Group Assignment 1

Dat Nguyen
Ivan Revilla
Kelly Tsai

Watts-Strogatz Algorithm (Small-World Network Model)

- Purpose: simulates small-world properties
  - Will generate networks that balances randomness and regularity
- Algorithm:
  - Begins with a lattice. Each node in the lattice will be connected to k nearest neighbors
  - Steps:
    - Starts off with a ring lattice with N nodes connected to the k neighbors on either sides
      - Ensures high clustering and regularity
    - Rewire the edges randomly to a new node with probability (p)
- This algorithm should ensure that nodes will be reachable in only a few steps
- Has high clustering coefficient since the neighbors of a node are likely to be connected
  - Ex: friends of friends might know each other

Barabasi-Albert Algorithm

- Purpose: Generates a network with a “scale-free” structure
  - Some nodes/hubs might have significantly more connections than the others
- Algorithm:
  - Start with a small fully connected network of nodes
  - When a new node is added, connect it to the number of existing nodes in the network
  - Nodes with more connections will be more likely to get additional connections
  - Repeat until the desired size is reached
- This will simulate a power-law degree distribution
  - Few nodes have many connections and most have few
- This is stable against random failures
  - Vulnerable to targeted attacks on the hub

Description of Implementation

- The logging setup uses a class to log the output the console and file under test_{dataset_name}.log
- Load the dataset network data using the DataLoader and build the NetworkX Graph
    - We focus on the largest component so that all the nodes are reachable.
- The original function will print the analysis we are looking at (ex: size, average degree, clustering coefficient, etc.) using the implementation from scratch
- Use the Watts-Strogatz model (watts function) to output the same analysis as the original function but this time using the Watts-Strogatz algorithm
- Use the Barabasi-Albert model(barabasi function) to output the same analysis as the original function but this time using the Barabasi-Albert algorithm
- We then sequentially call all 3 functions (BA model, Watt-Strogata model, original model) and log the results to the console and log file to compare metrics of these 3 models.

Pseudocode for Watts Strogatz

```
function watts_strogatz_graph(n, k, p=0.1):
    create an empty graph G
    create a list of nodes from 0 to n-1

    # Create a regular ring lattice
    for each integer j from 1 to range(1, k//2):                    (if k even is k neighbors or if k odd → k-1 neighbors)
        Set target as rotated arrays [j,j+1,j+2,...,n-1,0,1,...,j-1]
        Then append zip( [0,1,2,...,n-1], [j,j+1,j+2,...,n-1,0,1,...,j-1] ) to the graph
              (by this way we have 0-1, ..., 0-k//2 and in this we also have 0-(n-1), 0-(n-2),...0-(n-k//2)) → which connect 0 with k neighbors

    # Rewire edges with probability p
    for each pair of nodes (u, v) in the Lattice Graph:
        if a random number is less than p:
            randomly choose a node w
            while w is the same as u or an edge (u, w) already exists in G:
                randomly choose a different node w
            if node u has reached its maximum possible edges:
                break out of the loop

            if valid new node w was found:
                remove the edge (u, v)
                add a new edge (u, w) to G

    return the graph
```

Implementation for Watts Strogatz

```python
def watts_strogatz_graph(self, n, k, p=0.1):
    G = nx.Graph()
    nodes = list(range(n))  # Nodes are labeled 0 to n-1

    # Step 1: Create a regular ring lattice
    for j in range(1, k // 2 + 1):  # k is odd then k-1
        # Targets are the k nearest neighbors
        targets = nodes[j:] + nodes[:j]
        G.add_edges_from(zip(nodes, targets))

    # Step 2: Rewire edges with probability p
    for j in range(1, k // 2 + 1):
        targets = nodes[j:] + nodes[:j]
        for u, v in zip(nodes, targets):
            if random.random() < p:  # With probability p, rewire the edge
                w = random.choice(nodes)
                # Avoid self-loops or duplicate edges
                while w == u or G.has_edge(u, w):
                    w = random.choice(nodes)
                    if G.degree(u) >= n - 1:
                        break  # Skip if the node has all possible edges
                else:
                    # Rewire the edge (u, v) to (u, w)
                    G.remove_edge(u, v)
                    G.add_edge(u, w)

    return G
```

Pseudocode for Barabasi Albert

```
function barabasi_albert_graph(n, m):
    # Initialize the graph
    create an initial star graph G with m+1 nodes (we use star graph since it will create a single central node with m edges connecting to other m nodes)

    # Create a list for preferential attachment
    initialize a list of node (repeated_nodes) with each node repeated according to its degree. (if node 1 has degree of 3 so it would appear in this array 3
times. The reason is to increase the probability of wiring the new node added with these high-degree nodes
    same with real-world, new user always follow celebrity when first creating account)

    # Add nodes with preferential attachment
    set source = 1 as the next node to be added to the current graph.
    while source is less than n:
        initialize an empty set targets
        while targets has fewer than m unique nodes:
            randomly select a node from repeated_nodes
            add the selected node to targets

        # Add edges from the new node to selected target nodes
        add edges between source and each node in targets in graph G

        # Update repeated_nodes list to reflect new degree (new edge connect to the node which cause increasing degree of the node)
        for each node in targets:
            add it once to repeated_nodes
        add source to repeated_nodes m times

        increment source by 1 (go to add the new node and continue the loop)

    return graph
```

Implementation for Barabasi Albert

```python
def barabasi_albert_graph(self, n, m):
    # Step 1: Initialize the graph with a small initial star graph of m+1 nodes
    G = nx.star_graph(m)

    # Step 2: Create a list of nodes with repetitions based on degree
    repeated_nodes = [node for node, degree in G.degree() for _ in range(degree)]

    # Step 3: Add remaining nodes with preferential attachment
    source = len(G)
    while source < n:
        # Choose m unique nodes from repeated_nodes list based on degree
        targets = set()
        while len(targets) < m:
            selected = random.choice(repeated_nodes)
            targets.add(selected)

        # Add new node `source` and connect it to `m` target nodes
        G.add_edges_from((source, target) for target in targets)

        # Update the repeated nodes list for preferential attachment
        repeated_nodes.extend(targets)
        repeated_nodes.extend([source] * m)

        source += 1  # Move to the next node to add

    return G
```

Average Shortest Path Length Calculation

- **Problem**: Direct average shortest path calculation is infeasible for large networks.
- **Subsampling Method**:
  - Sample 10% and 20% of nodes to create manageable subgraphs.
  - Extract the largest connected component for shortest path calculation.
  - Result varies significantly with each sample, limiting accuracy.
  - Problem:
    i. Results fluctuate across different samples.
    ii. Inconsistent estimates limit robustness for representing the full network.
- **Solution - Random Walk Approach**:
  - Conduct multiple random walks to estimate path lengths.
  - Optionally weight node selection by degree centrality for better coverage.
  - Produces a stable, representative estimate of the network's path length.

Amazon Analysis

Metric	Original Network	Barabasi-Albert	Watts-Strogatz
Size	334,863	334,863	334,863
Average Degree	5.529855493141971	4	4
Average Path Length (10% Nodes)	4.048633879781421	5.090410790549052	2.9111
Average Path Length (20% Nodes)	15.510698093574641	11.857689227324226	4.614035087719298
Average Path Length (Random Walk)	8.0306	6.0676	14.6178
Clustering Coefficient	0.3967463932787655	0.0002792626507916	0.37244455906122276

Twitch Analysis

Metric	Original Network	Barabasi-Albert	Watts-Strogatz
Size	168,114	168,114	168,114
Average Degree	80.86842261798542	80	80
Average Path Length (10% Nodes)	3.844158964486854	4.220375010480189	7.578791107000047
Average Path Length (20% Nodes)	3.3423478583201973	3.603310185546865	5.486381718559269
Average Path Length (Random Walk)	2.6008	2.8442	3.7027
Clustering Coefficient	0.15992499330884682	0.0032898140766689653	0.5407262238058127

How does the network models simulate real world?

- **Original Network:**
  - Original network metrics can represent an actual network of interactions (ex: user interactions via Amazon website, and Twitch gamer)
  - High clustering means that the nodes in the network are highly interconnected (ex: groups of people that share mutual connections)
  - The longer average shortest path lengths is, the more time it would take to traverse across different communities or groups
    - Sparsely connected networks
  - Some real-world examples can include:
    - Online marketplaces
    - Target ads based on a small group
    - Areas where clustering is high within the communities but paths are longer between the nodes

How does the network models simulate real world? (cont'd)

- **Watts-Strogatz Model:**
  - "Small-world" effect
    - Most nodes will be reachable within a few steps
    - Local neighborhoods are highly clustered
  - High clustering indicates that the nodes are densely interconnected
    - Ex: close or tightly-knit communities
  - The Watts-Strogatz model closely matches original metrics when tested with the Amazon dataset based on our experiment result. In this kind of applications, usually they can group relevant items together (Amazon products), as shown by the high clustering coefficient.
  - However, for applications like Twitch or other social media platforms, this kind of network model does not simulate the structure well. The clustering coefficient of the Watts-Strogatz model is too low to effectively capture the highly interconnected nature of these networks.

How does the network models simulate real world? (cont'd)

- **Barabasi-Albert Model:**
  - Connections in this model are based on preferential attachment
    - Nodes that have more connections are more likely to gain new ones which in return creates hubs
  - Because of the hubs, this model has a shorter path length and can reach nodes more quickly even in very large networks
  - Low clustering in the Barabasi-Albert (BA) model means that connections are often more global.
  - In contrast to Watts-Strogatz model: In our experiments, the BA model performed significantly better on the Twitch dataset. It more accurately approximated the average shortest path length and clustering coefficient of the original network.
  - Reasoning: The Twitch dataset has many "celebrity" nodes (high-degree nodes), and the BA model simulates network growth influenced by these high-degree nodes, making it a good fit for this kind of data.

Summary

- Watts-Strogatz:
  - Lowest average path length
  - High clustering coefficient
  - Best for networks where we need to consider local and global connections
- Barabasi-Albert Model:
  - Moderate average path lengths
  - Very low clustering coefficient
    - Basically no clustering among neighboring nodes
  - Best for networks without strong and stable community structures
- The Watts-Strogatz graph is better suited for creating cliques or tightly grouped nodes/groups
- The Barabasi-Albert Model is better suited for social networking as it models a power-law distribution
- Random Walk Algorithm is better for calculating average shortest path length of large network. The error is not very high but still can represent the network better than calculating average shortest path length of subsample of the original graph.
- In terms of the datasets used. Watts-Strogatz is a better fit for the Amazon dataset since you can group items of relevance together. The Barabasi-Albert Model for the Twitch dataset is better suited to see the social connections between streamers.

Tasks

Dat Nguyen	Implemented algorithms from scratch. Focused on source code optimization and implementation on HPC. Wrote slide and analyzed data.
Ivan Revilla	Implemented algorithms and presentation slides. Organized and wrote presentation slides
Kelly Tsai	Constructed graph and inputted data for presentation slides and report. Organized and wrote presentation slides. Analyzed data