities)
print(paste("Modularity:", modularity_value))

# Identify the number of communities
num_communities <- length(unique(membership))
print(paste("Number of Communities:", num_communities))

# List the members of each community
community_list <- split(V(undirected_book_network)$name, membership)
community_list

# Reorder the nodes based on community membership
ordered_indices <- order(membership)
```

```r
ordered_adjacency_matrix <- as.matrix(get.adjacency(undirected_book_network))[ordered_indices, ordered_

# Plot the reordered adjacency matrix
image(ordered_adjacency_matrix, main="Reordered Adjacency Matrix Based on Community", xlab="Nodes", ylab
axis(1, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(undirected_book_network)$name[ordered
axis(2, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(undirected_book_network)$name[ordered

## Video
edges_list_V <- as_edgelist(video.network, names = TRUE)


igraph_network_V <- graph.edgelist(edges_list_V, directed = TRUE)

network.adjacency_V <- as_adj(igraph_network_V)

network.adjacency_V

image(Matrix(network.adjacency_V))

# Create the adjacency matrix
adj_matrix <- as_adjacency_matrix(video.network, sparse = FALSE)

# Set up the plot with customizations
par(mar = c(5, 5, 4, 2) + 0.1)  # Increase margins for axis labels

# Visualize the adjacency matrix with gridlines and node labels
image(1:nrow(adj_matrix), 1:ncol(adj_matrix), adj_matrix,
      main = "Adjacency Matrix For Videos",
      xlab = "Nodes", ylab = "Nodes",
      axes = FALSE, col = c("white", "black"))


# Add axis labels
axis(1, at = 1:nrow(adj_matrix), labels = rownames(adj_matrix), las = 2, cex.axis = 0.7, tick = FALSE)
axis(2, at = 1:ncol(adj_matrix), labels = colnames(adj_matrix), las = 2, cex.axis = 0.7, tick = FALSE)

# Hierarchical clustering to reorder the adjacency matrix
d <- dist(adj_matrix) # Distance matrix
hc <- hclust(d)       # Hierarchical clustering
order <- hc$order     # Order of the nodes

# Reorder the adjacency matrix
adj_matrix_reordered <- adj_matrix[order, order]

# Set up the plot with customizations
par(mar = c(5, 5, 4, 2) + 0.1)  # Increase margins for axis labels

# Plot the reordered adjacency matrix with gridlines and node labels
image(1:nrow(adj_matrix_reordered), 1:ncol(adj_matrix_reordered), adj_matrix_reordered,
      main = "Reordered Adjacency Matrix For Videos",
      xlab = "Nodes", ylab = "Nodes",
      axes = FALSE, col = c("lightpink", "darkblue"))
```

```r
# Add gridlines
grid(nx = nrow(adj_matrix_reordered), ny = ncol(adj_matrix_reordered), col = "gray", lty = "dotted")

# Add axis labels
axis(1, at = 1:nrow(adj_matrix_reordered), labels = rownames(adj_matrix_reordered), las = 2, cex.axis =
axis(2, at = 1:ncol(adj_matrix_reordered), labels = colnames(adj_matrix_reordered), las = 2, cex.axis =

# Detect communities
communities <- cluster_walktrap(video.network)

# Get the membership vector
membership <- membership(communities)

# Calculate modularity to quantify the strength of the detected communities
modularity_value <- modularity(communities)
print(paste("Modularity:", modularity_value))

# Identify the number of communities
num_communities <- length(unique(membership))
print(paste("Number of Communities:", num_communities))

# List the members of each community
community_list <- split(V(video.network)$name, membership)
community_list

# Reorder the nodes based on community membership
ordered_indices <- order(membership)
ordered_adjacency_matrix <- as.matrix(get.adjacency(video.network))[ordered_indices, ordered_indices]

# Plot the reordered adjacency matrix
image(ordered_adjacency_matrix, main="Reordered Adjacency Matrix Based on Community for Video Network",
axis(1, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(video.network)$name[ordered_indices]
axis(2, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(video.network)$name[ordered_indices]

# change the network to undirected
undirected_video_network <- as.undirected(video.network, mode="collapse")

# Detect communities using the fastgreedy algorithm
communities <- cluster_fast_greedy(undirected_video_network)

# Get the membership vector
membership <- membership(communities)

# Calculate modularity to quantify the strength of the detected communities
modularity_value <- modularity(communities)
print(paste("Modularity:", modularity_value))

# Identify the number of communities
num_communities <- length(unique(membership))
print(paste("Number of Communities:", num_communities))

# List the members of each community
community_list <- split(V(undirected_video_network)$name, membership)
```

```
community_list

# Reorder the nodes based on community membership
ordered_indices <- order(membership)
ordered_adjacency_matrix <- as.matrix(get.adjacency(undirected_video_network))[ordered_indices, ordered_

# Plot the reordered adjacency matrix
image(ordered_adjacency_matrix, main="Reordered Adjacency Matrix Based on Community", xlab="Nodes", ylab
axis(1, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(undirected_video_network)$name[ordere
axis(2, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(undirected_video_network)$name[ordere

edges_list_D <- as_edgelist(dvd.network, names = TRUE)


igraph_network_D <- graph.edgelist(edges_list_D, directed = TRUE)

network.adjacency_D <- as_adj(igraph_network_D)

network.adjacency_D

image(Matrix(network.adjacency_D))

# Create the adjacency matrix
adj_matrix <- as_adjacency_matrix(dvd.network, sparse = FALSE)

# Set up the plot with customizations
par(mar = c(5, 5, 4, 2) + 0.1)  # Increase margins for axis labels

# Visualize the adjacency matrix with gridlines and node labels
image(1:nrow(adj_matrix), 1:ncol(adj_matrix), adj_matrix,
      main = "Adjacency Matrix For DVD",
      xlab = "Nodes", ylab = "Nodes",
      axes = FALSE, col = c("white", "black"))



# Add axis labels
axis(1, at = 1:nrow(adj_matrix), labels = rownames(adj_matrix), las = 2, cex.axis = 0.7, tick = FALSE)
axis(2, at = 1:ncol(adj_matrix), labels = colnames(adj_matrix), las = 2, cex.axis = 0.7, tick = FALSE)

### Reorder vertices to highlight patterns.
# Hierarchical clustering to reorder the adjacency matrix
d <- dist(adj_matrix) # Distance matrix
hc <- hclust(d)       # Hierarchical clustering
order <- hc$order     # Order of the nodes

# Reorder the adjacency matrix
adj_matrix_reordered <- adj_matrix[order, order]

# Set up the plot with customizations
par(mar = c(5, 5, 4, 2) + 0.1)  # Increase margins for axis labels

# Plot the reordered adjacency matrix with gridlines and node labels
```

```r
image(1:nrow(adj_matrix_reordered), 1:ncol(adj_matrix_reordered), adj_matrix_reordered,
      main = "Reordered Adjacency Matrix For DVD",
      xlab = "Nodes", ylab = "Nodes",
      axes = FALSE, col = c("lightpink", "darkblue"))

# Add gridlines
grid(nx = nrow(adj_matrix_reordered), ny = ncol(adj_matrix_reordered), col = "gray", lty = "dotted")

# Add axis labels
axis(1, at = 1:nrow(adj_matrix_reordered), labels = rownames(adj_matrix_reordered), las = 2, cex.axis =
axis(2, at = 1:ncol(adj_matrix_reordered), labels = colnames(adj_matrix_reordered), las = 2, cex.axis =

# Detect communities
communities <- cluster_walktrap(dvd.network)

# Get the membership vector
membership <- membership(communities)

# Calculate modularity to quantify the strength of the detected communities
modularity_value <- modularity(communities)
print(paste("Modularity:", modularity_value))

# Identify the number of communities
num_communities <- length(unique(membership))
print(paste("Number of Communities:", num_communities))

# List the members of each community
community_list <- split(V(dvd.network)$name, membership)
community_list

# Reorder the nodes based on community membership
ordered_indices <- order(membership)
ordered_adjacency_matrix <- as.matrix(get.adjacency(dvd.network))[ordered_indices, ordered_indices]

# Plot the reordered adjacency matrix
image(ordered_adjacency_matrix, main="Reordered Adjacency Matrix Based on Community for DVD Network", xl
axis(1, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(dvd.network)$name[ordered_indices], l
axis(2, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(dvd.network)$name[ordered_indices], l

# change the network to undirected
undirected_dvd_network <- as.undirected(dvd.network, mode="collapse")

# Detect communities using the fastgreedy algorithm
communities <- cluster_fast_greedy(undirected_dvd_network)

# Get the membership vector
membership <- membership(communities)

# Calculate modularity to quantify the strength of the detected communities
modularity_value <- modularity(communities)
print(paste("Modularity:", modularity_value))

# Identify the number of communities
```

```r
num_communities <- length(unique(membership))
print(paste("Number of Communities:", num_communities))

# List the members of each community
community_list <- split(V(undirected_dvd_network)$name, membership)
community_list

# Reorder the nodes based on community membership
ordered_indices <- order(membership)
ordered_adjacency_matrix <- as.matrix(get.adjacency(undirected_dvd_network))[ordered_indices, ordered_i

# Plot the reordered adjacency matrix
image(ordered_adjacency_matrix, main="Reordered Adjacency Matrix Based on Community", xlab="Nodes", ylab
axis(1, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(undirected_dvd_network)$name[ordered
axis(2, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(undirected_dvd_network)$name[ordered

edges_list_F <- as_edgelist(a, names = TRUE)


igraph_network_F <- graph.edgelist(edges_list_F, directed = TRUE)

network.adjacency_F <- as_adj(igraph_network_F)

network.adjacency_F

image(Matrix(network.adjacency_F))

com <- cluster_walktrap(a)
plot(com, a, main="Walktrap Algorithm", vertex.size=10, edge.arrow.size=0.05, vertex.label.cex=0.7, ver

# Get the membership vector
mem <- membership(com)

# List the members of each community
community_list <- split(V(a)$name, mem)
community_list

# Reorder the nodes based on community membership
ordered_indices <- order(mem)
ordered_adjacency_matrix <- as.matrix(get.adjacency(a))[ordered_indices, ordered_indices]

# Plot the reordered adjacency matrix
image(ordered_adjacency_matrix, main="Reordered Adjacency Matrix", xlab="Nodes", ylab="Nodes", axes=FAL
axis(1, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(a)$name[ordered_indices], las=2, cex
axis(2, at=seq(0, 1, length.out=length(ordered_indices)), labels=V(a)$name[ordered_indices], las=2, cex
```