Exercise: Community Detection in the 117th United States Congress Social Network

Overview of the Dataset

This exercise is based on a social network dataset capturing interactions between members of the 117th United States Congress from February 9, 2022, to June 9, 2022. The network is a directed, weighted graph where edge weights represent empirically calculated probabilities of influence between Congress members. The dataset includes an edge list defining connections and their weights and a JSON file containing metadata, such as usernames.

Part 1: Data Import and Initial Network Visualization

Objective: Import the dataset, construct the network, and visualize its structure.

1. **Importing the Network:** Use NetworkX to load the directed, weighted edge list with:

   ```
   G = nx.read_weighted_edgelist("formatted_congress.edgelist",
   create_using=nx.DiGraph(), nodetype=int)
   ```
   Additionally, load the usernames from the JSON file:
   ```
   with open("congress_network_data.json", 'r') as f:
         data = json.load(f)
   usernames = data[0]['usernameList']
   ```

1. **Visualizing the Network:**
   - Use the `nx.spring_layout` function to arrange nodes for clarity.
   - Visualize the graph with `nx.draw`, and scale node sizes by their out-strength (sum of outgoing edge weights) using `G.out_edges(node, data=True)`.
2. **Key Functions:**
   - `nx.spring_layout(G)`
   - `nx.draw()`
3. **Enhanced Visualization:**
   - Scale edge widths by their weights.
   - Label key nodes (e.g., those with the highest out-strength) using `nx.draw_networkx_labels()`.

Part 2: Community Detection Using Greedy Modularity Maximization

Objective: Identify communities within the directed network using the greedy modularity algorithm.

1. **Detecting Communities:**

    - Use the `greedy_modularity_communities` function from NetworkX:

```
from networkx.algorithms.community import
greedy_modularity_communities
communities = list(greedy_modularity_communities(G))
```

2. **Computing Modularity:**
    - Evaluate the modularity score of the detected communities using:

```
modularity_score =
nx.algorithms.community.quality.modularity(G, communities)
```

    - The `modularity` function takes the graph and the list of communities as input and outputs a score measuring the quality of the partition (higher values indicate better separation of communities).
3. **Output Format:**
    - The result of the community detection is a list of sets, where each set contains the nodes in a community.
    - The modularity function returns a single numerical score.
4. **Visualizing Communities:**
    - Assign a unique color to each community and plot the network using the `node_color` parameter in `nx.draw()`.
5. **Key Functions:**
    - `greedy_modularity_communities(G)`
    - `nx.algorithms.community.quality.modularity(G, communities)`

Part 3: Community Detection Using the Louvain Algorithm

Objective: Identify communities in the undirected version of the network using the Louvain algorithm.

1. **Prepare the Network:** Convert the network to an undirected graph, as Louvain supports only undirected graphs:
`G_undirected = G.to_undirected()`
2. **Using the Louvain Algorithm:**
    - Use the `community` package

```
import community as community_louvain
partition =
community_louvain.best_partition(G_undirected,
weight="weight")
```

3. **Computing Modularity:**

- Convert the partition dictionary to a list of communities and calculate the modularity score:

```
    communities = [[] for _ in range(max(partition.values())
+ 1)]
    for node, comm in partition.items():
        communities[comm].append(node)
    modularity_score =
nx.algorithms.community.quality.modularity(G_undirected,
communities)
```

- Explain that modularity measures the quality of the partition by evaluating how well the network is divided into communities.

4. **Output Format:**
   - The Louvain algorithm's output (`partition`) is a dictionary where keys are node IDs and values are community IDs.
   - The modularity function outputs a numerical score.
5. **Visualizing Louvain Communities:**
   - Map communities to colors and use `nx.draw()` with the `node_color` parameter.
6. **Key Functions:**
   - `G.to_undirected()`
   - `community_louvain.best_partition(G_undirected, weight="weight")`
   - `nx.algorithms.community.quality.modularity(G_undirected, communities)`

Part 4: Stability of the Louvain Algorithm

Objective: Assess the stability of community assignments across multiple runs of the Louvain algorithm.

1. **Run Multiple Trials:**
   - Execute the Louvain algorithm several times with different random seeds.
   - Align community labels across runs using a label alignment function.
2. **Fluctuation Analysis:**
   - For each node, compute the proportion of times it is assigned to different communities.
   - Higher fluctuation indicates less stability.
3. **Results:**
   - Create tables showing the top 10 most and least fluctuating nodes, including their usernames and fluctuation percentages.
4. **Key Insight:**
   - The Louvain algorithm's randomness can lead to variability in community assignments.

5. **Key Functions:**
   - `community_louvain.best_partition(G_undirected, weight="weight", random_state=seed)`
   - `linear_sum_assignment` from `scipy.optimize` for label alignment.

Summary

This exercise guides you through importing, visualizing, and analyzing the dataset, as well as detecting communities and evaluating their stability. It introduces relevant NetworkX functions and external libraries, helping you develop a deeper understanding of network structure and community detection algorithms.