assignment5

October 21, 2024

1 Assignment 3

- Tanja Gurtner - 17-677-295
- Cyrill Georg Meier - 17-552-316

[49]:

```
import networkx as nx
import matplotlib.pyplot as plt
import os
import pandas as pd
import numpy as np
from PIL import Image
import scipy.stats as stats
import random
import math
```

1.1 Preparation

Download data from Git repository

[3]:

```
import os
import requests

# GitHub repository and folder details
repo_owner = "UZH-Cyrill-Meier"
repo_name = "NetworkScience"
folder_path = "Assignments/assignment 5/datasets"

# GitHub API URL to get contents of the folder
api_url = f"https://api.github.com/repos/{repo_owner}/{repo_name}/contents/
 ↪{folder_path}"

# Base URL for raw file download
raw_base_url = f"https://raw.githubusercontent.com/{repo_owner}/{repo_name}/
 ↪main/{folder_path}/"

# Local directory where you want to save the downloaded files
local_directory = "data"
```

```
# Create the local directory if it doesn't exist
if not os.path.exists(local_directory):
    os.makedirs(local_directory)

# Function to download a file from the raw GitHub URL
def download_file(file_name):
    file_url = raw_base_url + file_name
    local_path = os.path.join(local_directory, file_name)
    try:
        response = requests.get(file_url)
        response.raise_for_status()  # Check if the request was successful
        with open(local_path, 'wb') as file:
            file.write(response.content)
        print(f"Downloaded: {local_path}")
    except Exception as e:
        print(f"Error downloading {file_name}: {e}")

# Get the list of files in the GitHub folder
response = requests.get(api_url)
if response.status_code == 200:
    files = response.json()
    for file_info in files:
        if file_info['type'] == 'file':  # Check if it's a file (not a␣
 ↪directory)
            file_name = file_info['name']
            download_file(file_name)
else:
    print(f"Failed to retrieve folder contents: {response.status_code}")
```

```
Downloaded: data/.DS_Store
Downloaded: data/graph_Korea.gml
Downloaded: data/graph_dolphins.gml
Downloaded: data/graph_karate.gml
Downloaded: data/graph_madrid.gml
Downloaded: data/graph_starwars.gml
```

Get all existing files

[4]:

```
directory = os.fsencode(local_directory)
titels = ['korea', 'dolphins', 'karate', 'madrid', 'starwars']
files = []

for file in os.listdir(directory):
    filename = os.fsdecode(file)
    if filename.endswith(".gml"):
        files.append(os.path.join(local_directory, filename))
        continue
    else:
        continue
```

Read all the files and add to dict `graphs`

[5]:

```
graphs = []
for file in files:
  graphs.append(nx.read_gml(file))
```

[6]:

```
graphs
```

[6]:

```
[<networkx.classes.graph.Graph at 0x7a4b83c3c310>,
 <networkx.classes.graph.Graph at 0x7a4b83c3f5b0>,
 <networkx.classes.graph.Graph at 0x7a4b83c3c550>,
 <networkx.classes.graph.Graph at 0x7a4b83c3d270>,
 <networkx.classes.graph.Graph at 0x7a4b83c3c1c0>]
```

1.1.1 Datasets provided

- **graph_madrid.gml**: A network of associations among the terrorists involved in the 2004 Madrid train bombing, as reconstructed from press stories after-the-fact (Cardillo et al. [2013]).
- **graph_starwars.gml**: Network of interactions in Star Wars episode 4. Nodes are characters and edges represent a co-appearance in the same scene (Gabasova [2016]).
- **graph_korea.gml**: The network represents women in Korea discussing family planning. Edges represent a planning discussion (Sonquist [1984]).
- **graph_karate.gml**: Nodes represent members of a Karate club, and Edges represent a tie between two members (Zachary [1977]).
- **graph_dolphins.gml**: Dolphin social network: Nodes represent dolphins and Edges represent frequent associations observed among a group of 62 individuals (Lusseau et al. [2003]).

2 Exercise 1

(3 points) For the provided network datasets, find the communities using

(a) the greedy modularity maximization by Clauset Newman and Moore (Clauset et al. [2004]) and

(b) the label propagation algorithm. Assign to each community a color and draw the resulting graph, where each node is colored after the community it belongs to, while community internal links and inter-communities links are clearly recognizable.

Hint: in order to make the visualization meaningful, tune nodes and links colors, e.g., internal links are black and external links are light gray.

Hint: The greedy modularity maximization algorithm (Clauset et al. [2004]) is available as `greedy_modularity_communities()`.
Label propagation algorithm is available as `label_propagation_communities()`.

[10]:

```
# Step 2: Find communities using greedy modularity maximization
greedy_communities = []
label_communities = []
for graph in graphs:
 greedy_communities.append(list(nx.algorithms.community.
 ↪greedy_modularity_communities(graph)))
 label_communities.append(list(nx.algorithms.community.
 ↪label_propagation_communities(graph)))
```

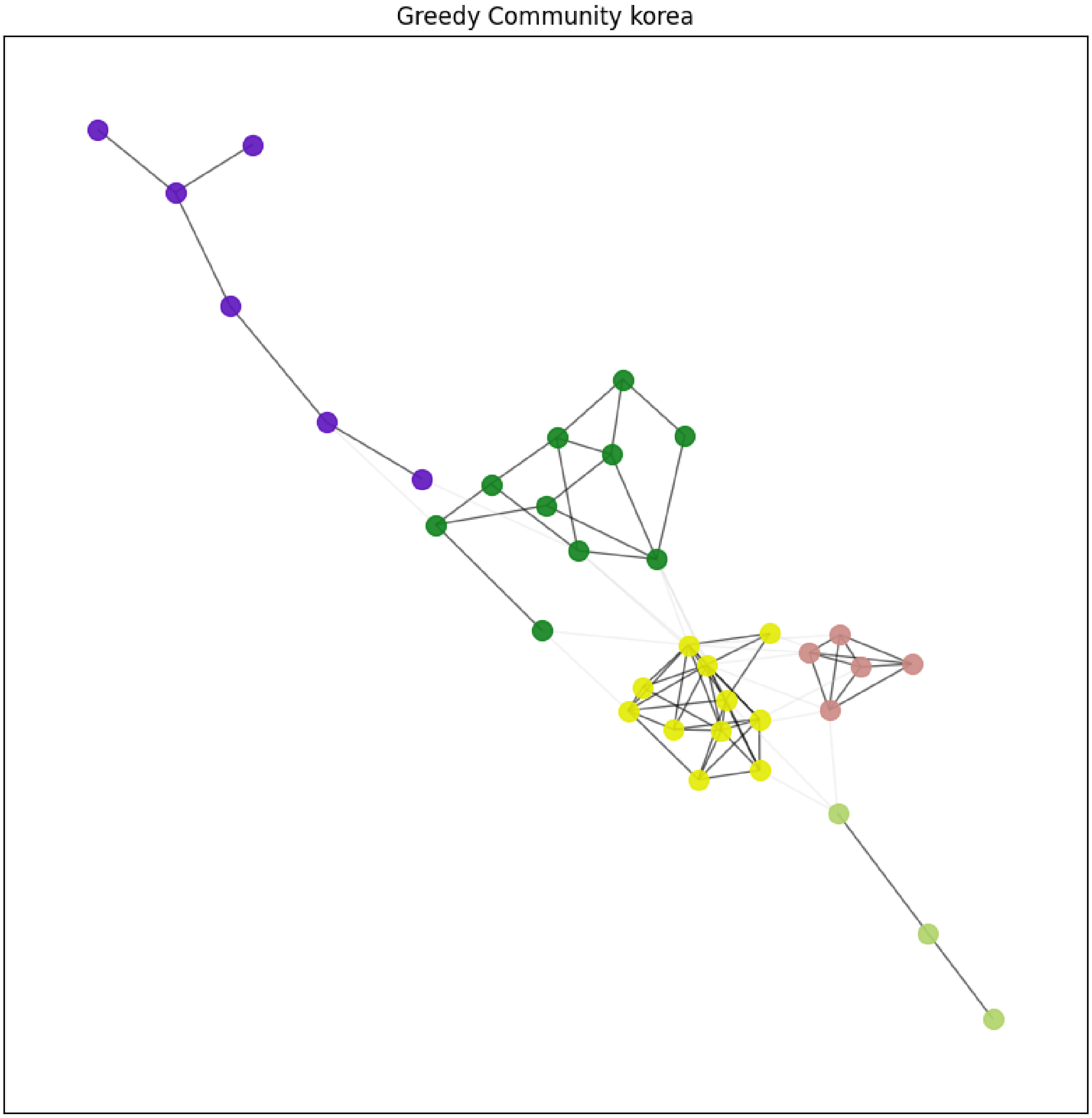
[8]:

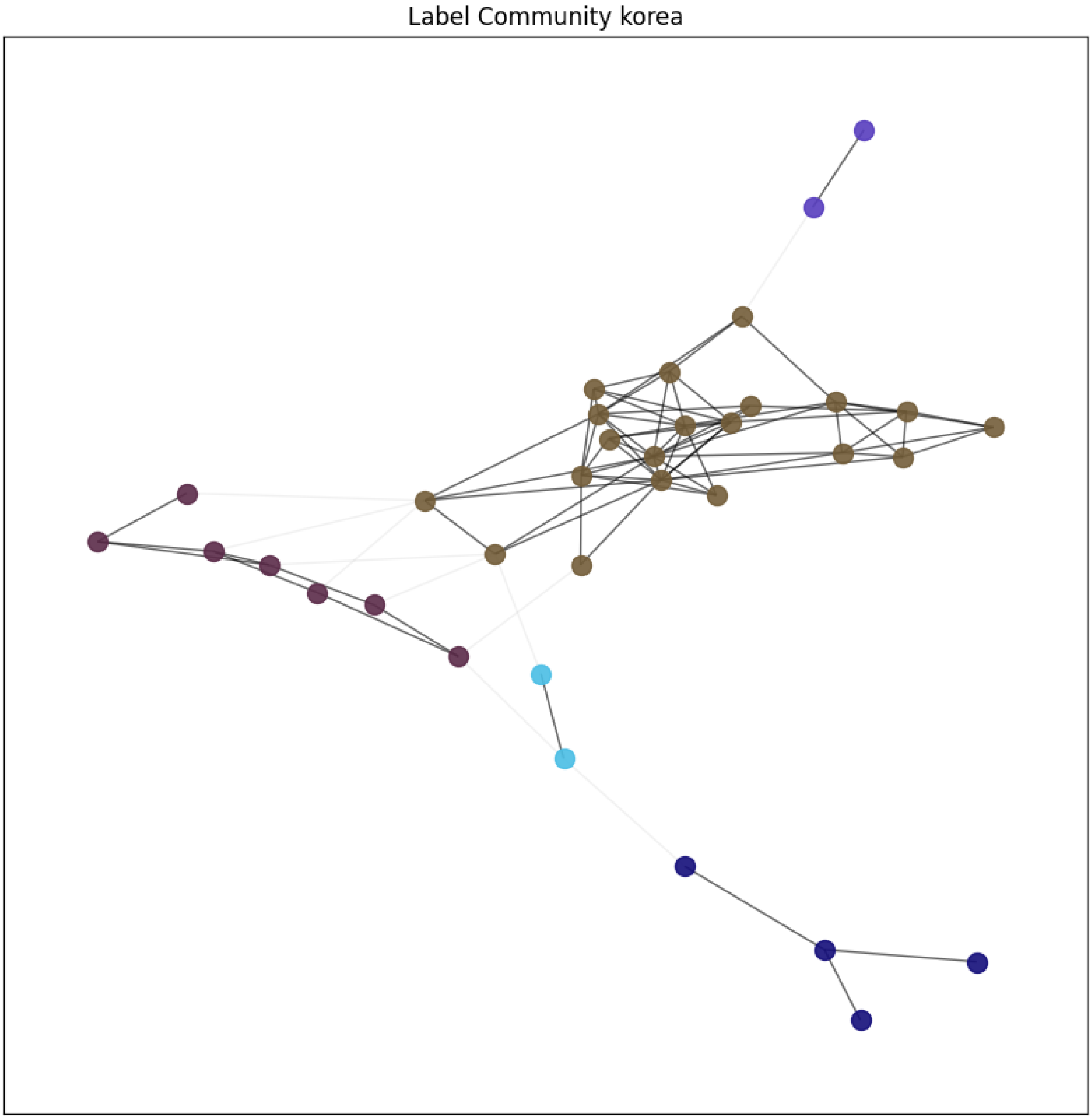
```
# Function to plot communities with distinct node colors and tuned link colors
def plot_colored_communities(graph, communities, title):
    pos = nx.spring_layout(graph)  # Layout for graph nodes
    plt.figure(figsize=(10, 10))

    # Generate a unique color for each community
    colors = [f"#{random.randint(0, 0xFFFFFF):06x}" for _ in␣
 ↪range(len(communities))]

    # Convert communities to a dictionary for quick lookup
    node_community_map = {}
    for i, community in enumerate(communities):
        for node in community:
            node_community_map[node] = i

    # Draw nodes, each community with a different color
    for i, community in enumerate(communities):
        nx.draw_networkx_nodes(graph, pos, nodelist=list(community),␣
 ↪node_color=colors[i], node_size=100, alpha=0.9)

    # Draw edges
    internal_edges = []
    external_edges = []
    for u, v in graph.edges():
        # Check if the two nodes belong to the same community
        same_community = any(u in community and v in community for community in␣
 ↪communities)

        if same_community:
            internal_edges.append((u, v))  # Internal edges (same community)
        else:
            external_edges.append((u, v))  # External edges (different␣
 ↪communities)

    # Draw internal edges (within the same community) in black
    nx.draw_networkx_edges(graph, pos, edgelist=internal_edges,␣
 ↪edge_color="black", alpha=0.6)
```

```
    # Draw external edges (between different communities) in light gray
    nx.draw_networkx_edges(graph, pos, edgelist=external_edges,␣
↪edge_color="lightgray", alpha=0.3)

    plt.title(title)
    plt.show()
```

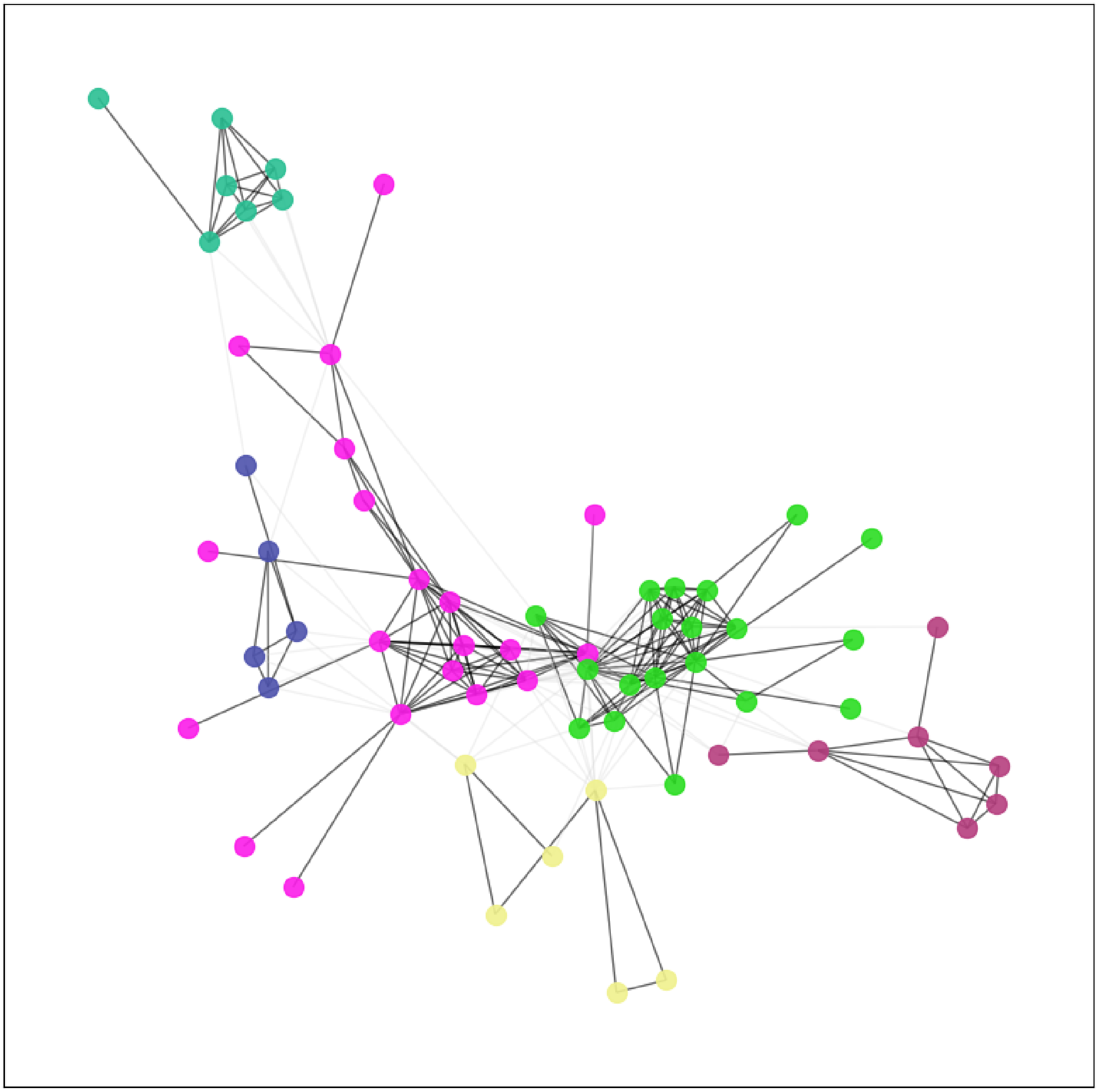
[11]:

```
for graph, greedy_community, label_community, titel in zip(graphs,␣
↪greedy_communities, label_communities, titels):
    plot_colored_communities(graph, greedy_community, "Greedy Community " +titel)
    plot_colored_communities(graph, label_community, "Label Community " +titel)
```

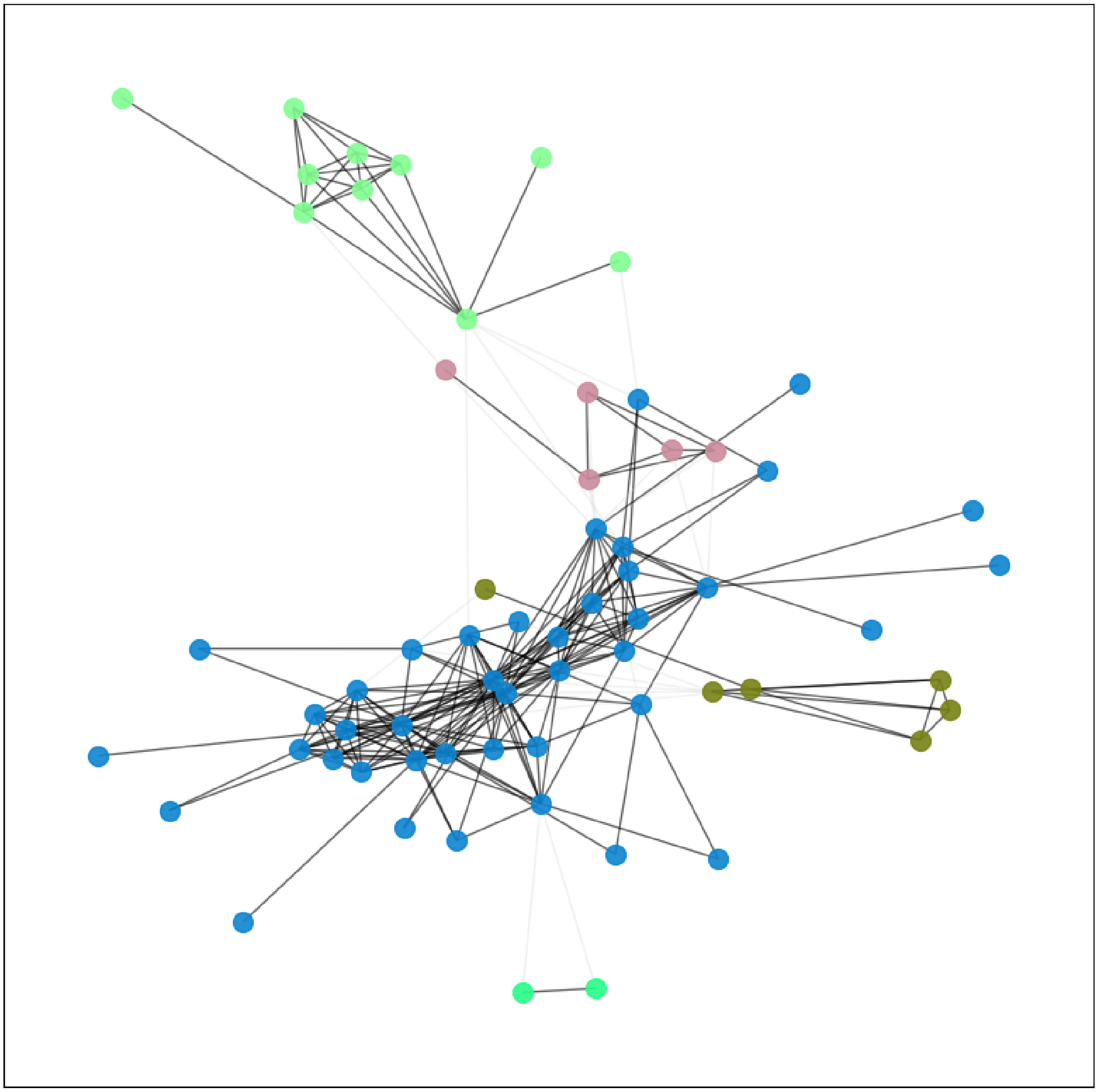



Label Community korea

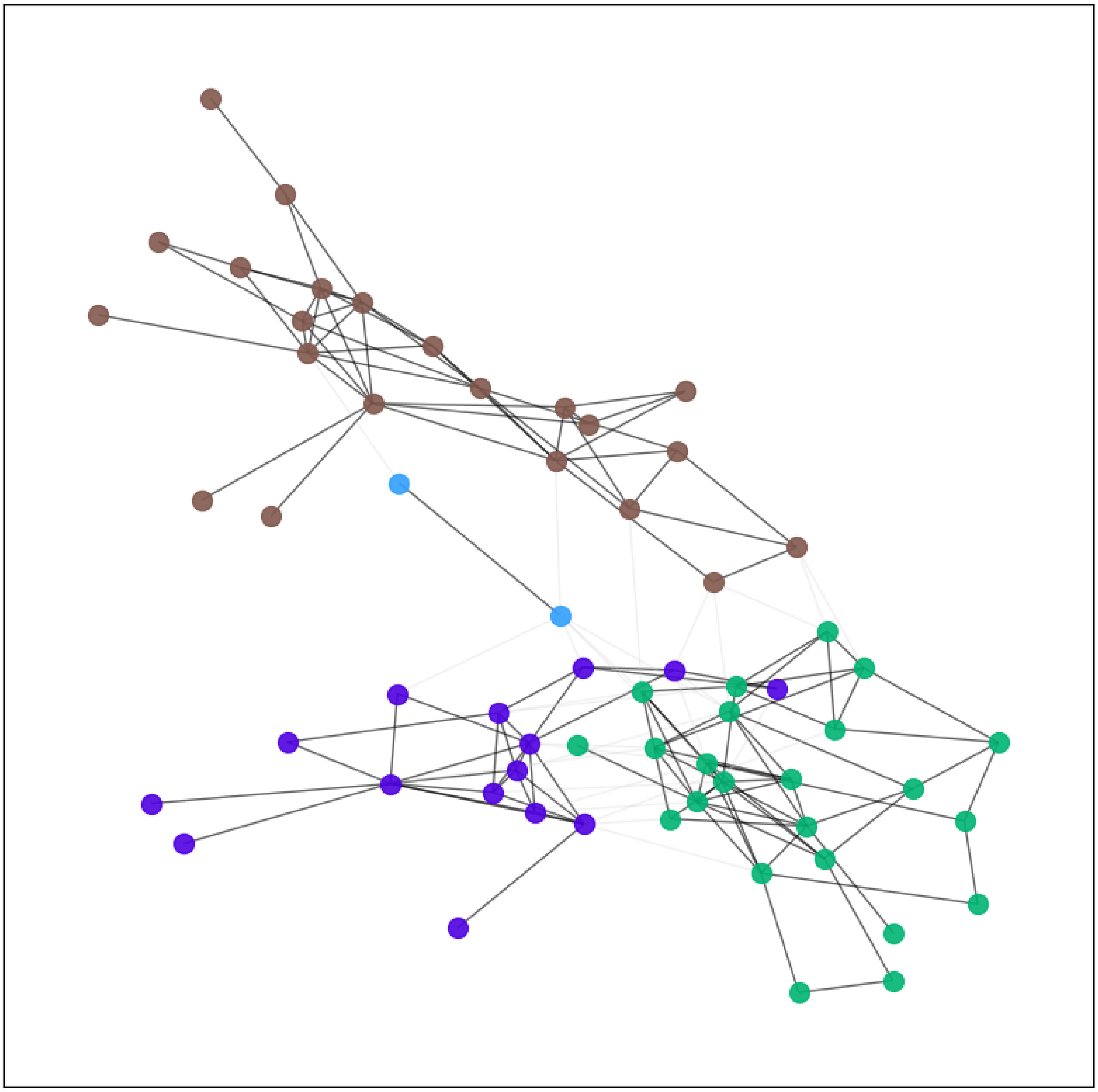
Greedy Community dolphins

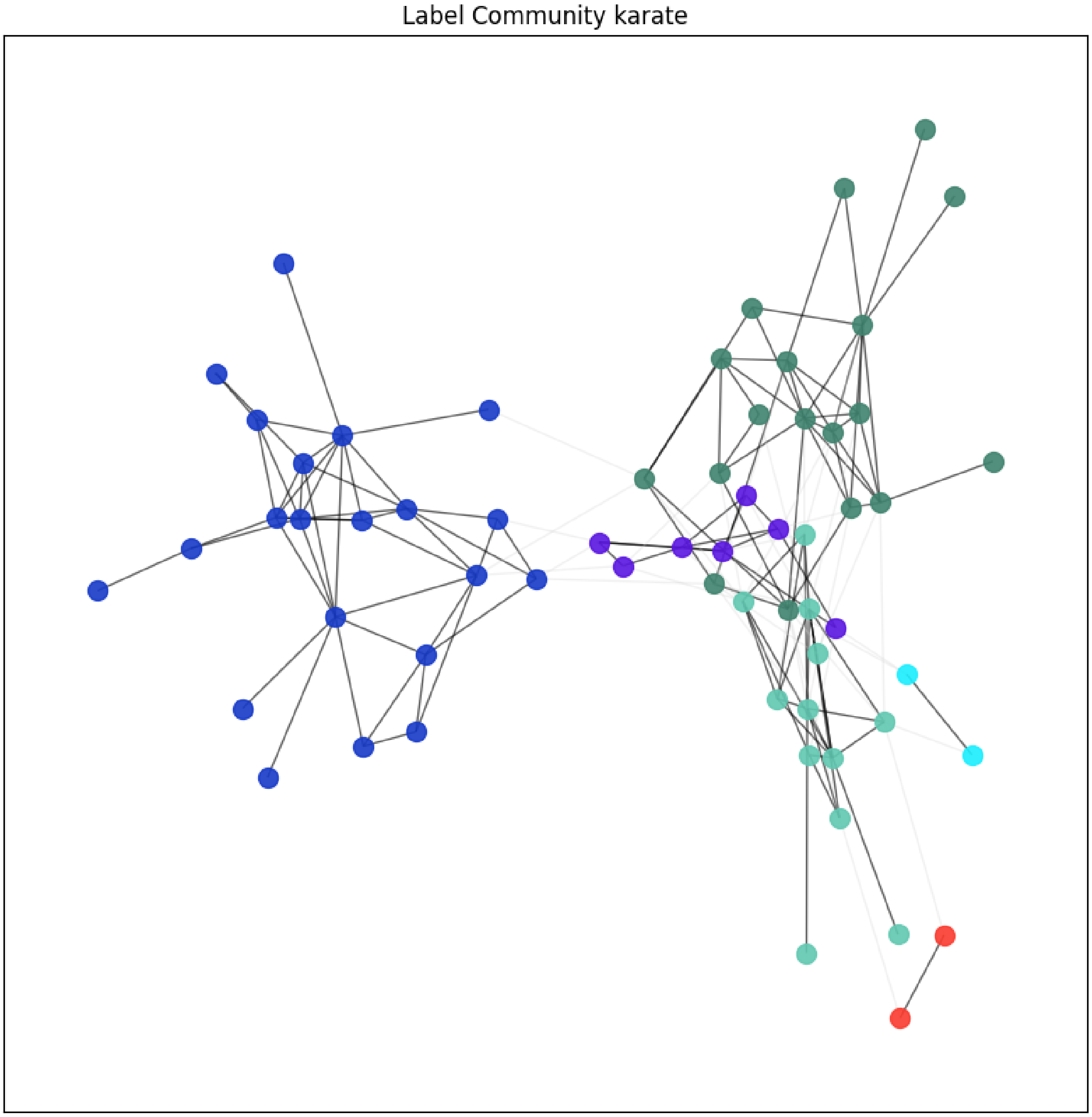


Label Community dolphins

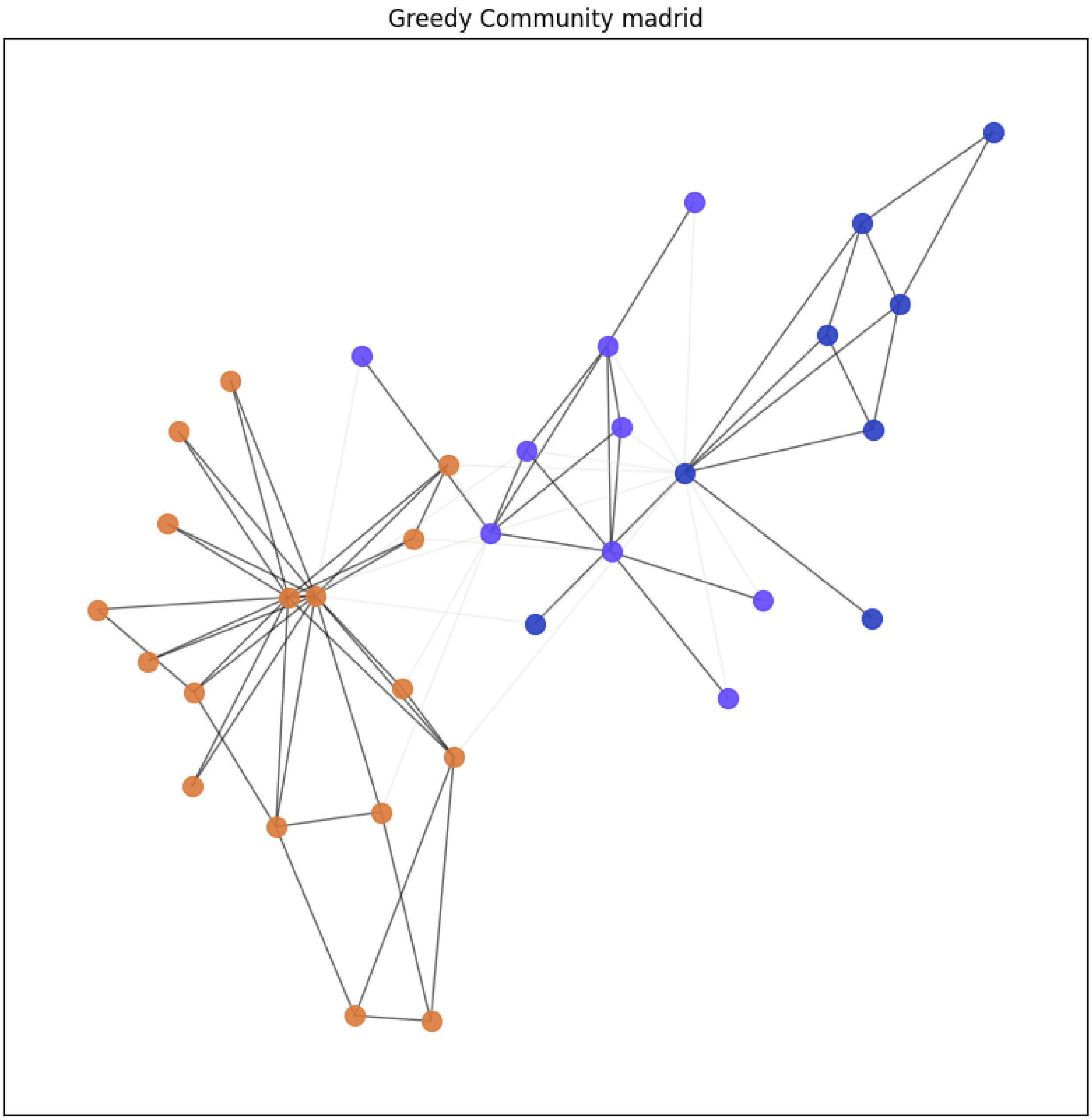


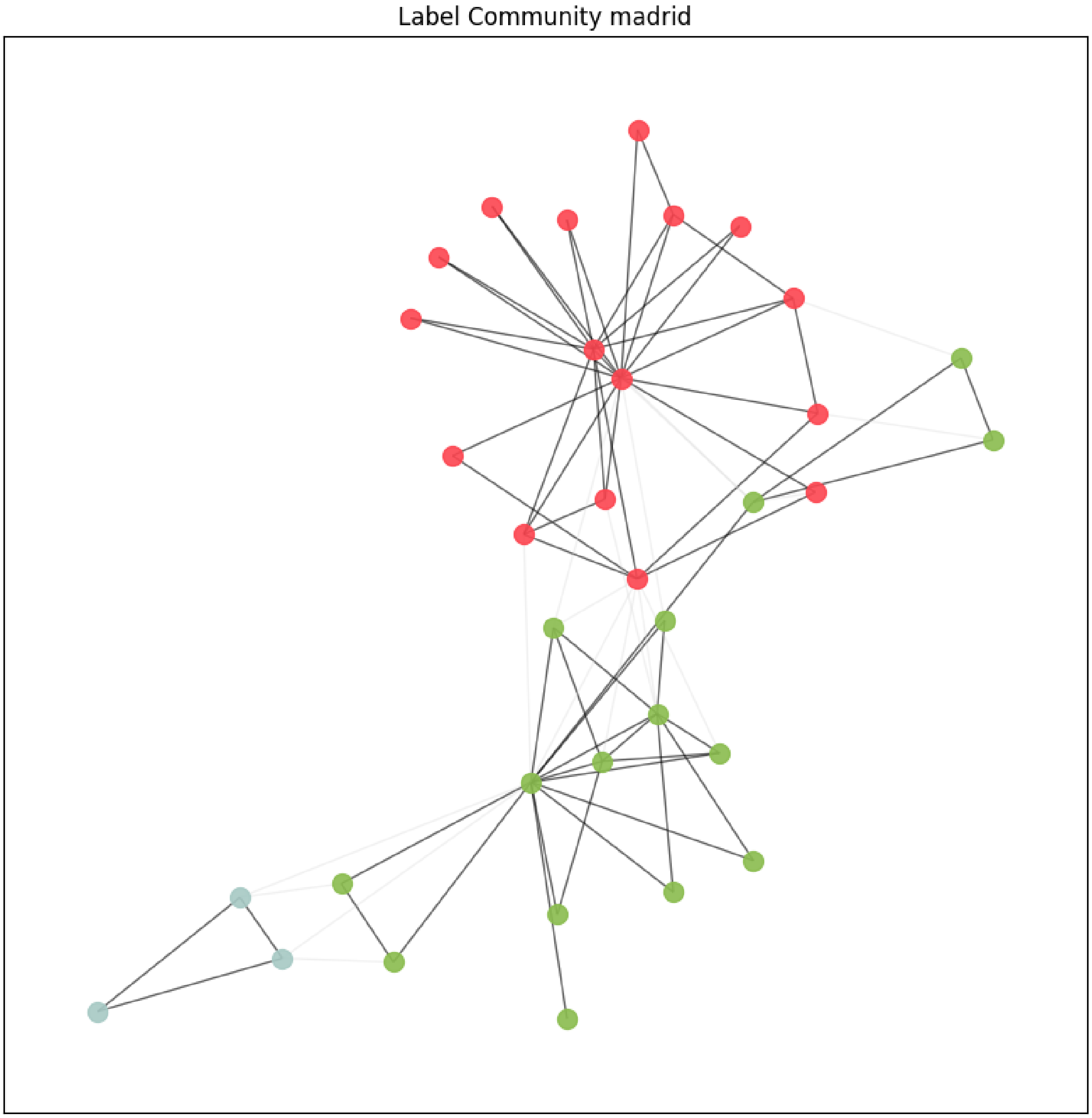
Greedy Community karate

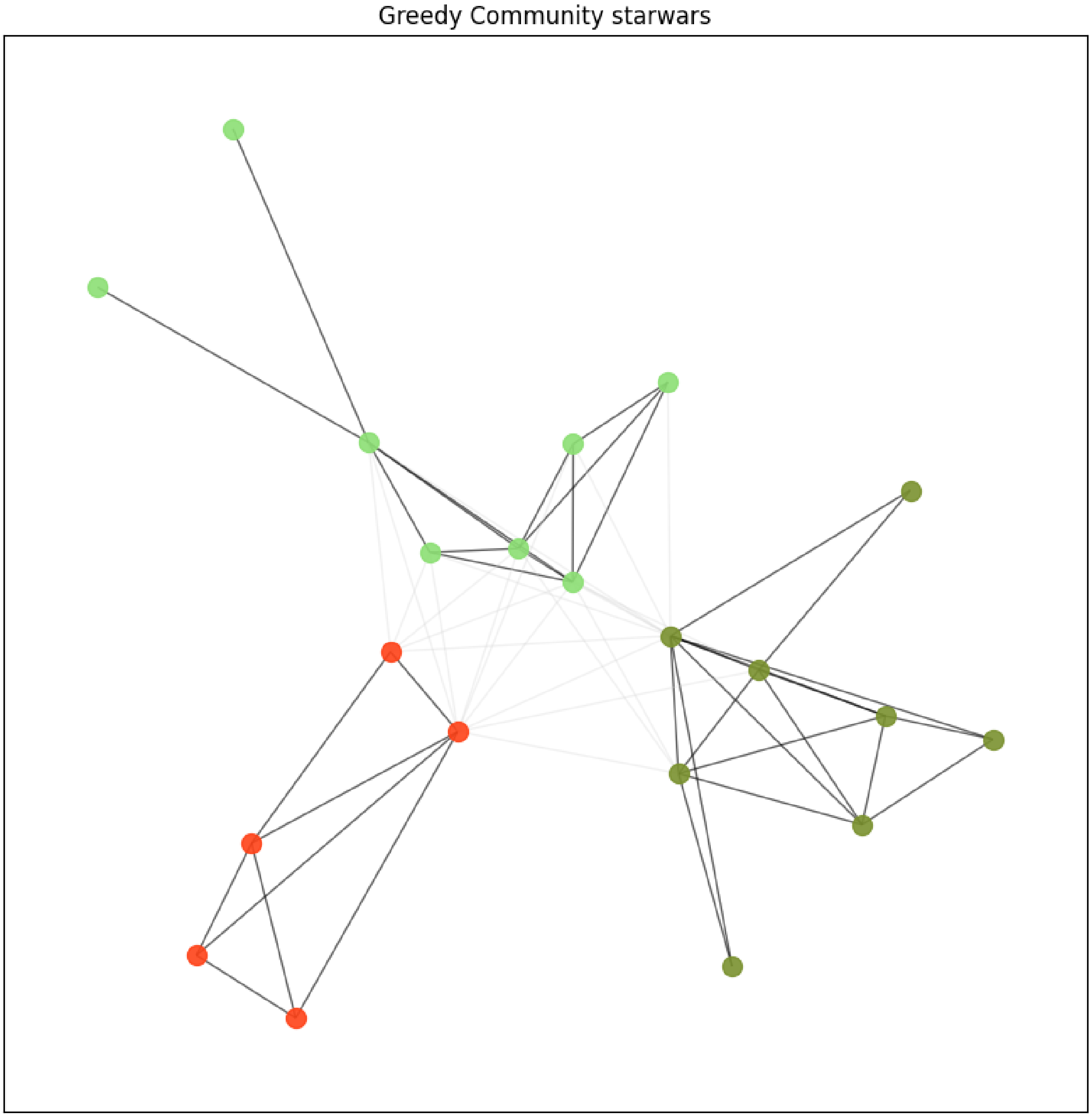


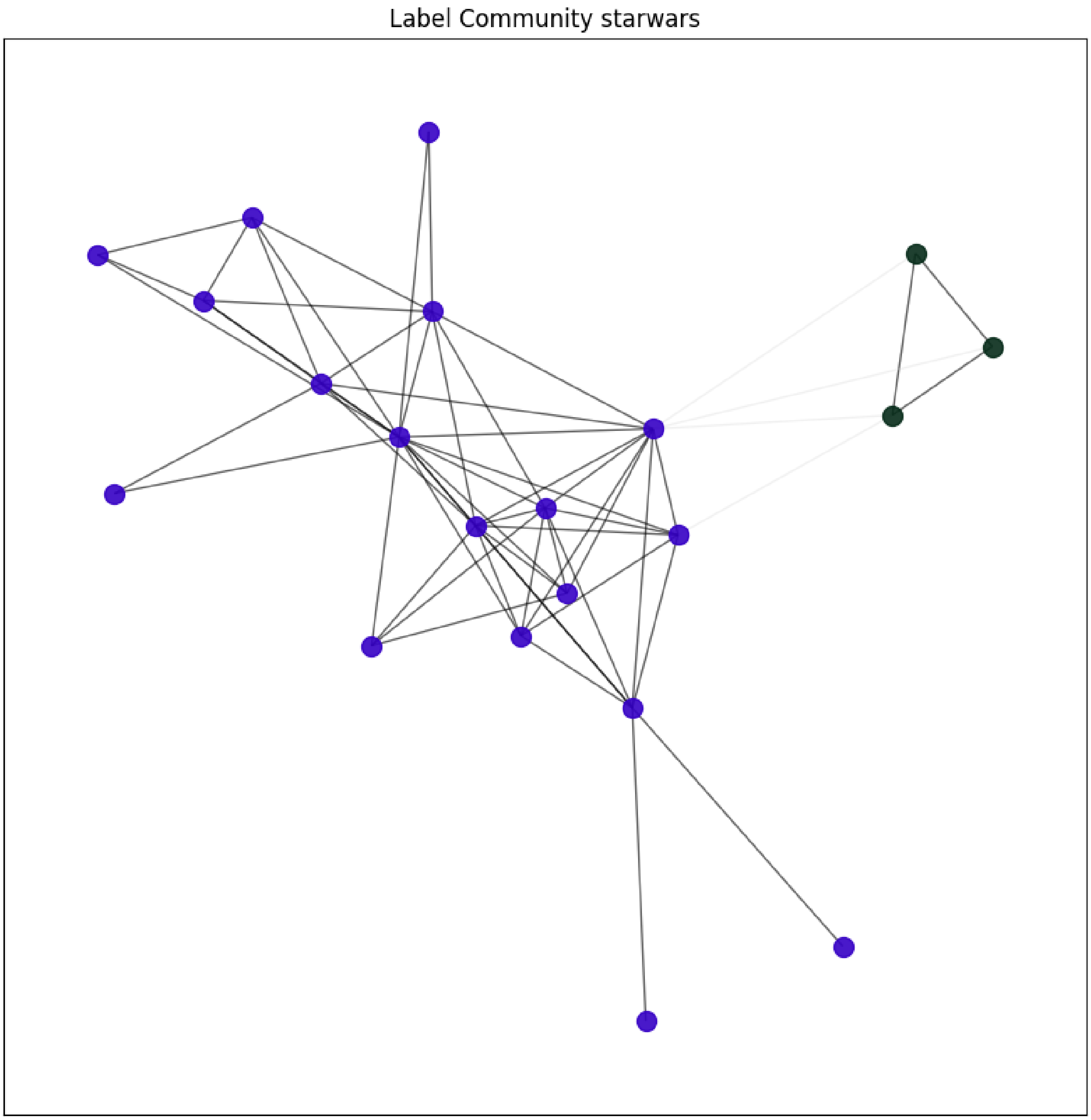


Label Community karate


Greedy Community madrid


Label Community madrid


Greedy Community starwars



Label Community starwars

#Exercise 2 **(3 points)** Randomize each network and repeat the exercise at point (1). Compare the number of communities obtained before and after randomization and the quality of community detection before and after randomization.

[17]:
```
# Function to detect communities and return the community structure and␣
↪modularity
def detect_communities(graph, algorithm="greedy"):
    if algorithm == "greedy":
        # Use Greedy Modularity Maximization
        communities = list(nx.algorithms.community.
↪greedy_modularity_communities(graph))
    elif algorithm == "label_propagation":
        # Use Label Propagation Algorithm
        communities = list(nx.algorithms.community.
 ↪label_propagation_communities(graph))

    # Calculate modularity
    modularity = nx.algorithms.community.quality.modularity(graph, communities)

    return communities, modularity
```

[18]:

```
# Function to randomize the graph while preserving the degree distribution
def randomize_graph(graph, swaps=2):
    randomized_graph = graph.copy()
    # Perform edge swaps to randomize the graph
    nx.double_edge_swap(randomized_graph, nswap=swaps * len(randomized_graph.
 ↪edges), max_tries=10000)
    return randomized_graph
```

[19]:

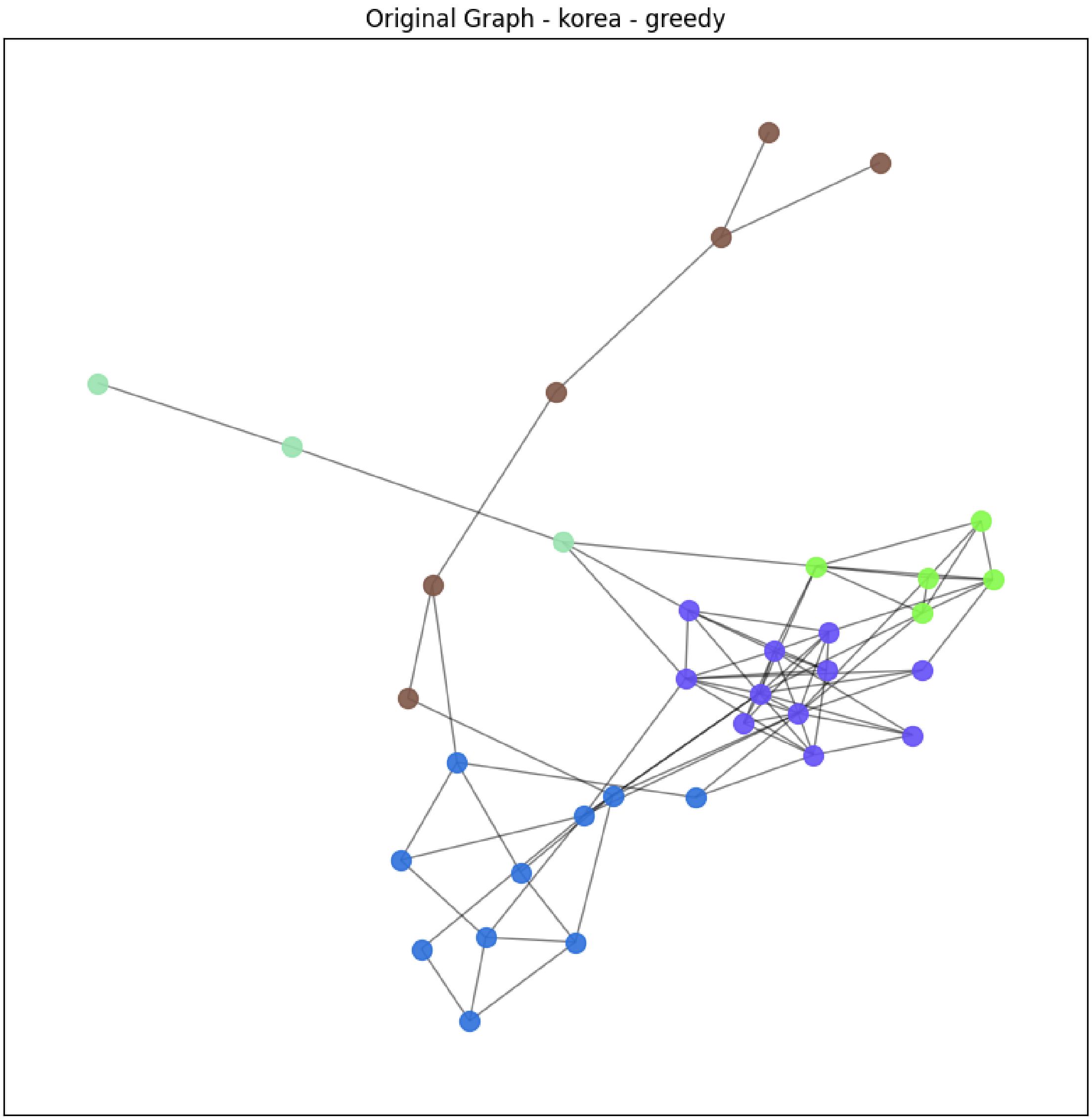
```
# Function to visualize communities
def plot_communities(graph, communities, title):
    pos = nx.spring_layout(graph)  # Layout for graph nodes
    plt.figure(figsize=(10, 10))

    # Assign different colors to each community
    colors = [f"#{random.randint(0, 0xFFFFFF):06x}" for _ in␣
 ↪range(len(communities))]

    # Draw nodes for each community
    for i, community in enumerate(communities):
        nx.draw_networkx_nodes(graph, pos, nodelist=list(community),␣
 ↪node_color=colors[i], node_size=100, alpha=0.9)

    # Draw edges
    nx.draw_networkx_edges(graph, pos, alpha=0.5)

    plt.title(title)
    plt.show()
```

[40]:

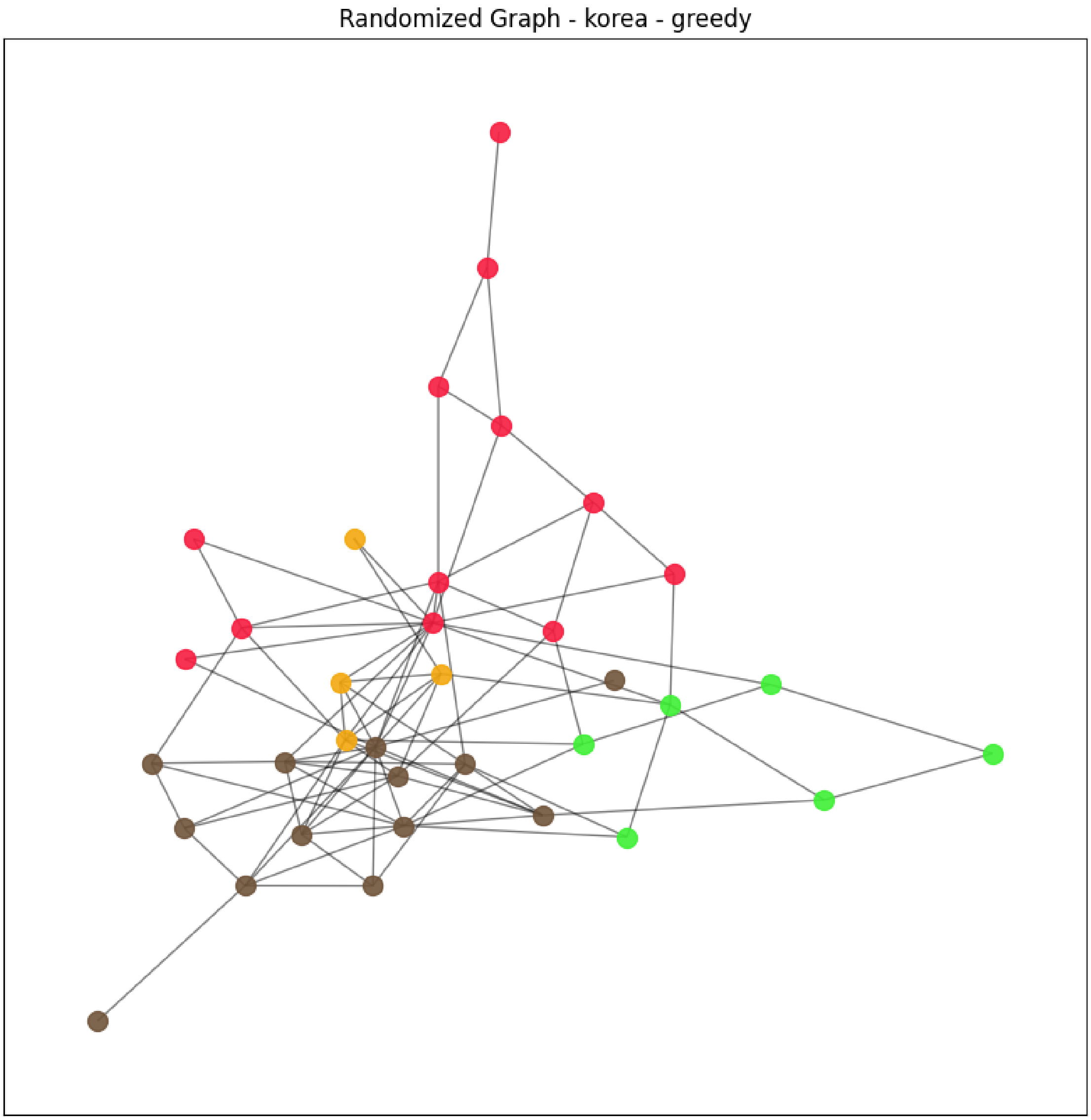
```
# Function to run community detection before and after randomization and␣
 ↪compare results
def compare_community_detection(graph, algorithm, titel):
    # Step 1: Detect communities in the original graph
    communities_original, modularity_original = detect_communities(graph,␣
 ↪algorithm=algorithm)

    # Plot original communities
    plot_communities(graph, communities_original, "Original Graph - " + titel +␣
↪" - " + algorithm)

    print(f"{titel}: Number of communities (Original Graph):␣
↪{len(communities_original)}")
    print(f"{titel}: Modularity (Original Graph): {modularity_original:.4f}")

    # Step 2: Randomize the graph
    randomized_graph = randomize_graph(graph)

    # Step 3: Detect communities in the randomized graph
    communities_randomized, modularity_randomized =␣
↪detect_communities(randomized_graph, algorithm=algorithm)

    # Plot randomized communities
    plot_communities(randomized_graph, communities_randomized, "Randomized␣
↪Graph - " + titel + " - " + algorithm)

    print(f"{titel}: Number of communities (Randomized Graph):␣
↪{len(communities_randomized)}")
    print(f"{titel}: Modularity (Randomized Graph): {modularity_randomized:.
↪4f}")
```

[41]:

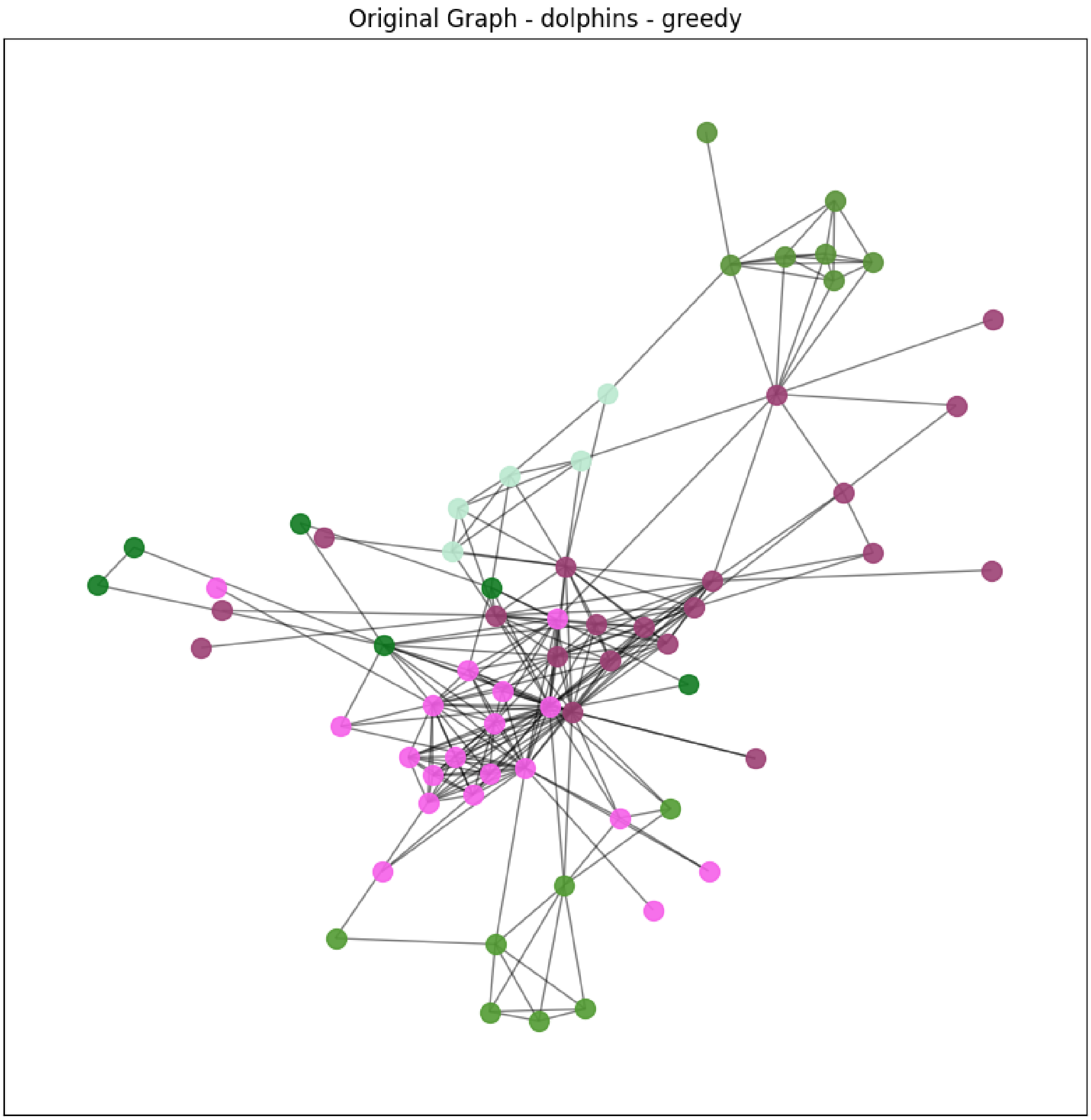
```
for graph, titel in zip(graphs, titels):
  compare_community_detection(graph, "greedy", titel)
```



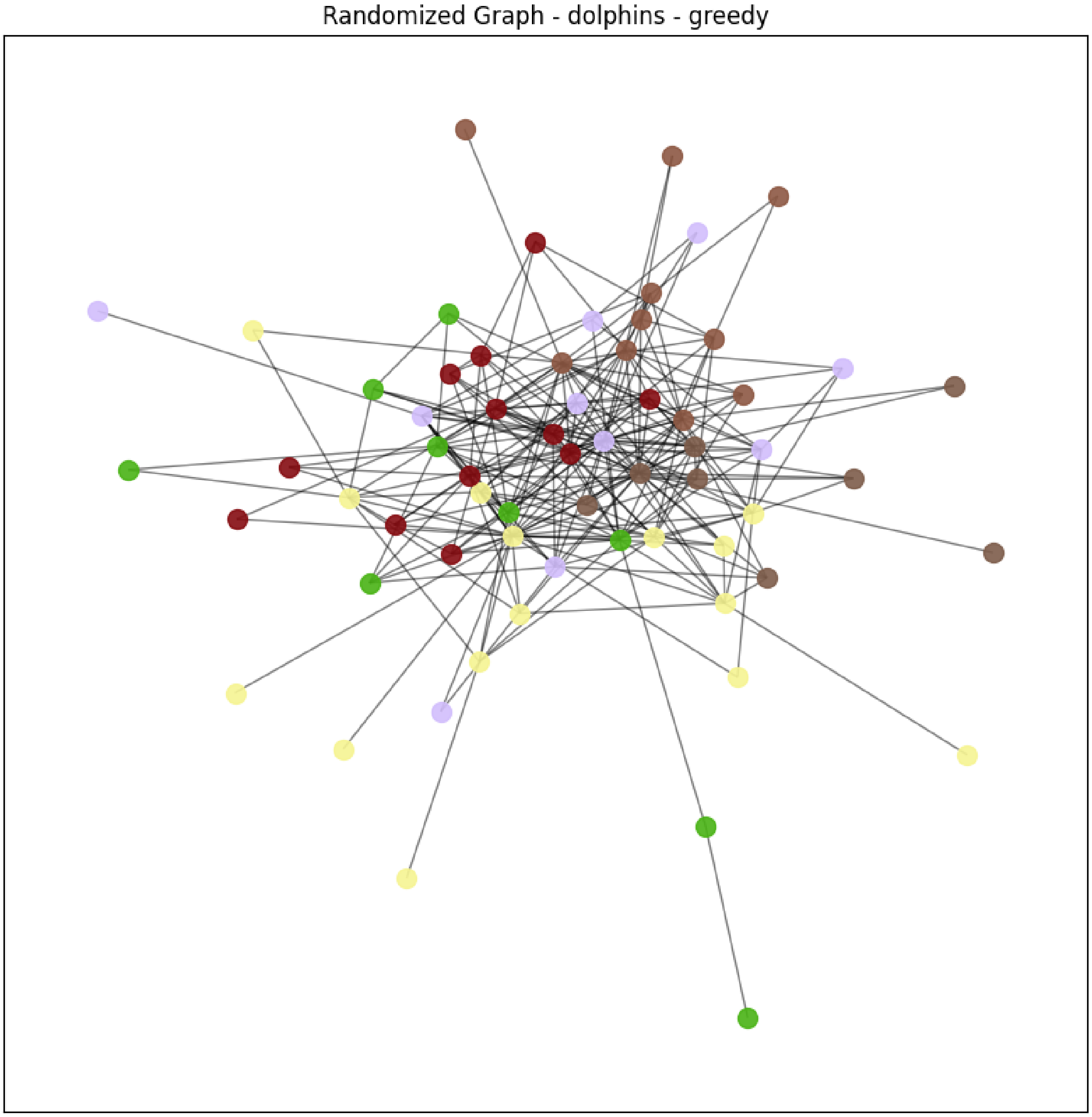
```
korea: Number of communities (Original Graph): 5
korea: Modularity (Original Graph): 0.4471
```



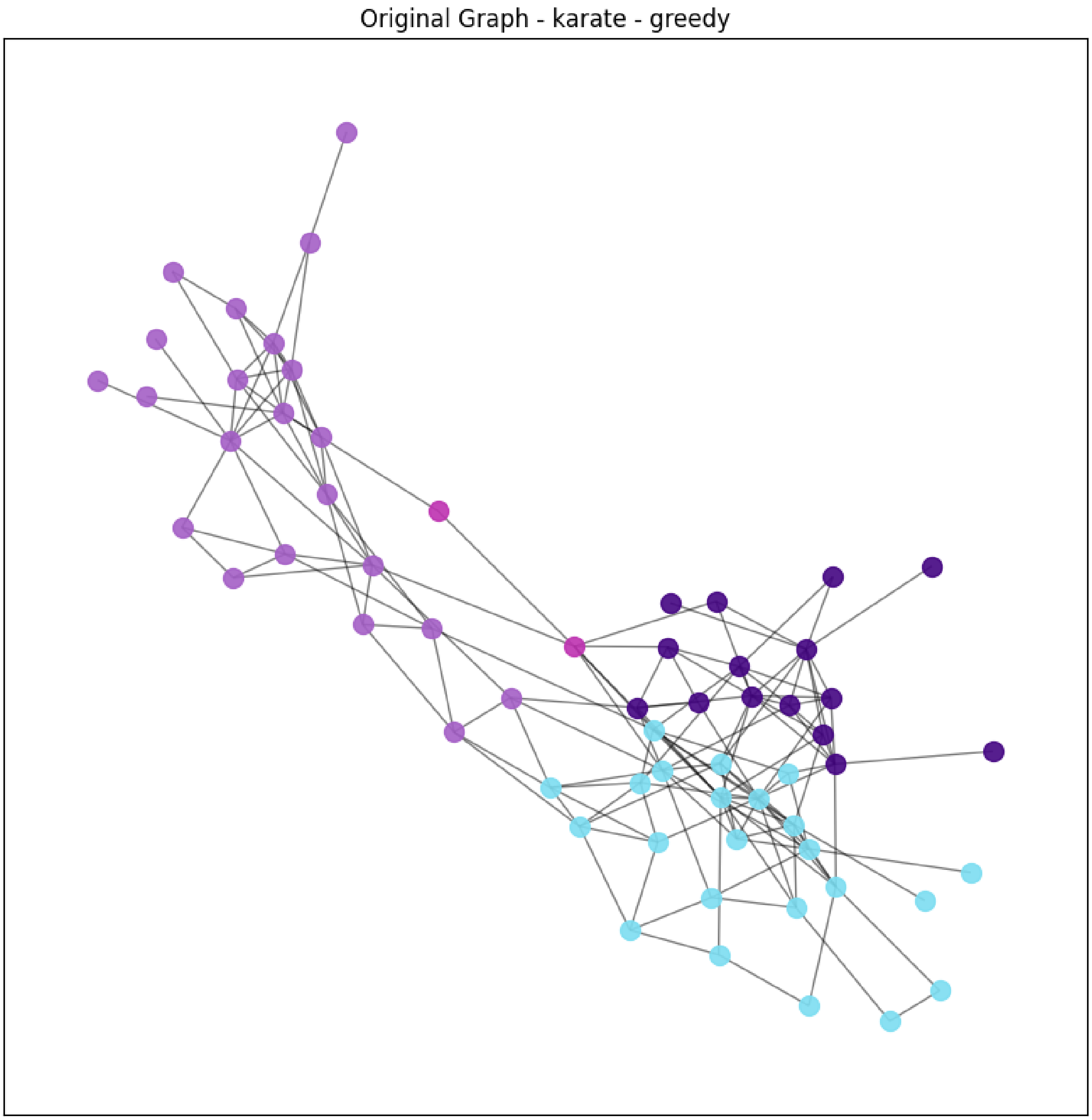
```
korea: Number of communities (Randomized Graph): 4
korea: Modularity (Randomized Graph): 0.3320
```



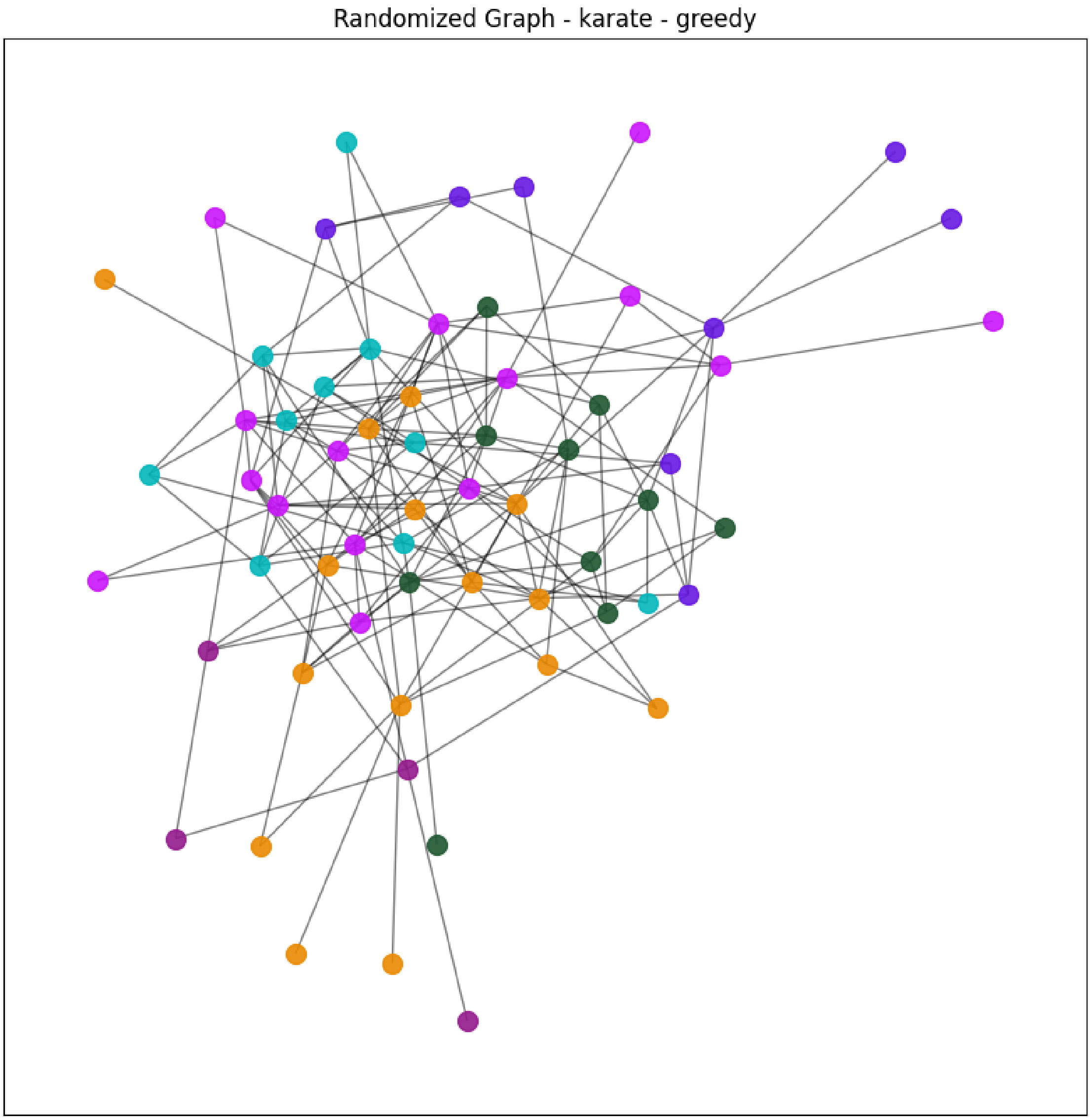
```
dolphins: Number of communities (Original Graph): 6
dolphins: Modularity (Original Graph): 0.4103
```



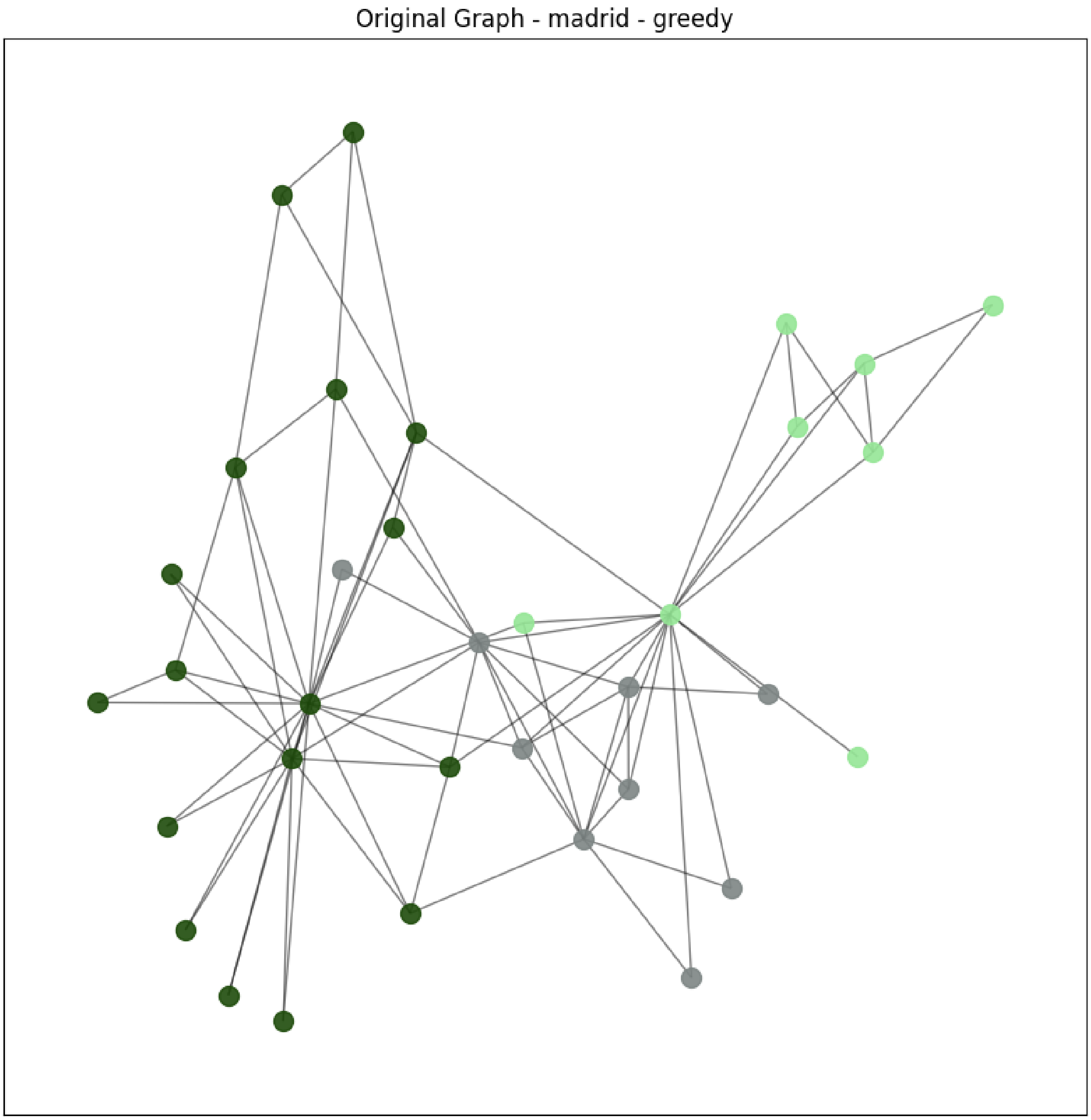
```
dolphins: Number of communities (Randomized Graph): 6
dolphins: Modularity (Randomized Graph): 0.2199
```



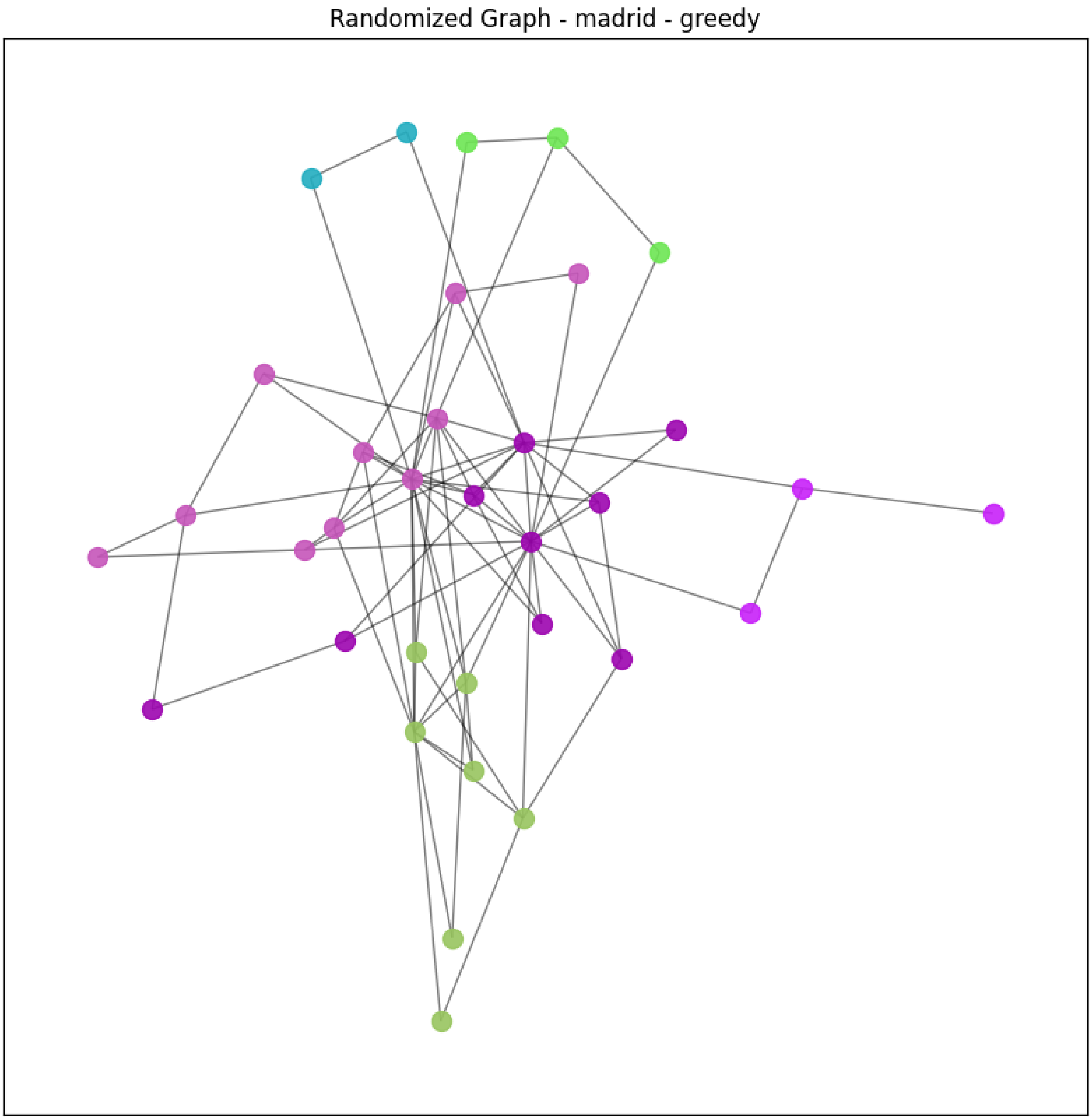
```
karate: Number of communities (Original Graph): 4
karate: Modularity (Original Graph): 0.4955
```



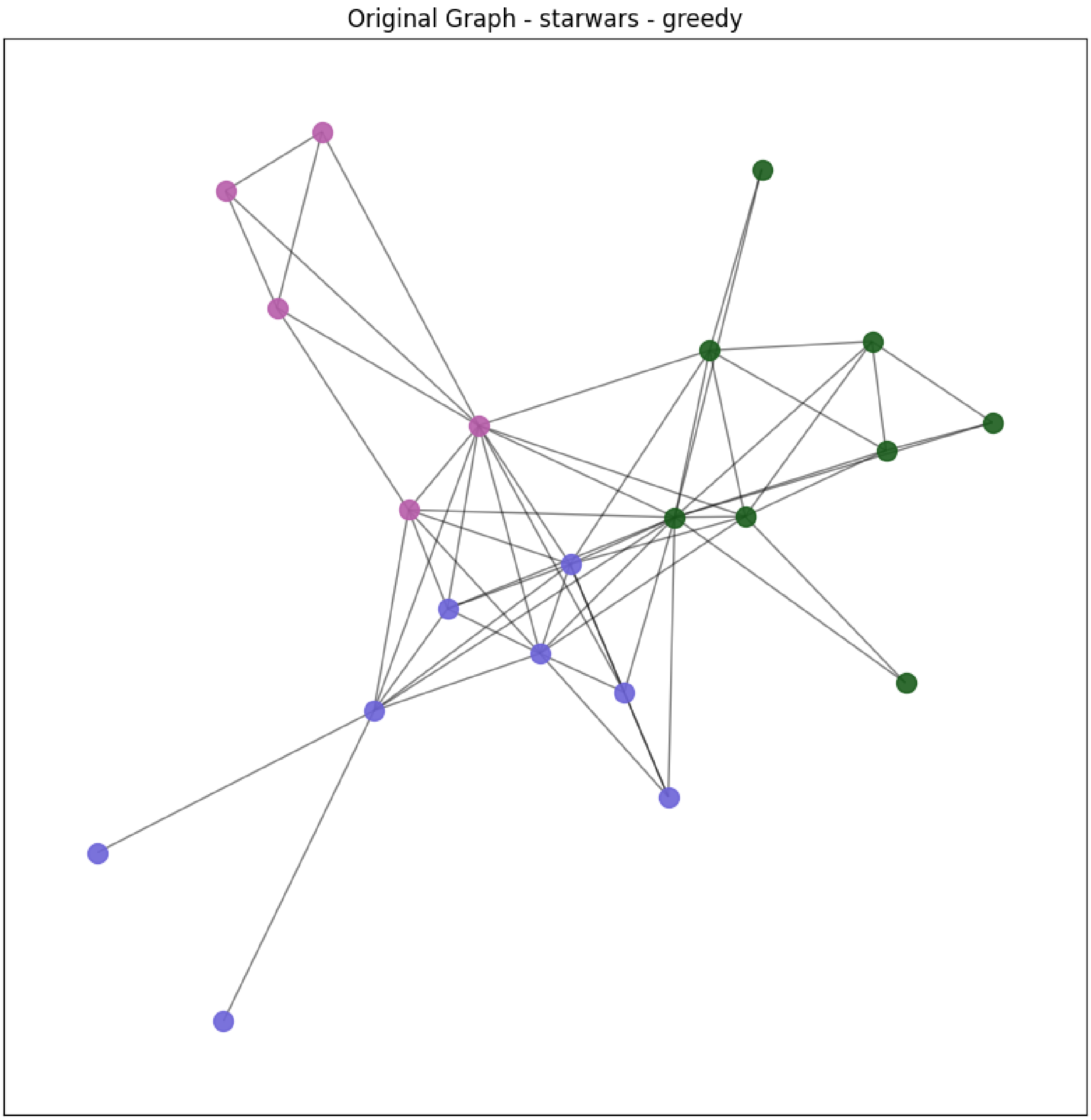
```
karate: Number of communities (Randomized Graph): 6
karate: Modularity (Randomized Graph): 0.3465
```



```
madrid: Number of communities (Original Graph): 3
madrid: Modularity (Original Graph): 0.3807
```

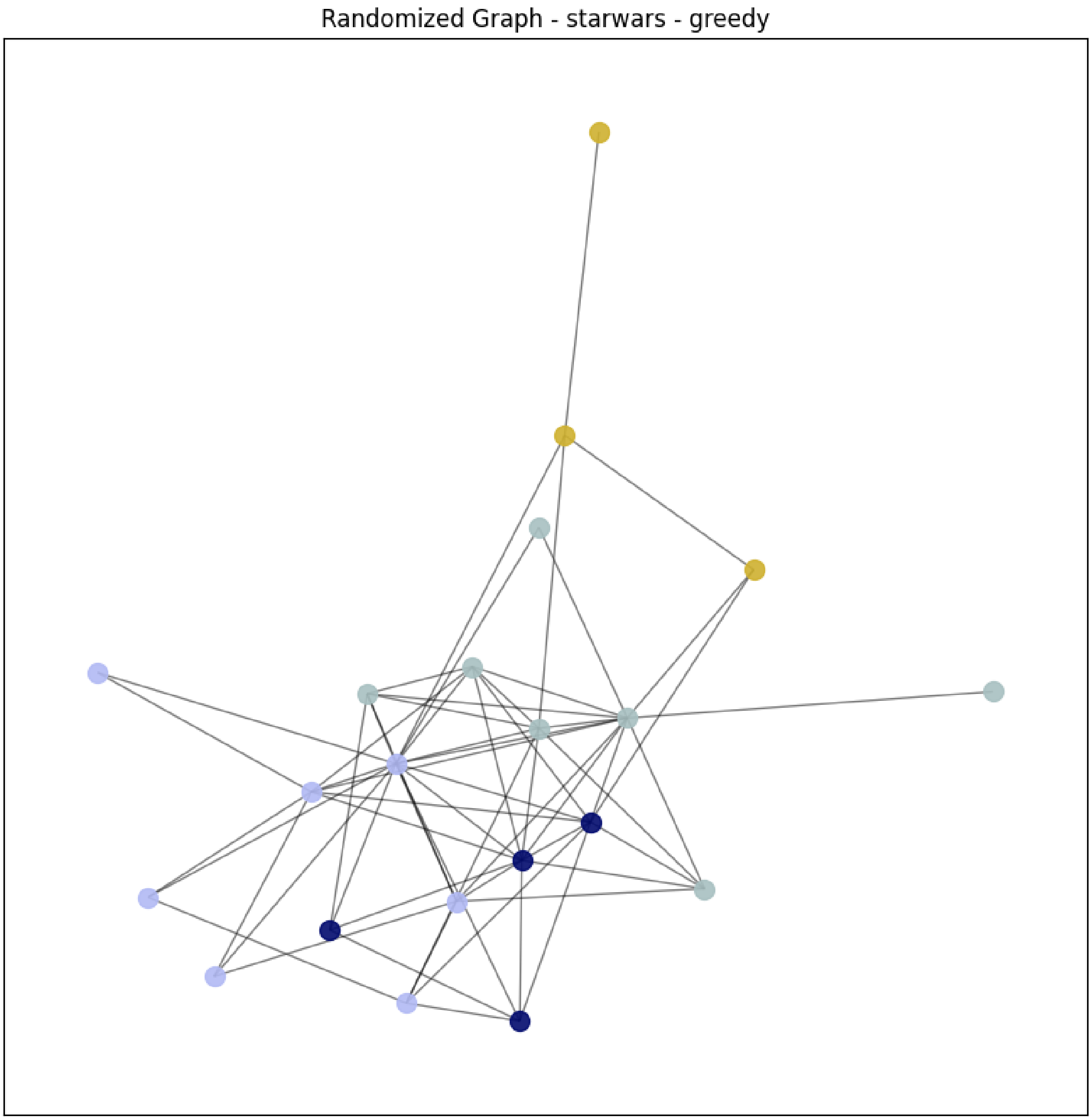


```
madrid: Number of communities (Randomized Graph): 6
madrid: Modularity (Randomized Graph): 0.2953
```



Original Graph - starwars - greedy

```
starwars: Number of communities (Original Graph): 3
starwars: Modularity (Original Graph): 0.2871
```

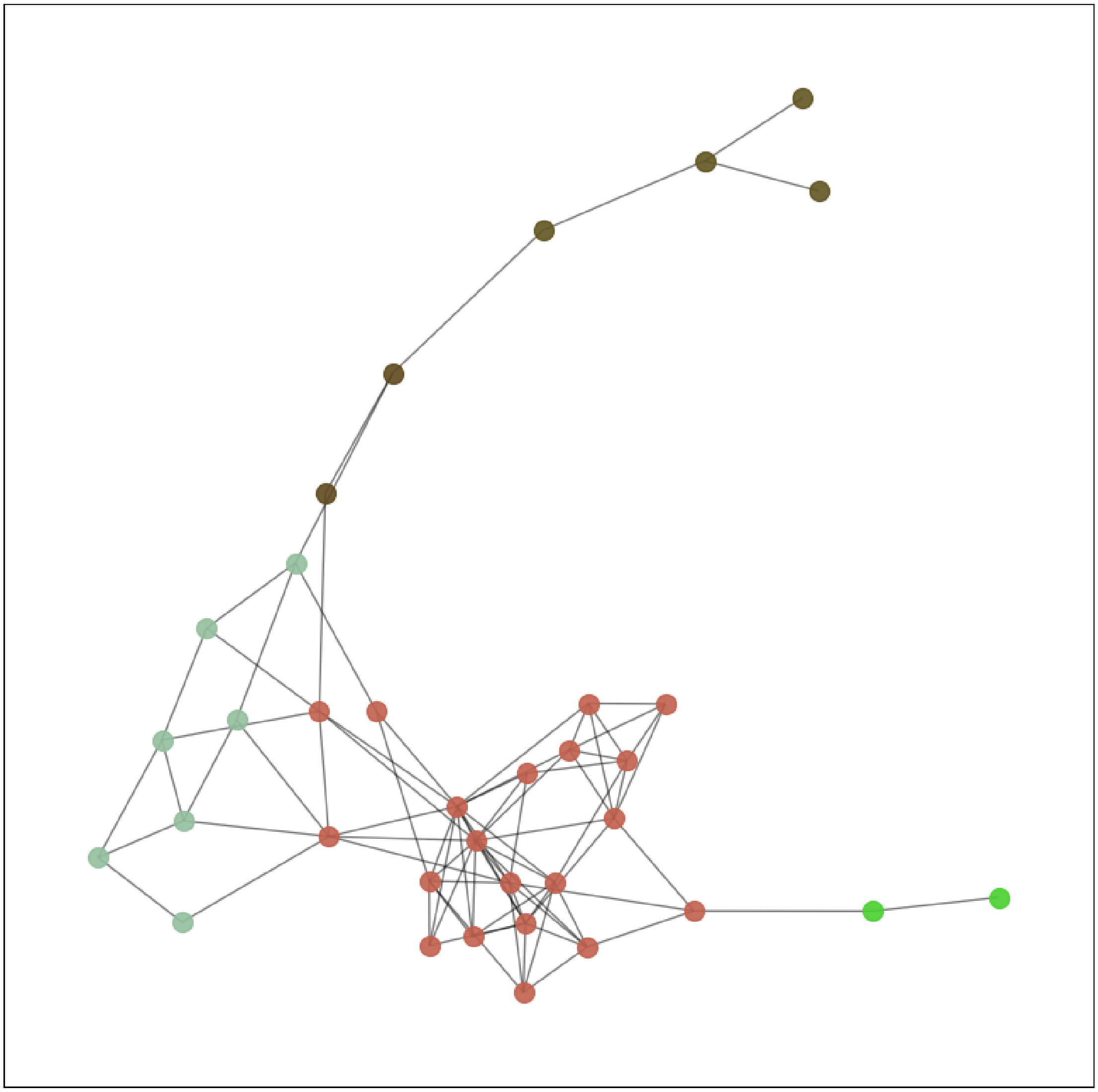


```
starwars: Number of communities (Randomized Graph): 4
starwars: Modularity (Randomized Graph): 0.1626
```

Mit Label propagation

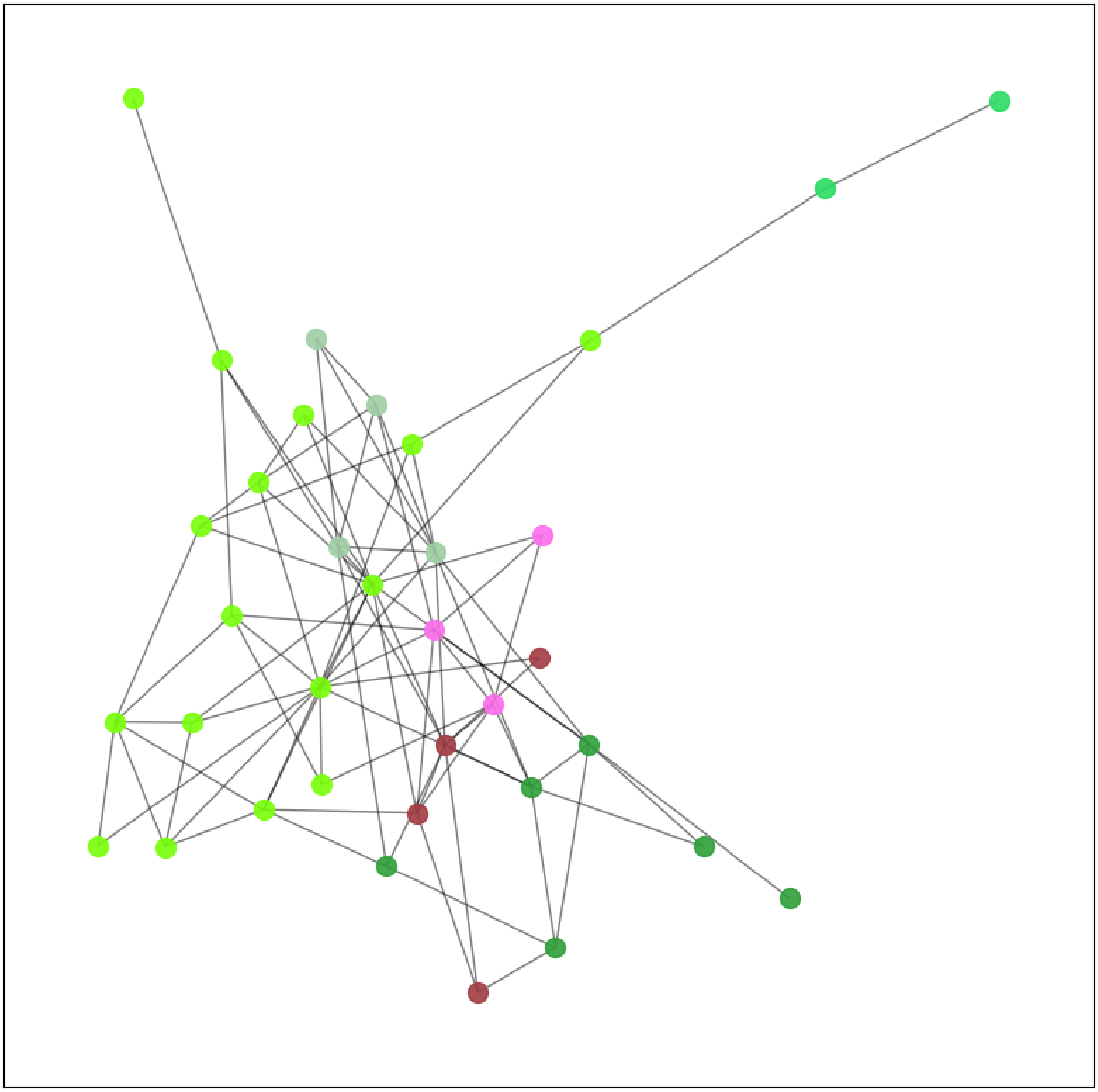
```
[42]: for graph, titel in zip(graphs, titels):
          compare_community_detection(graph, "label_propagation", titel)
```

Original Graph - korea - label_propagation

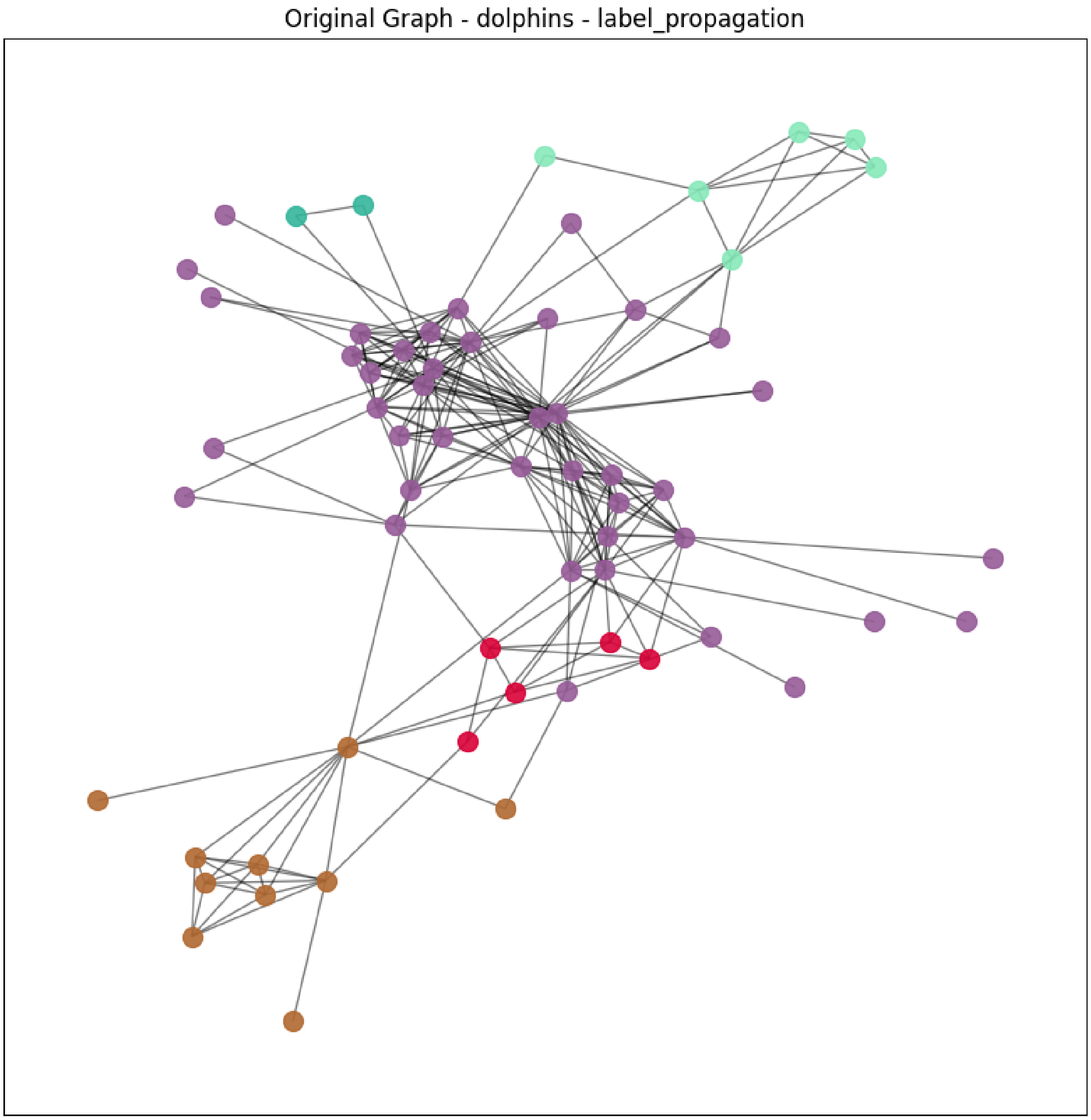


```
korea: Number of communities (Original Graph): 5
korea: Modularity (Original Graph): 0.2605
```

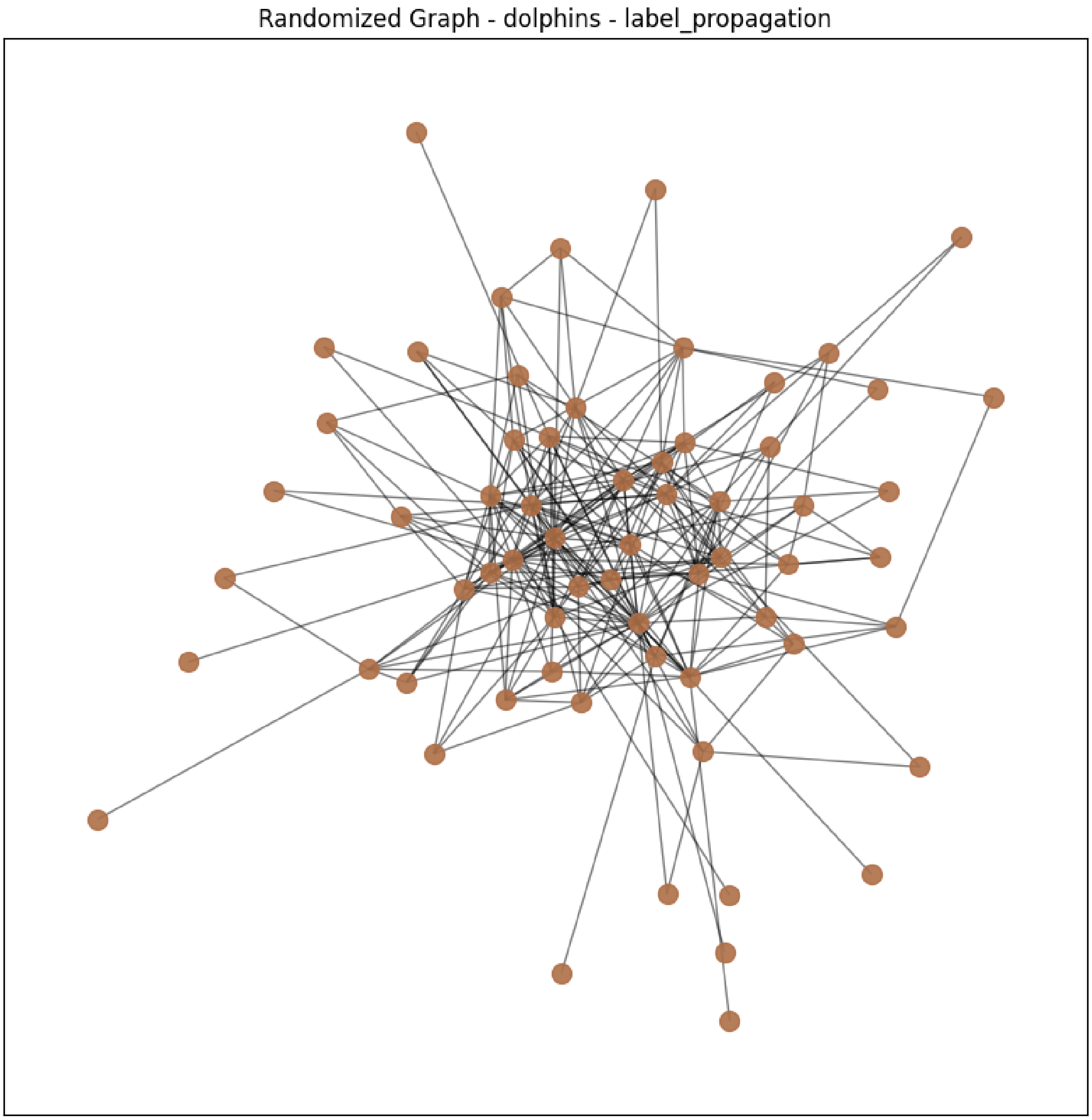
Randomized Graph - korea - label_propagation



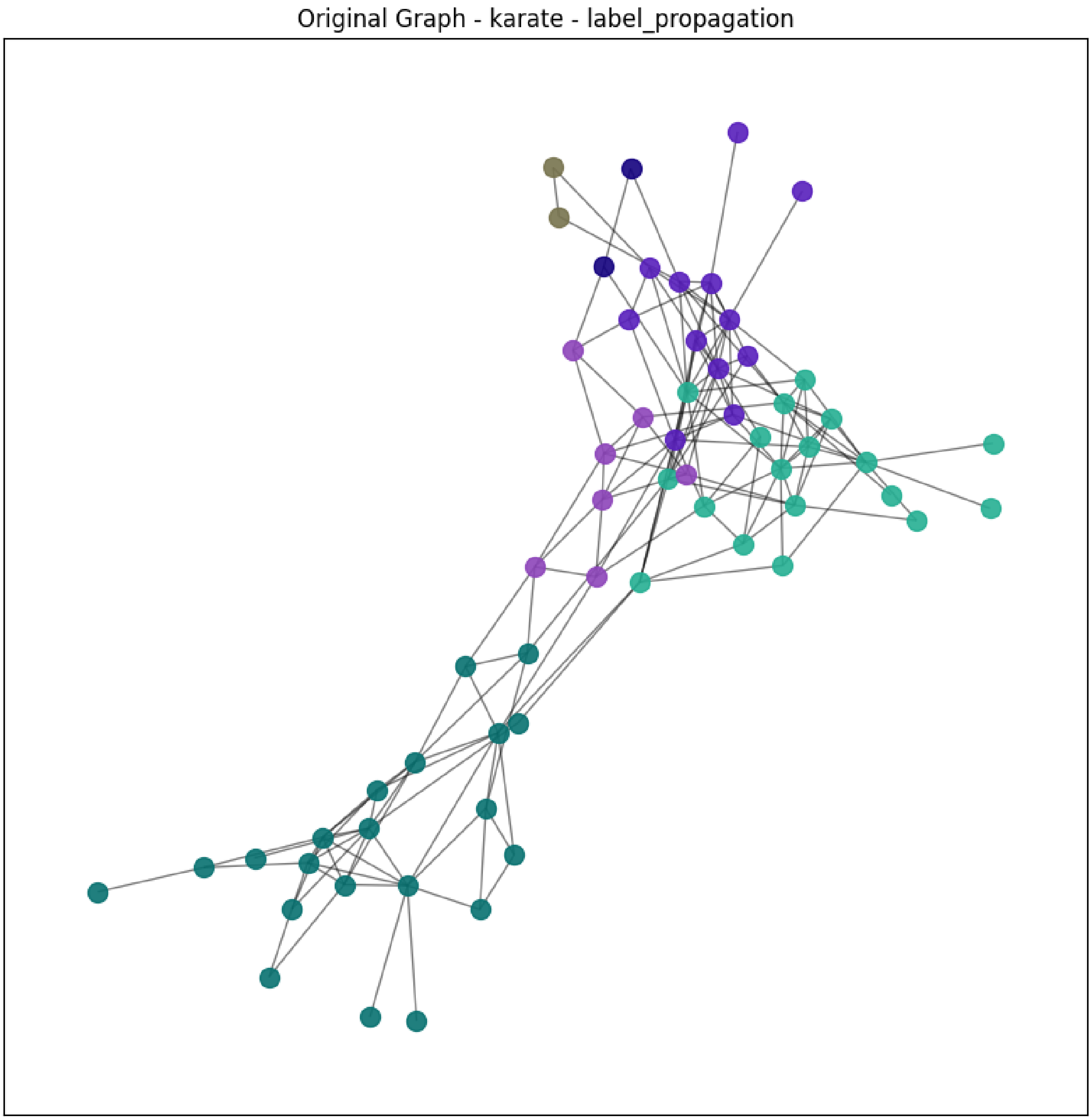
```
korea: Number of communities (Randomized Graph): 6
korea: Modularity (Randomized Graph): 0.3262
```



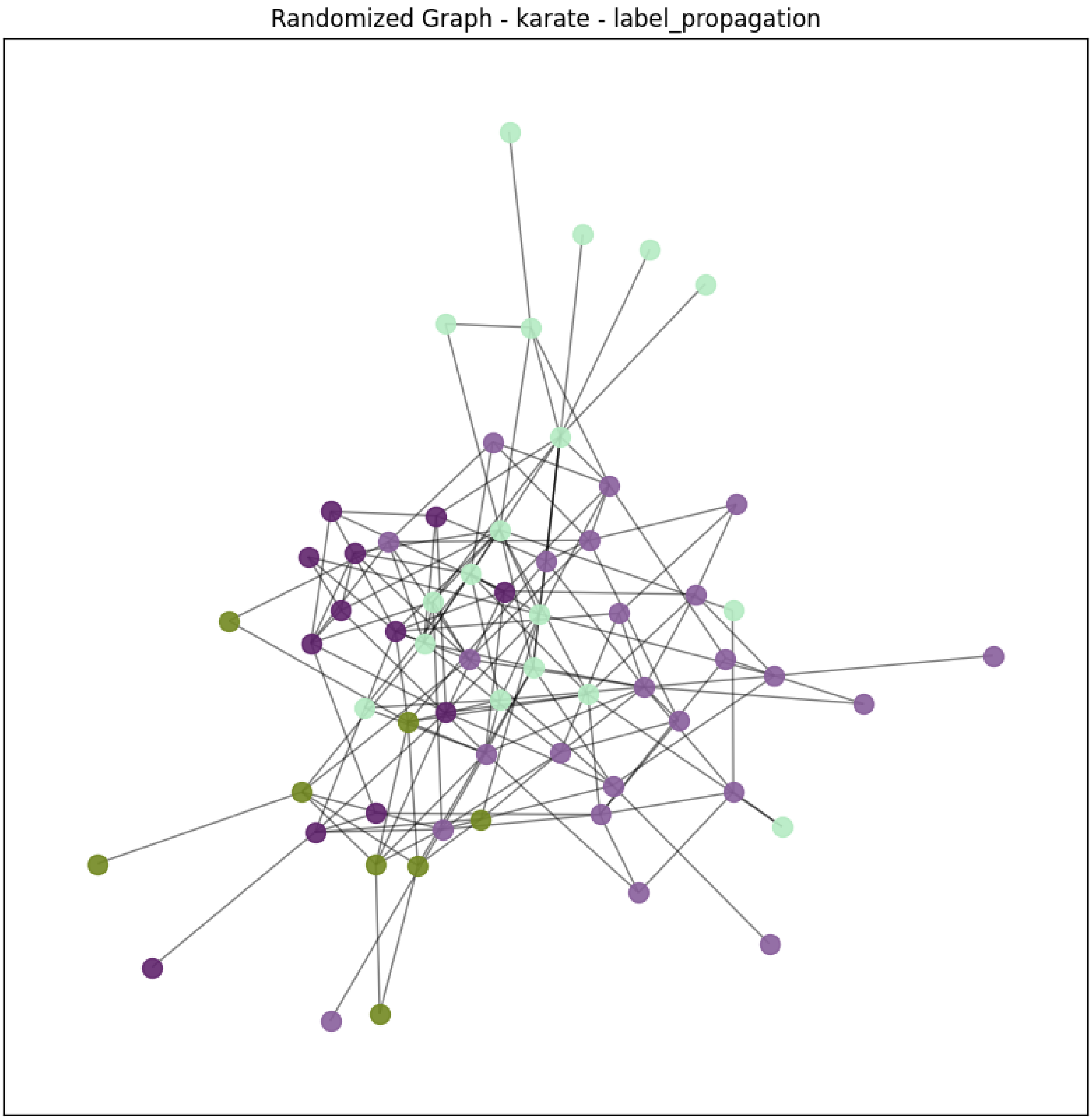
```
dolphins: Number of communities (Original Graph): 5
dolphins: Modularity (Original Graph): 0.2873
```



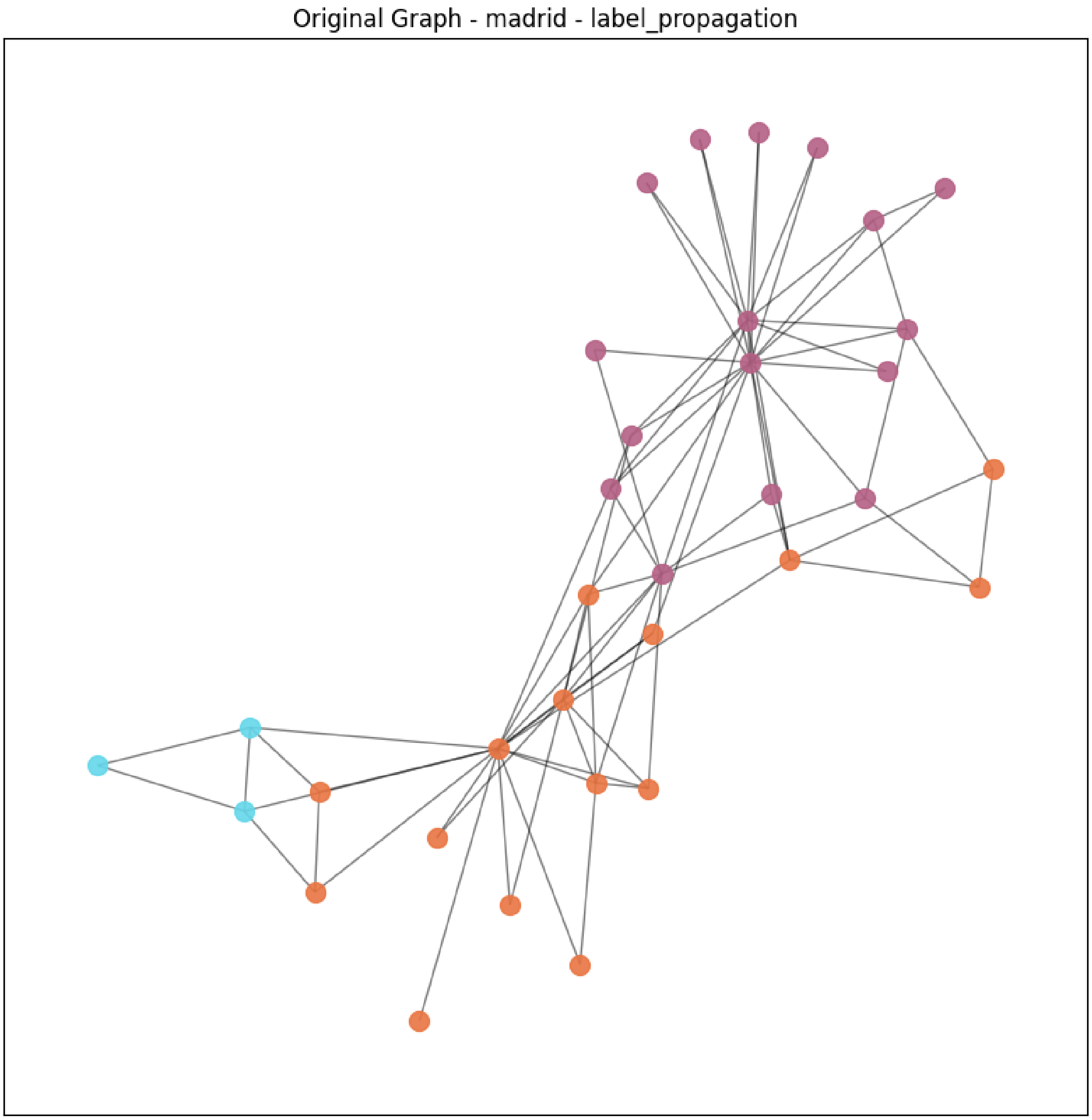
```
dolphins: Number of communities (Randomized Graph): 1
dolphins: Modularity (Randomized Graph): 0.0000
```



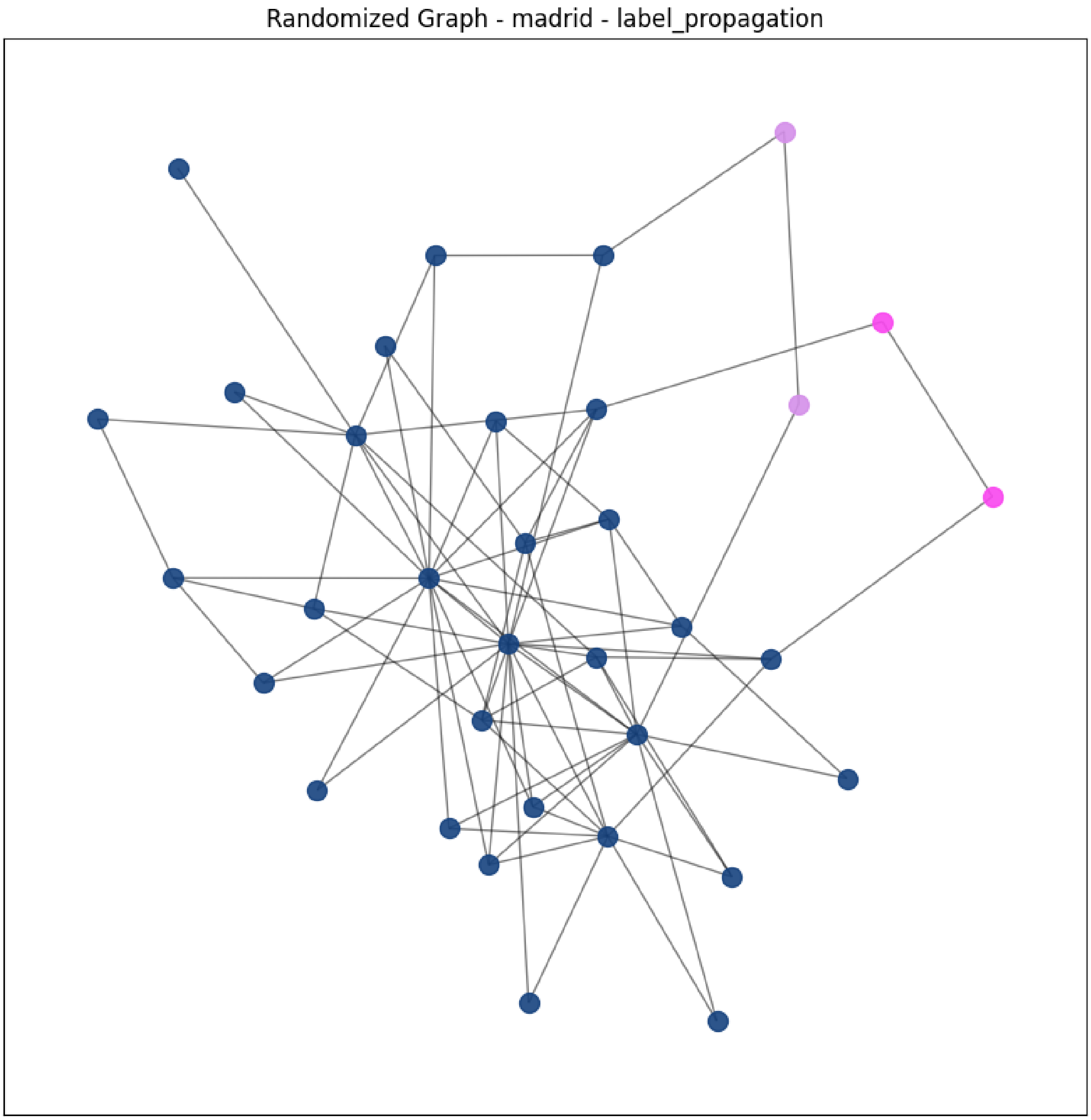
```
karate: Number of communities (Original Graph): 6
karate: Modularity (Original Graph): 0.4986
```



```
karate: Number of communities (Randomized Graph): 4
karate: Modularity (Randomized Graph): 0.3257
```

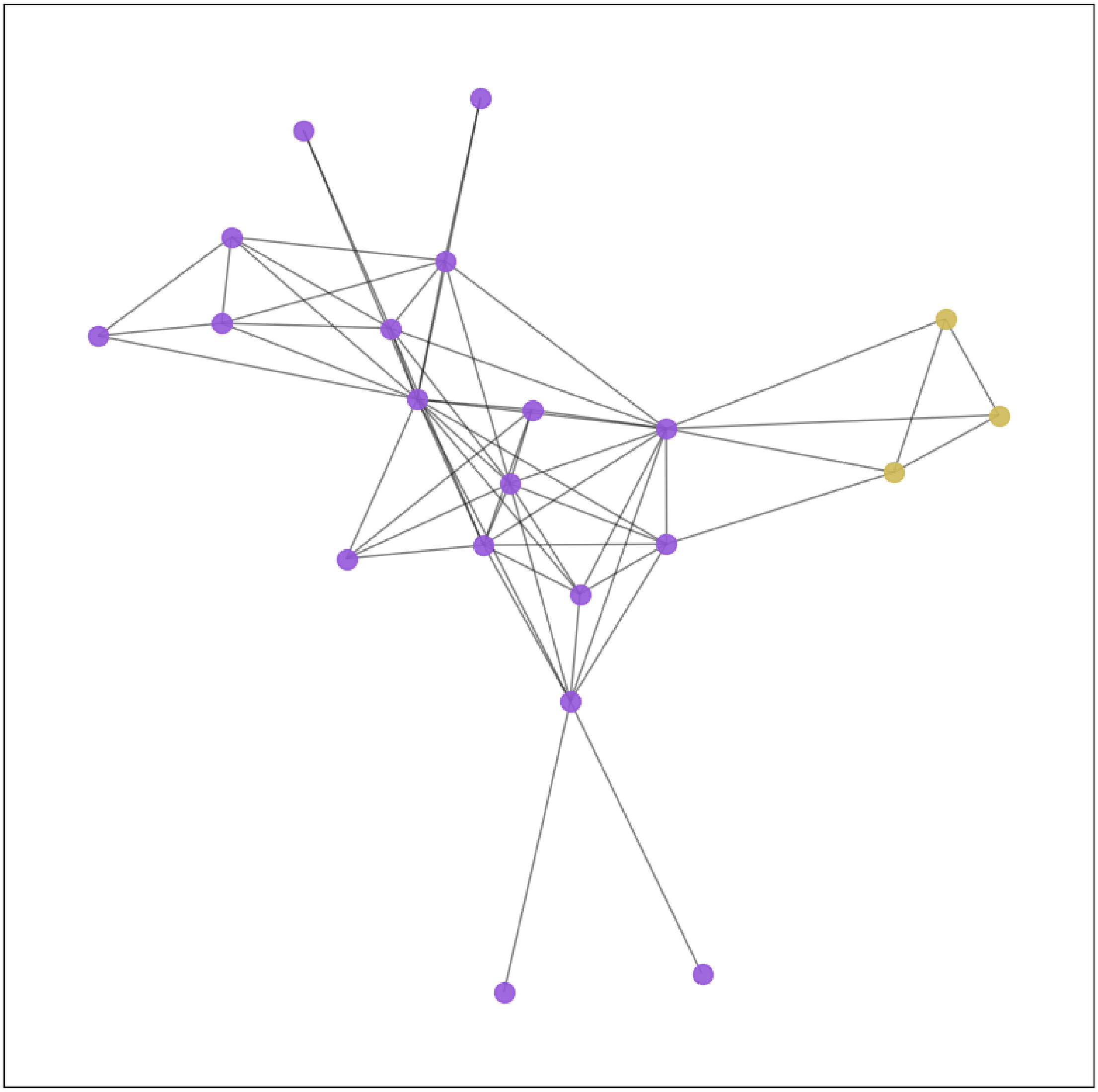


```
madrid: Number of communities (Original Graph): 3
madrid: Modularity (Original Graph): 0.3251
```

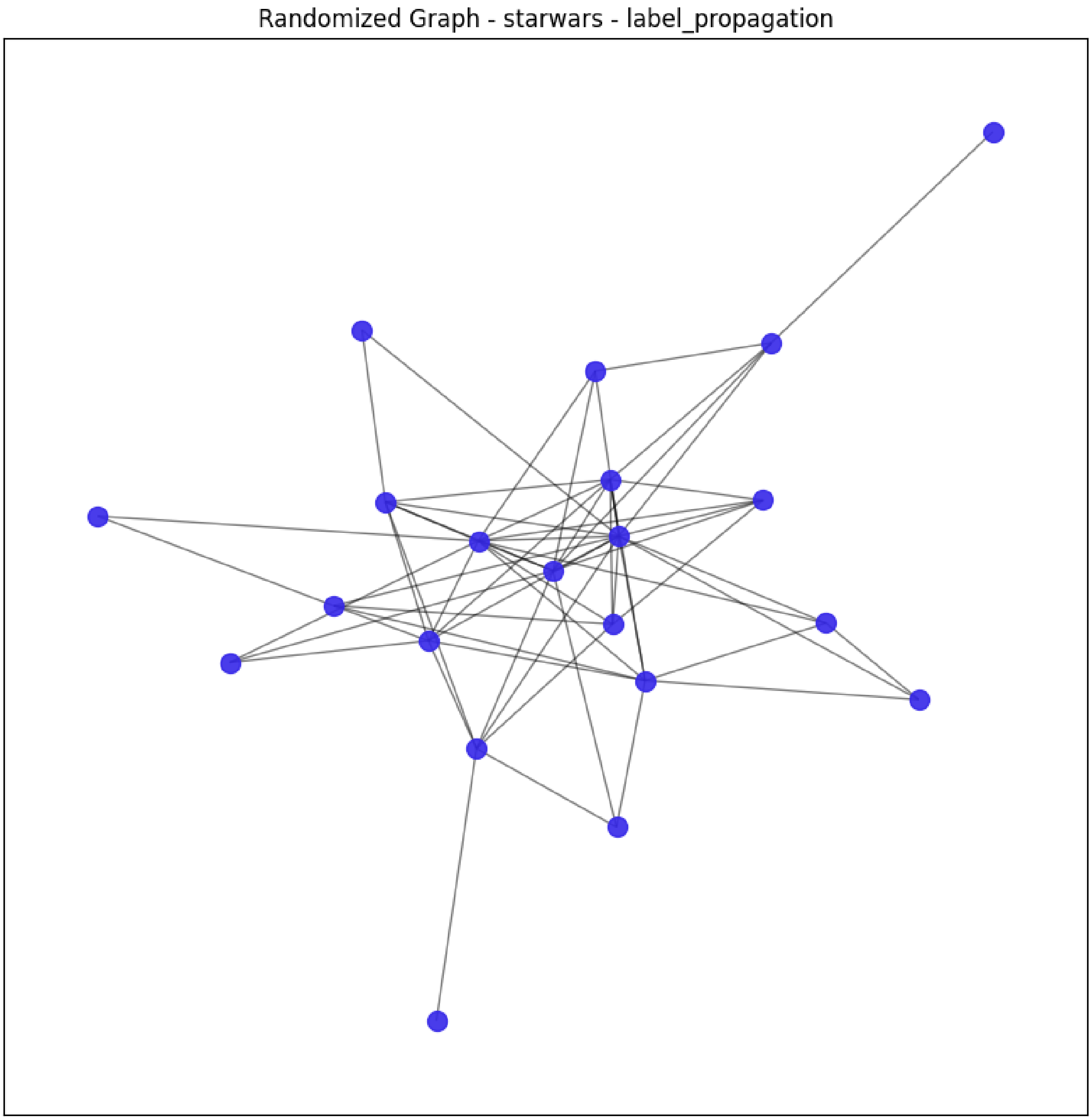


```
madrid: Number of communities (Randomized Graph): 3
madrid: Modularity (Randomized Graph): 0.0473
```

Original Graph - starwars - label_propagation



```
starwars: Number of communities (Original Graph): 2
starwars: Modularity (Original Graph): 0.0861
```



```
starwars: Number of communities (Randomized Graph): 1
starwars: Modularity (Randomized Graph): 0.0000
```

Before Randomization: You observed well-defined communities with a high modularity score (indicating strong community structure). After Randomization: The communities are less distinct, and the modularity score is lower, indicating a loss of meaningful community structure in the random graph.

[53]:

```
# Function to collect number of communities and modularity for both algorithms
def collect_community_data(graph, graph_name):
    # Initialize a dictionary to store the data
    data = {
        "Graph": [],
        "Algorithm": [],
        "Randomized": [],
        "Number of Communities": [],
        "Modularity": []
    }

    for algorithm in ["greedy", "label_propagation"]:
        # Detect communities in the original graph
        communities_original, modularity_original = detect_communities(graph,␣
↪algorithm=algorithm)
        data["Graph"].append(graph_name)
        data["Algorithm"].append(algorithm)
        data["Randomized"].append("No")
        data["Number of Communities"].append(len(communities_original))
        data["Modularity"].append(modularity_original)

        # Randomize the graph and repeat community detection
        randomized_graph = randomize_graph(graph)
        communities_randomized, modularity_randomized =␣
↪detect_communities(randomized_graph, algorithm=algorithm)
        data["Graph"].append(graph_name)
        data["Algorithm"].append(algorithm)
        data["Randomized"].append("Yes")
        data["Number of Communities"].append(len(communities_randomized))
        data["Modularity"].append(modularity_randomized)

    return pd.DataFrame(data)
```

[54]:

```
for graph, graph_name in zip(graphs, titels):
    df = collect_community_data(graph, graph_name)
    print("Community Detection Results for " + graph_name)
    print(df)
    print()
```

```
Community Detection Results for korea
   Graph          Algorithm Randomized  Number of Communities  Modularity
0  korea             greedy         No                      5    0.447066
1  korea             greedy        Yes                      6    0.276148
2  korea  label_propagation         No                      5    0.260488
3  korea  label_propagation        Yes                      1    0.000000

Community Detection Results for dolphins
      Graph          Algorithm Randomized  Number of Communities    Modularity
0  dolphins             greedy         No                      6  4.102864e-01
1  dolphins             greedy        Yes                      6  2.207743e-01
2  dolphins  label_propagation         No                      5  2.872530e-01
3  dolphins  label_propagation        Yes                      1  1.110223e-16
```

```
Community Detection Results for karate
    Graph          Algorithm Randomized  Number of Communities  Modularity
0  karate             greedy         No                      4    0.495491
1  karate             greedy        Yes                      6    0.339286
2  karate  label_propagation         No                      6    0.498576
3  karate  label_propagation        Yes                      6    0.280369

Community Detection Results for madrid
    Graph          Algorithm Randomized  Number of Communities  Modularity
0  madrid             greedy         No                      3    0.380671
1  madrid             greedy        Yes                      4    0.323389
2  madrid  label_propagation         No                      3    0.325115
3  madrid  label_propagation        Yes                      2    0.024326

Community Detection Results for starwars
      Graph          Algorithm Randomized  Number of Communities  Modularity
0  starwars             greedy         No                      3    0.287083
1  starwars             greedy        Yes                      5    0.190972
2  starwars  label_propagation         No                      2    0.086111
3  starwars  label_propagation        Yes                      1    0.000000
```