Amazon Product Co-Purchasing Network Analysis

Mu Niu, Kai Wa Ho, Jingtang

2024-06-07

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

We processed the raw data by cleaning and integrating nodes and edges into an igraph object. Using dplyr for data manipulation and igraph for network construction, we ensured only valid and complete data were analyzed. For our analysis, we sampled the network to create four subgraphs (Music, Book, Video, and DVD) by expanding from a randomly selected node to capture densely connected nodes, facilitating a better understanding of the relationships between products.

Analysis

Description and Visualization of Subgraphs

We visualized subgraphs for the Music, Book, Video, and DVD categories using tools like the Plotly package in R. These visualizations revealed dense connections and distinct clusters, reflecting co-purchasing patterns within each category. For instance, the Music subgraph showed overlapping demographics across various

genres, while the Book subgraph highlighted strong links between non-fiction and history genres. Analyzing these visualizations provided insights into customer preferences and potential opportunities for cross-selling.

In the DVD subgraph, we observed that customers who purchased comedy DVDs often bought diverse music genres, indicating broad tastes and suggesting targeted marketing strategies. These visualizations deepened our understanding of product interconnections and informed strategies to enhance product recommendations and marketing efforts.

Metrics and Community Detection Results

To explore the structural properties and influential nodes within each network, we examined metrics like density, path length, degree distribution, and centrality. The Music network had a density of 0.237, indicating moderate connectivity with frequent co-purchases and high centrality for influential nodes. The Book network, with a density of 0.137, showed fewer direct connections but highlighted key nodes maintaining its structure.

Community detection algorithms such as Walktrap, Infomap, and FastGreedy provided nuanced views of network structures. Walktrap and Infomap revealed tightly-knit communities, while FastGreedy captured broader structures. The Music network's moderately high modularity scores indicated well-defined community structures, with clusters around influential hubs. The Book and DVD networks exhibited a complex interplay of smaller and larger communities, reflecting diverse co-purchasing behaviors.

Propagation and Community Influence in Co-Purchase Networks:

The propagation of recommendations in viral marketing, as detailed in the paper, aligns closely with the co-purchase behaviors observed in our analysis of the Amazon product networks. In the Music network, nodes like "Movimiento Music" act as hubs, driving further purchases and forming dense co-purchase chains. These clusters, detected through community detection algorithms, mirror the tightly-knit groups described in viral marketing, where frequent co-purchases create robust communities. Understanding these propagation dynamics helps in identifying how recommendations can spread effectively through co-purchase networks, enhancing marketing strategies and product recommendations.

Interpretation of Adjacency Matrix Reorderings

An adjacency matrix is an essential tool for visualizing and analyzing relationships in social networks. Each cell in the matrix represents a co-purchase relationship between products, with grey cells indicating connections. The original matrix for the combined network showed dense intra-network connections and minimal inter-network interactions, highlighting the stability and strong internal community structures within each category.

Reordering adjacency matrices based on hierarchical and community structures clarified the network organization. For the combined network, the reordered matrix is nearly identical to the original one, albeit rotated 90 degrees counterclockwise. This alignment is logical, as each individual network should be perceived as a community, aligning with the principles of the walktrap algorithm. Walktrap alogrithm simulates random walks to detect communities, and these walks tend to stay within densely connected subgroups of nodes.

In the Music network's reordered matrices, we identified five distinct clusters centered around influential nodes. For instance, Cluster 1(influential node: 240889 - "De La Soul Is Dead") shows a dense network with nodes having robust connections across various music genres, mirroring the idea that frequently co-purchased items form tight-knit clusters. Notably, community detection algorithms seem to aggregate broader communities, which may merge smaller, distinct clusters into larger ones. This can reduce the apparent number of clusters which we see in the original matrix.

In the Book network's reordered matrices, we observed three distinct communities, centered around the influential nodes 1041, 245481, and 33389. These findings align perfectly with the results from our community

detection analysis. The reordered matrices enhance the visibility of these communities, making the patterns and internal structures clearer compared to the original matrix. This phenomenon is consistent across the reordered matrices for the other networks as well, where the community structures become more pronounced and easily identifiable, highlighting the effectiveness of the reordering process in revealing the network's inherent organization.

Results

Key Findings from the Analysis

Our analysis highlighted unique structural characteristics and influential nodes in the Music, Book, Video, and DVD networks. The Music network, with its high density and short path length, was the most interconnected, indicating robust co-purchase relationships. The Book network, though more dispersed, relied on key nodes to maintain connectivity. The Video and DVD networks exhibited moderate density and compactness, with influential nodes playing significant roles in their community structures.

Community detection revealed distinct clusters reflecting underlying co-purchasing behaviors. Walktrap and Infomap identified tightly-knit communities within each network, while FastGreedy showed broader, less distinct groupings. The consistent appearance of specific clusters across algorithms indicates the stability and robustness of these networks' structures, providing insights into product linkages through customer purchases.

Practical Applications for Marketing Strategies: Insights from the viral marketing study are particularly relevant for devising marketing strategies in the context of our co-purchase network analysis. The dense clusters and influential nodes identified in networks like Music and Book suggest potential for targeted marketing and product bundling. For example, leveraging the strong intra-community ties and rapid co-purchase propagation can inform effective promotional campaigns. The temporal dynamics observed, such as the short average path length in the Music network, highlight opportunities for timely and impactful product recommendations, reflecting the rapid influence spread seen in viral marketing.

Visual and Textual Summaries

Visualizations like adjacency matrices and network graphs effectively illustrated the structure and dynamics of the co-purchase networks. Both the original and reordered matrices highlighted dense intra-category connections and community definitions identified by detection algorithms. For instance, the reordered Music network matrix clearly showed clusters around influential nodes such as "Movimiento Music," emphasizing cohesive groups of frequently co-purchased items.

In our analysis of the Amazon product co-purchasing network, minimal differences between the original and reordered adjacency matrices were noted across all categories (Books, Video, Music, DVD) and the combined network. This can be attributed to two main factors. First, the networks' inherent community structures were well-formed due to dense intra-category connections, making clusters visible even in the original matrices. These strong internal connections within categories contrasted with sparser inter-category links, which maintained the clarity of clusters. Second, community detection algorithms like Walktrap and Fastgreedy reinforced these existing dense clusters. As a result, the reordered matrices closely resembled the original ones, underscoring the networks' stability and robustness.

These visual insights were further supported by textual explanations that linked clusters to customer behaviors, such as the diverse musical preferences connected to comedy DVD buyers or demographic overlaps in book and music purchases. The textual analysis provided practical implications, indicating how understanding network structures can enhance marketing strategies and improve product recommendations. Together, these visual and textual elements offered a comprehensive view of the network dynamics, highlighting their potential to influence customer behavior and boost sales.

Conclusion

We analyzed the Amazon 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 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.

Further Elaboration:

Since we do not have enough space in this report, further analysis and interpretation can be accessed via this **link**

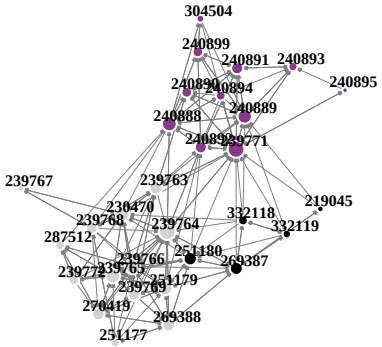
Acknowledgments:

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.

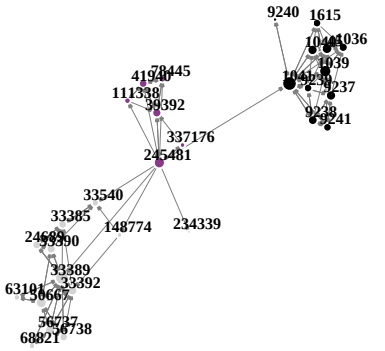
Appendices

- Figures and Printed Result

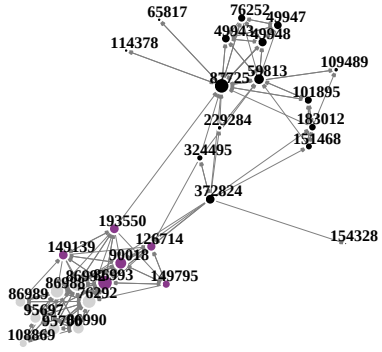
Music Products Community



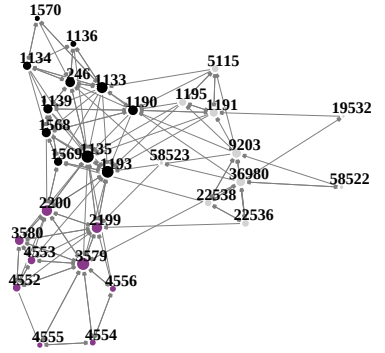
Book Products Community



Video Products Community



DVD Products Community



Music

```
## Network Density: 0.2367816
```

```
## Average Path Length: 1.889655
```

```
## Network Diameter: 3
```

```
## Node Degree:
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    3.00    9.00   14.00   13.73   18.50   27.00
```

```
## Node Betweenness Centrality:
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   2.440   7.782  33.767  32.210 185.015
```

```
## Node Closeness Centrality:
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01205 0.01476 0.01639 0.01626 0.01810 0.02041
```

```
## Node Eigenvector Centrality:
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.09343 0.28684 0.44291 0.46907 0.65328 1.00000
```

```
## Edge Betweenness:
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.970   6.071   9.141  10.460  71.000
```

```
## Walktrap Algorithm:
```

```
## 219045 230470 239763 239764 239765 239766 239767 239768 239769 239771 239772
##      1      2      1      1      2      2      2      2      2      3      2
## 240888 240889 240890 240891 240892 240893 240894 240895 240899 251177 251179
##      3      3      3      3      3      3      3      3      3      2      2
## 251180 269387 269388 270419 287512 304504 332118 332119
##      1      1      2      2      2      3      1      1
```

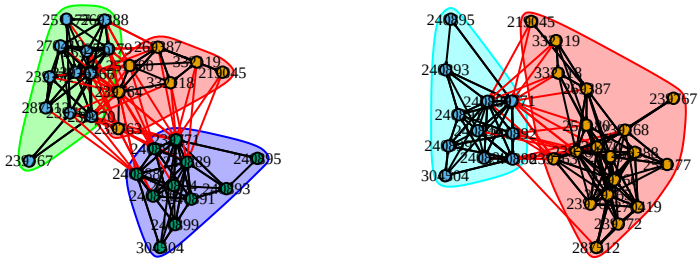
```
## Modularity: 0.3337261
```

```
## Infomap Algorithm:
```

```
## 219045 230470 239763 239764 239765 239766 239767 239768 239769 239771 239772
##      1      1      1      1      1      1      1      1      1      2      1
## 240888 240889 240890 240891 240892 240893 240894 240895 240899 251177 251179
##      2      2      2      2      2      2      2      2      2      1      1
## 251180 269387 269388 270419 287512 304504 332118 332119
##      1      1      1      1      1      2      1      1
```

```
## Modularity: 0.3301913
```

Walktrap Algorithm **Infomap Algorithm**



Book

```
## Network Density: 0.1367816

## Average Path Length: 2.604598

## Network Diameter: 4

## Node Degree:

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   5.000   7.000   7.933  10.000  20.000

## Node Betweenness Centrality:

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   2.000  16.300   8.354 126.500

## Node Closeness Centrality:

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## 0.01000 0.03846 0.05882 0.05155 0.06667 0.10000       1

## Node Eigenvector Centrality:
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.007996 0.013327 0.024175 0.234801 0.511089 1.000000
```

```
## Edge Betweenness:
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.875   3.333   7.513   9.000 104.500
```

```
## Walktrap Algorithm:
```

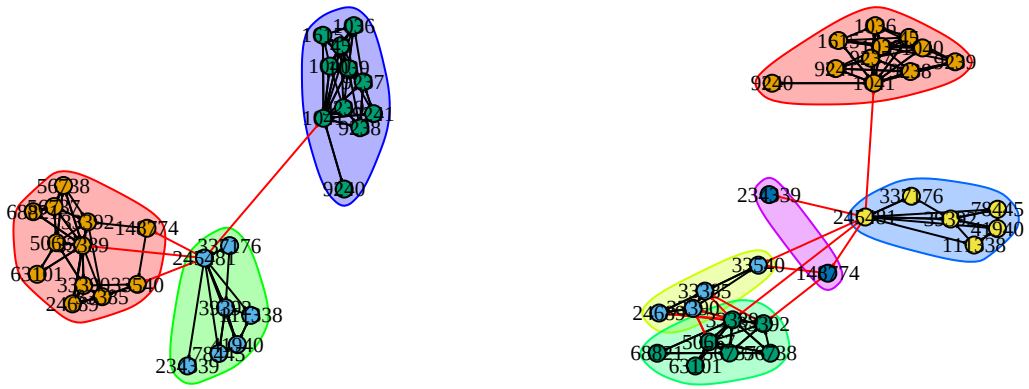
```
##     45   1036   1039   1040   1041   1615   9237   9238   9239   9240   9241
##      3      3      3      3      3      3      3      3      3      3      3
##  24689  33385  33389  33390  33392  33540  39392  41940  50667  56737  56738
##      1      1      1      1      1      1      2      2      1      1      1
##  63101  68821  78445 111338 148774 234339 245481 337176
##      1      1      2      2      1      2      2      2
```

```
## Modularity: 0.5908128
```

```
## Infomap Algorithm:
```

```
##     45   1036   1039   1040   1041   1615   9237   9238   9239   9240   9241
##      1      1      1      1      1      1      1      1      1      1      1
##  24689  33385  33389  33390  33392  33540  39392  41940  50667  56737  56738
##      2      2      3      2      3      2      4      4      3      3      3
##  63101  68821  78445 111338 148774 234339 245481 337176
##      3      3      4      4      5      5      4      4
```

```
## Modularity: 0.5614716
```



Video

```
## Network Density: 0.1885057

## Average Path Length: 2.271264

## Network Diameter: 4

## Node Degree:

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    7.25   10.00   10.93   14.75   23.00

## Node Betweenness Centrality:

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##   0.0000   0.2708   2.0762  23.8000  12.5214 242.4333

## Node Closeness Centrality:

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## 0.01389 0.01493 0.02083 0.03198 0.05000 0.07692       1

## Node Eigenvector Centrality:
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.008135 0.016008 0.134973 0.347216 0.704178 1.000000
```

```
## Edge Betweenness:
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.738   2.880   7.720   8.137 168.000
```

```
## Walktrap Algorithm:
```

```
##  49943  49947  49948  59813  65817  76252  76292  86988  86989  86990  86992
##      1      1      1      1      1      1      2      2      2      2      2
##  86993  87725  90018  95697  95700 101895 108869 109489 114378 126714 149139
##      2      1      2      2      2      1      2      1      1      2      2
## 149795 151468 154328 183012 193550 229284 324495 372824
##      2      1      1      1      2      1      1      1
```

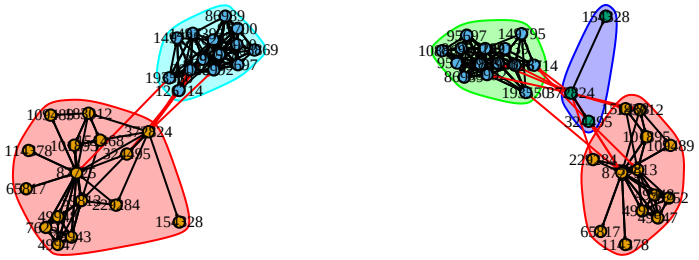
```
## Modularity: 0.4274242
```

```
## Infomap Algorithm:
```

```
##  49943  49947  49948  59813  65817  76252  76292  86988  86989  86990  86992
##      1      1      1      1      1      1      2      2      2      2      2
##  86993  87725  90018  95697  95700 101895 108869 109489 114378 126714 149139
##      2      1      2      2      2      1      2      1      1      2      2
## 149795 151468 154328 183012 193550 229284 324495 372824
##      2      1      3      1      2      1      3      3
```

```
## Modularity: 0.425119
```

Walktrap Algorithm **Infomap Algorithm**



DVD

```
## Network Density: 0.1712644

## Average Path Length: 2.050575

## Network Diameter: 4

## Node Degree:

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   6.250  10.000   9.933  12.750  19.000

## Node Betweenness Centrality:

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   1.882  16.701  35.467  67.937 152.102

## Node Closeness Centrality:

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01299 0.01551 0.01755 0.01729 0.01852 0.02083

## Node Eigenvector Centrality:
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0470  0.2187  0.4037  0.4569  0.6383  1.0000
```

```
## Edge Betweenness:
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   4.133   8.000  11.819  16.289  69.222
```

```
## Walktrap Algorithm:
```

```
##   246  1133  1134  1135  1136  1139  1190  1191  1193  1195  1568  1569  1570
##     4     4     4     1     4     1     1     1     1     1     1     1     4
##  2199  2200  3579  3580  4552  4553  4554  4555  4556  5115  9203 19532 22536
##     2     2     2     2     2     2     2     2     2     1     3     3     3
## 22538 36980 58522 58523
##     3     3     3     3
```

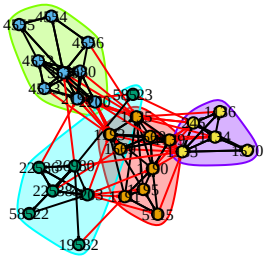
```
## Modularity: 0.419328
```

```
## Infomap Algorithm:
```

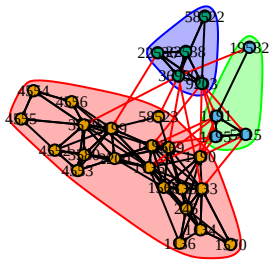
```
##   246  1133  1134  1135  1136  1139  1190  1191  1193  1195  1568  1569  1570
##     1     1     1     1     1     1     1     2     1     2     1     1     1
##  2199  2200  3579  3580  4552  4553  4554  4555  4556  5115  9203 19532 22536
##     1     1     1     1     1     1     1     1     1     2     3     2     3
## 22538 36980 58522 58523
##     3     3     3     1
```

```
## Modularity: 0.2380073
```

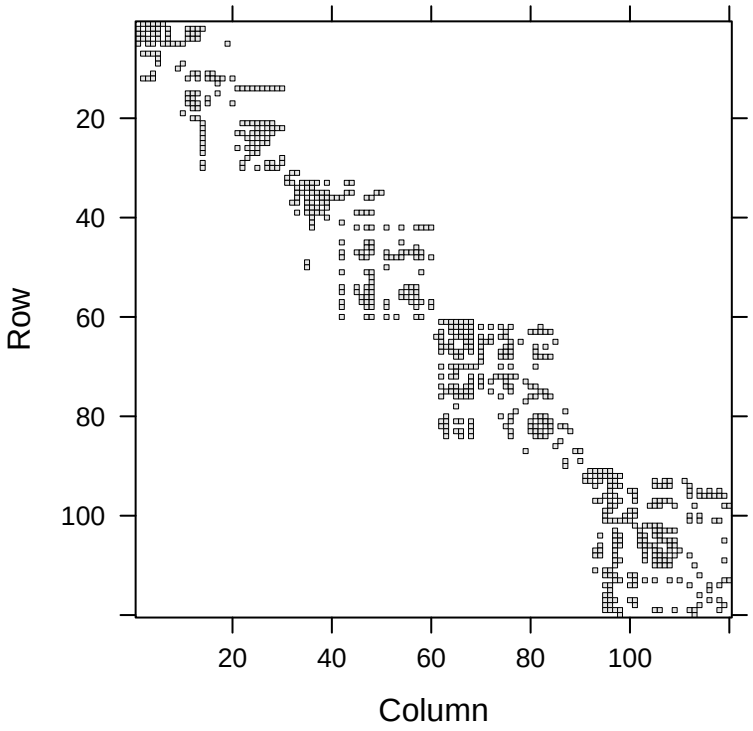
Walktrap Algorithm



Infomap Algorithm

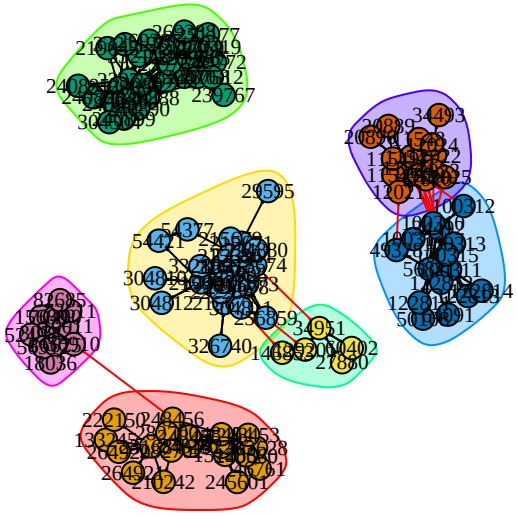


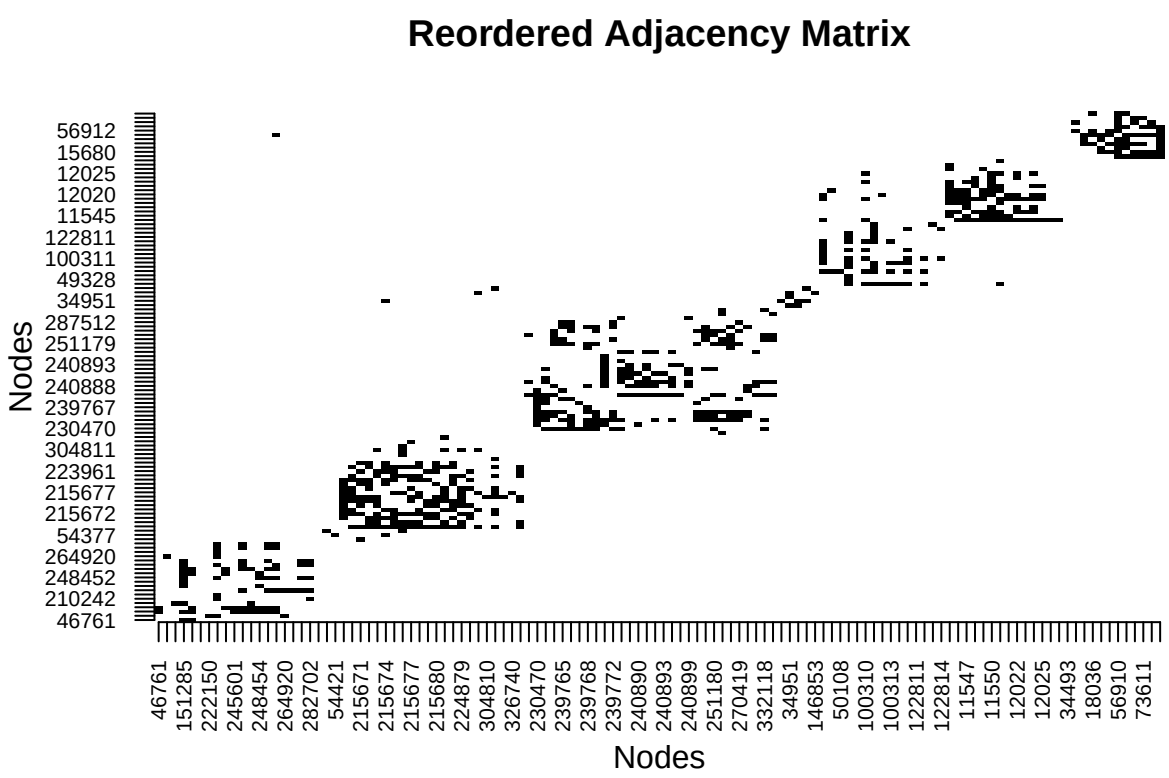
Combined



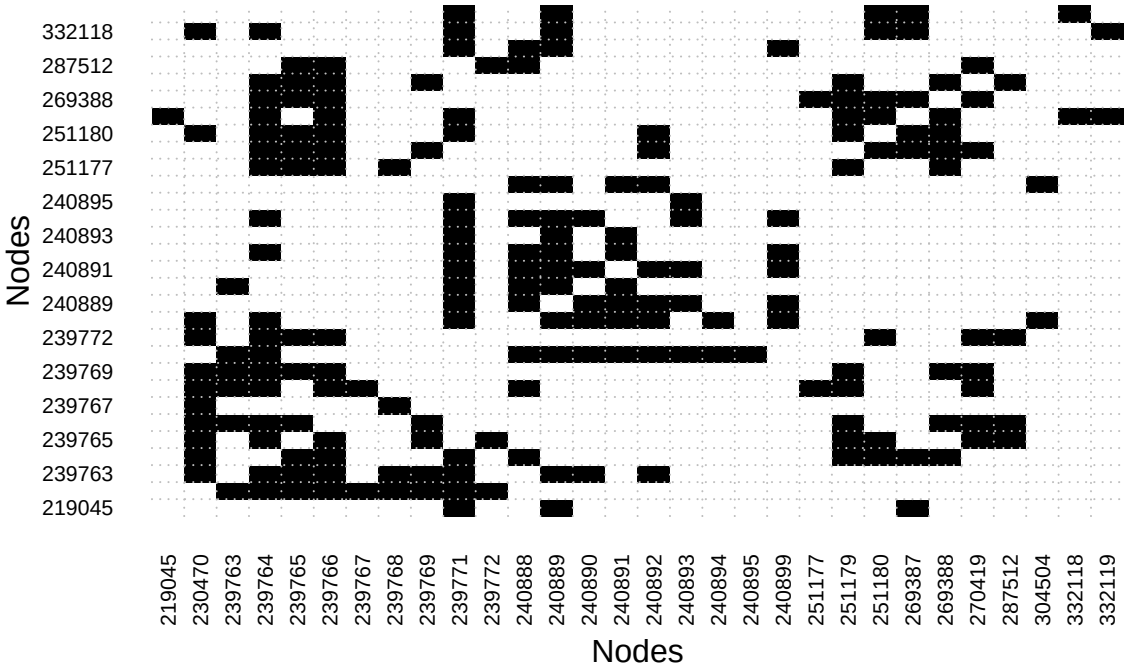
Dimensions: 120 x 120

Walktrap Algorithm

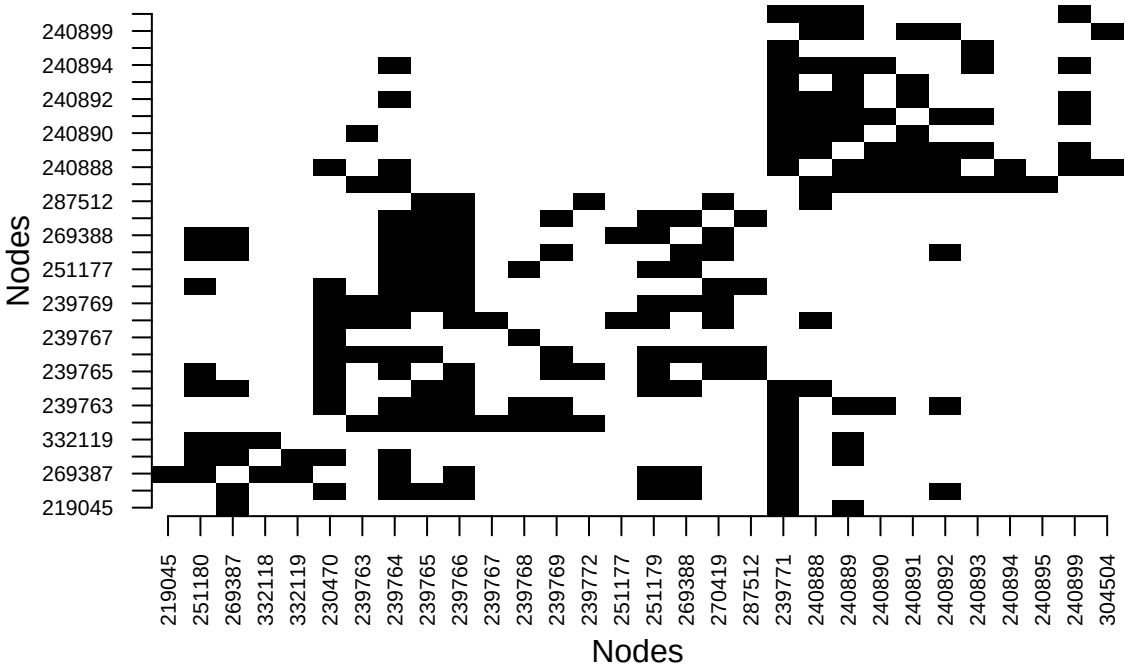


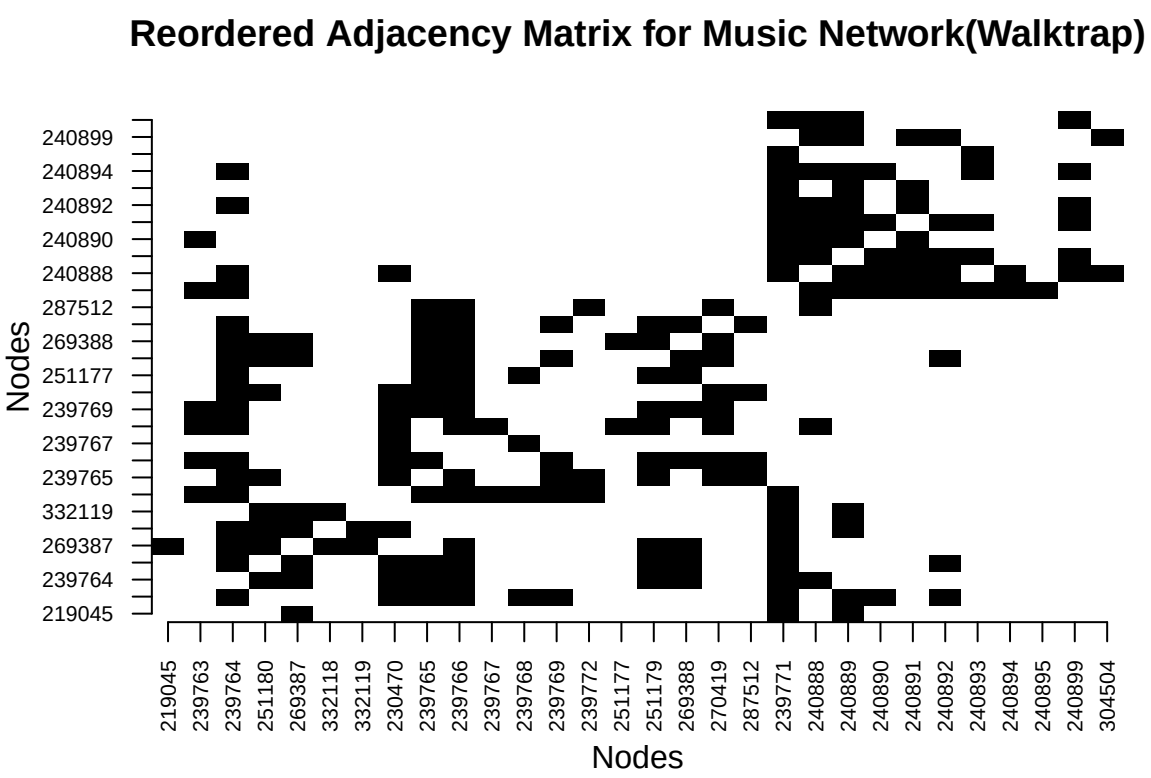

Reordered Adjacency Matrix
Nodes
56912
15680
12025
12020
11545
122811
100311
49328
34951
287512
251179
240893
240888
239767
230470
304811
223961
215677
215672
54377
264920
248452
210242
46761
46761
151285
222150
245601
248454
264920
282702
54421
215671
215674
215677
215680
224879
304810
326740
230470
239765
239768
239772
240890
240893
240899
251180
270419
332118
34951
146853
50108
100310
100313
122811
122814
11547
11550
12022
12025
34493
18036
56910
73611
Nodes

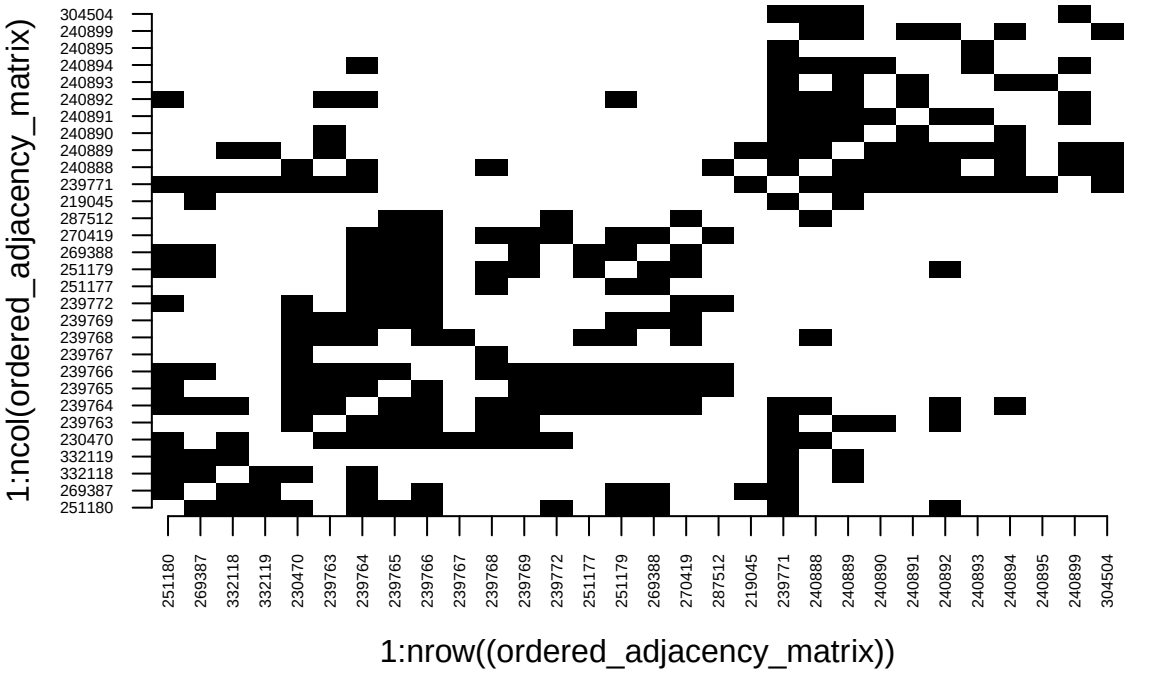
Music


Adjacency Matrix For Music
Nodes
332118
287512
269388
251180
251177
240895
240893
240891
240889
239772
239769
239767
239765
239763
219045
219045
230470
239763
239764
239765
239766
239767
239768
239769
239771
239772
240888
240889
240890
240891
240892
240893
240894
240895
240899
251177
251179
251180
269387
269388
270419
287512
304504
332118
332119
Nodes

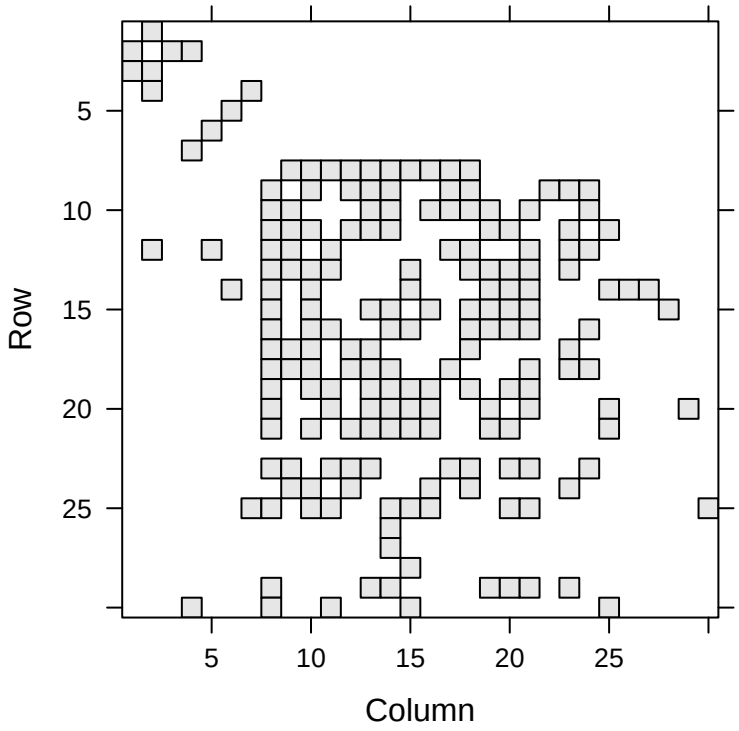
```
## [1] 0.3675181
```


Reordered Adjacency Matrix for Music Network(Optimal)
Nodes
240899
240894
240892
240890
240888
287512
269388
251177
239769
239767
239765
239763
332119
269387
219045
Nodes
219045
251180
269387
332118
332119
230470
239763
239764
239765
239766
239767
239768
239769
239772
251177
251179
269388
270419
287512
239771
240888
240889
240890
240891
240892
240893
240894
240895
240899
304504


Reordered Adjacency Matrix for Music Network(Walktrap)
Nodes
240899
240894
240892
240890
240888
287512
269388
251177
239769
239767
239765
332119
269387
239764
219045
219045
239763
239764
251180
269387
332118
332119
230470
239765
239766
239767
239768
239769
239772
251177
251179
269388
270419
287512
239771
240888
240889
240890
240891
240892
240893
240894
240895
240899
304504
Nodes


Reordered Adjacency Matrix for Music Network(Fastgreedy)
1:ncol(ordered_adjacency_matrix)
304504
240899
240895
240894
240893
240892
240891
240890
240889
240888
239771
219045
287512
270419
269388
251179
251177
239772
239769
239768
239767
239766
239765
239764
239763
230470
332119
332118
269387
251180
251180
269387
332118
332119
230470
239763
239764
239765
239766
239767
239768
239769
239772
251177
251179
269388
270419
287512
219045
239771
240888
240889
240890
240891
240892
240893
240894
240895
240899
304504
1:nrow((ordered_adjacency_matrix))

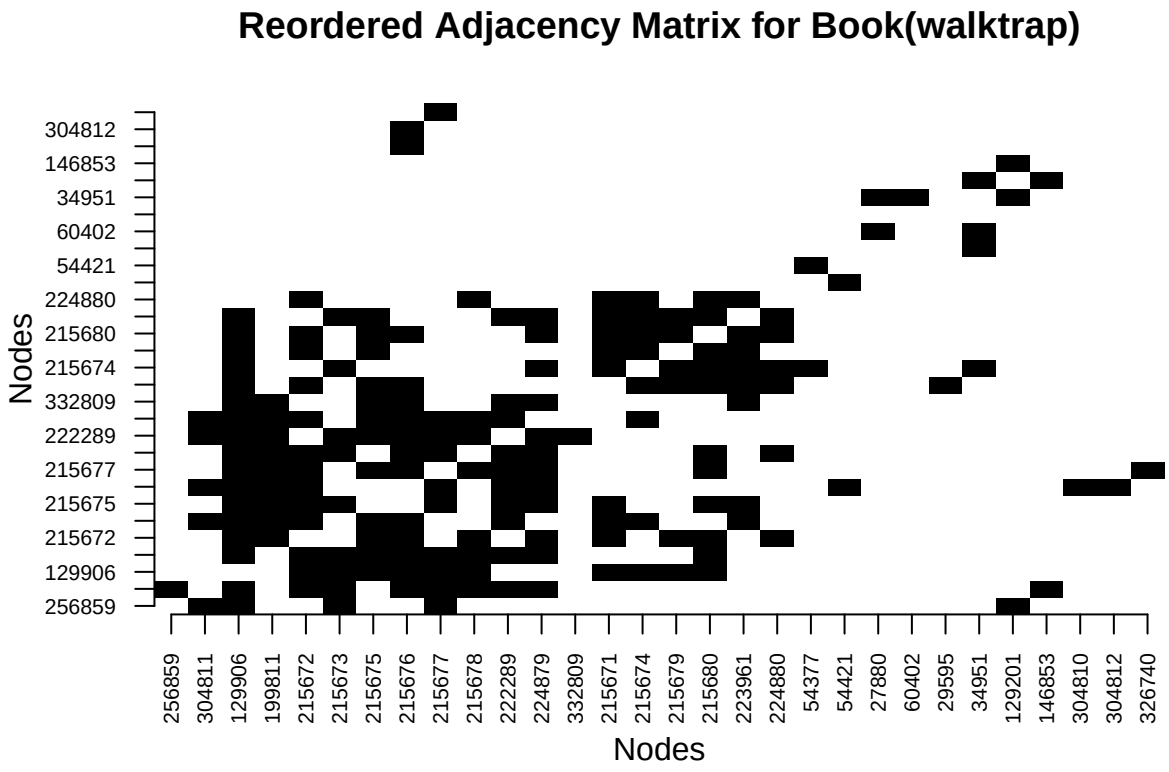
Book



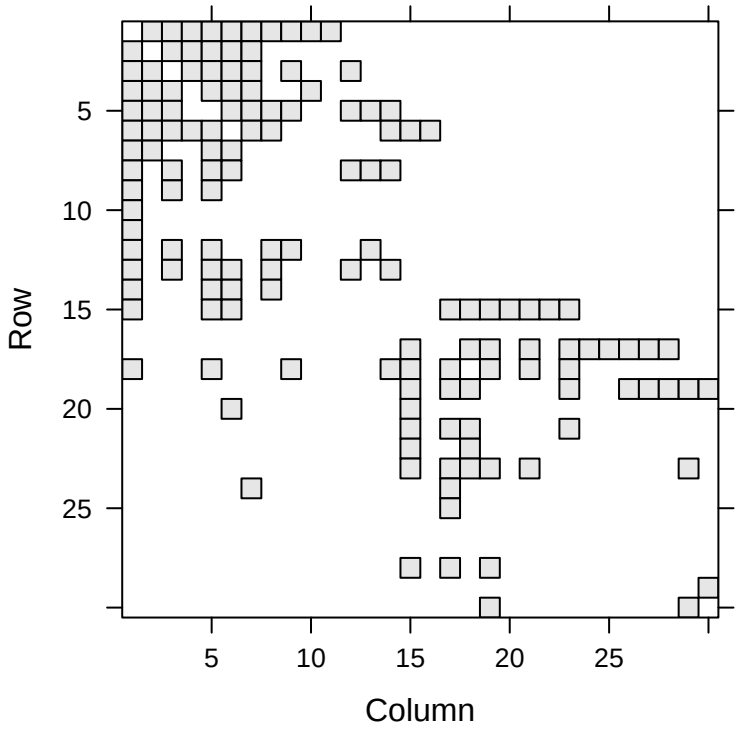
Dimensions: 30 x 30

```
## [1] "Modularity: 0.172013661431651"

## [1] "Number of Communities: 12"

## $`1`
## [1] "256859" "304811"
##
## $`2`
##  [1] "129906" "199811" "215672" "215673" "215675" "215676" "215677" "215678"
##  [9] "222289" "224879" "332809"
##
## $`3`
## [1] "215671" "215674" "215679" "215680" "223961" "224880"
##
## $`4`
## [1] "54377" "54421"
##
## $`5`
## [1] "27880" "60402"
##
## $`6`
## [1] "29595"
##
## $`7`
## [1] "34951"
```

```
##
## $`8`
## [1] "129201"
##
## $`9`
## [1] "146853"
##
## $`10`
## [1] "304810"
##
## $`11`
## [1] "304812"
##
## $`12`
## [1] "326740"
```



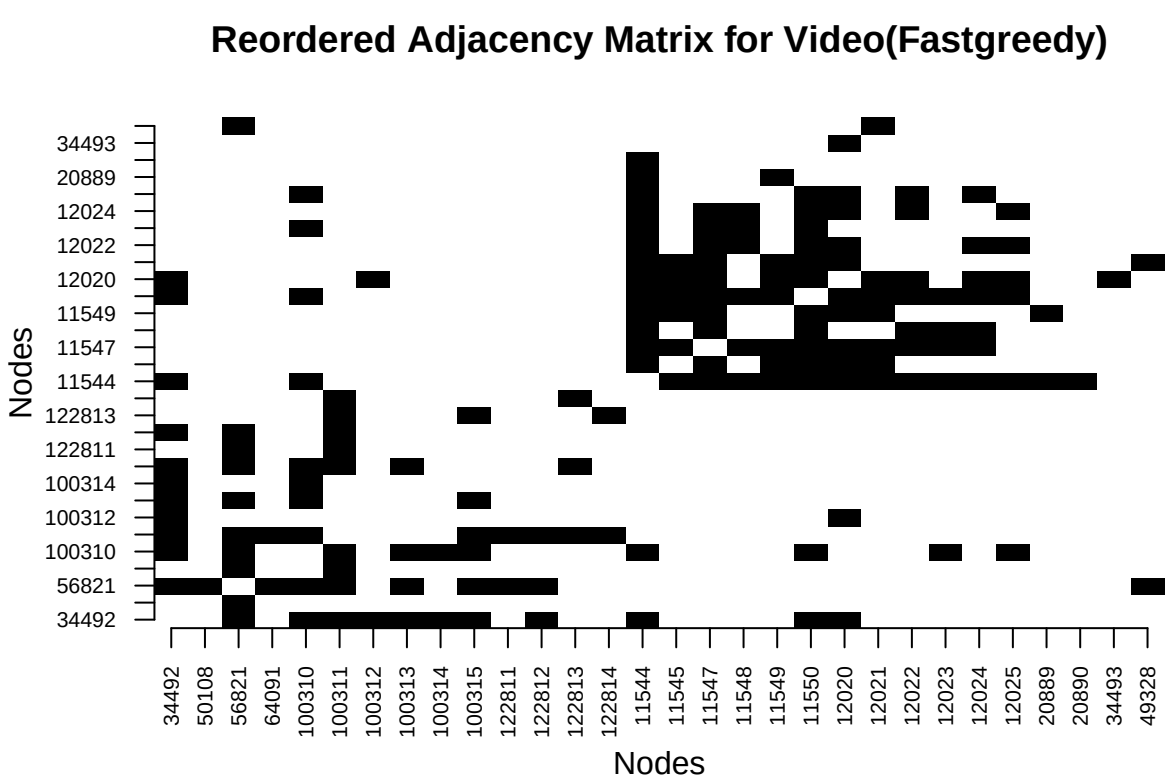
Video



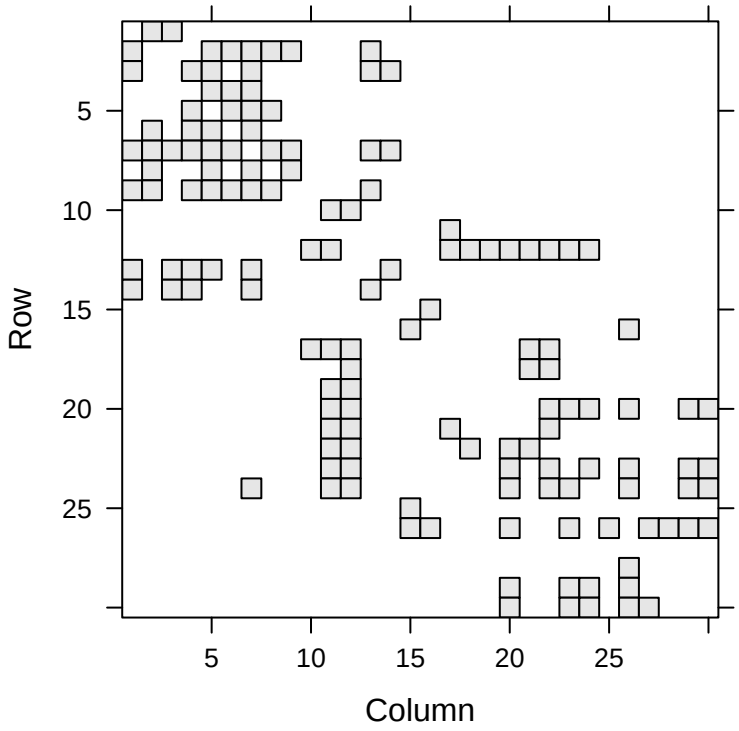
Dimensions: 30 x 30

```
## [1] "Modularity: 0.370128154313648"
```

```
## [1] "Number of Communities: 2"
```


Reordered Adjacency Matrix for Video(Fastgreedy)
Nodes
34493
20889
12024
12022
12020
11549
11547
11544
122813
122811
100314
100312
100310
56821
34492
34492
50108
56821
64091
100310
100311
100312
100313
100314
100315
122811
122812
122813
122814
11544
11545
11547
11548
11549
11550
12020
12021
12022
12023
12024
12025
20889
20890
34493
49328
Nodes

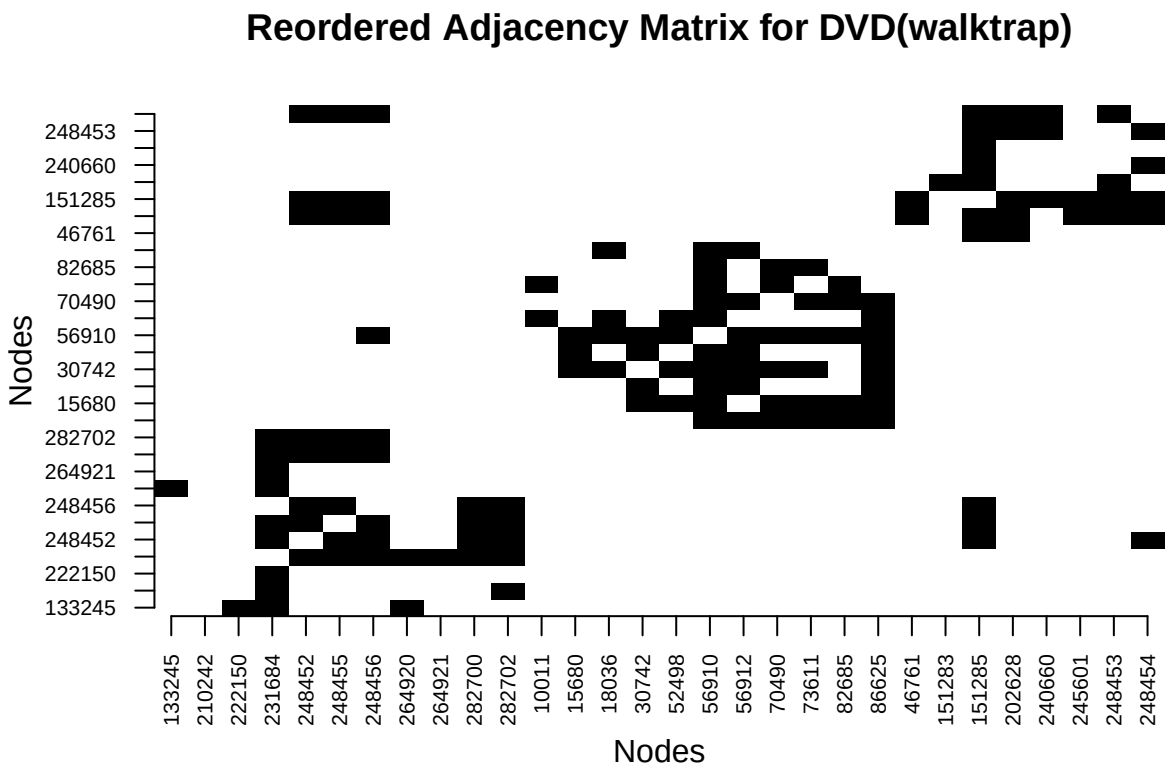
DVD



Dimensions: 30 x 30

```
## [1] "Modularity: 0.549635112054241"
```

```
## [1] "Number of Communities: 3"
```



- 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")
```

```r
# 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))

# 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))
```

```r
# 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
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
```

```r
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
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")
```

```r
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
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.
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)
```

```r
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=
plot(infomap_communities_dvd, dvd.network, main="Infomap Algorithm", vertex.size=10, edge.arrow.size=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
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
```

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

# 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]
```

```r
# Plot the reordered adjacency matrix
image(ordered_adjacency_matrix, main="Reordered Adjacency Matrix Based on Community for Book Network", x
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)
ordered_adjacency_matrix <- as.matrix(get.adjacency(undirected_book_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_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
```

```r
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"))

# 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
```

```r
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
```

```r
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
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))
```

```r
# 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
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_in

# 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))
```

```r
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
```