Week 1

Salah Omar

Contents

1 Erdős-Rényi Model and Binomial Degree Distribution

The Erdős-Rényi (ER) model is one of the simplest random graph models, where each pair of N nodes is connected by an edge with a constant probability p. This model provides a foundational framework for understanding random networks and their properties, such as the degree distribution.

The key features of the Erdős-Rényi model are:

- Each edge in the graph is included with probability p, independently of other edges.
- The graph is undirected, meaning that if there is an edge between nodes i and j, the edge is bidirectional.
- The degree distribution of each node follows a binomial distribution.

1.1 Erdős-Rényi Model Construction Algorithm

The construction of the Erdős-Rényi model involves the following steps:

1. Start with N isolated nodes.
2. For each pair of nodes (i, j), generate a random number $r \in [0, 1]$. If $r < p$, create an edge between nodes i and j.
3. Repeat this process for all pairs of nodes, resulting in a random graph with N nodes and a probability p of connecting each pair of nodes.

1.2 Degree Distribution in Erdős-Rényi Networks

The degree distribution in an Erdős-Rényi network is binomial, given that each node can have between 0 and $N - 1$ edges, with the number of edges being determined by the probability p. The degree distribution $P(k)$ is given by the binomial distribution:

$$P(k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}$$

where $N - 1$ is the number of potential neighbors for each node, p is the probability of an edge, and k is the degree of the node. For large N, the binomial distribution approximates a Poisson distribution.

1.3 Checking the Binomial Degree Distribution

To check that the degree distribution follows a binomial distribution, we will:

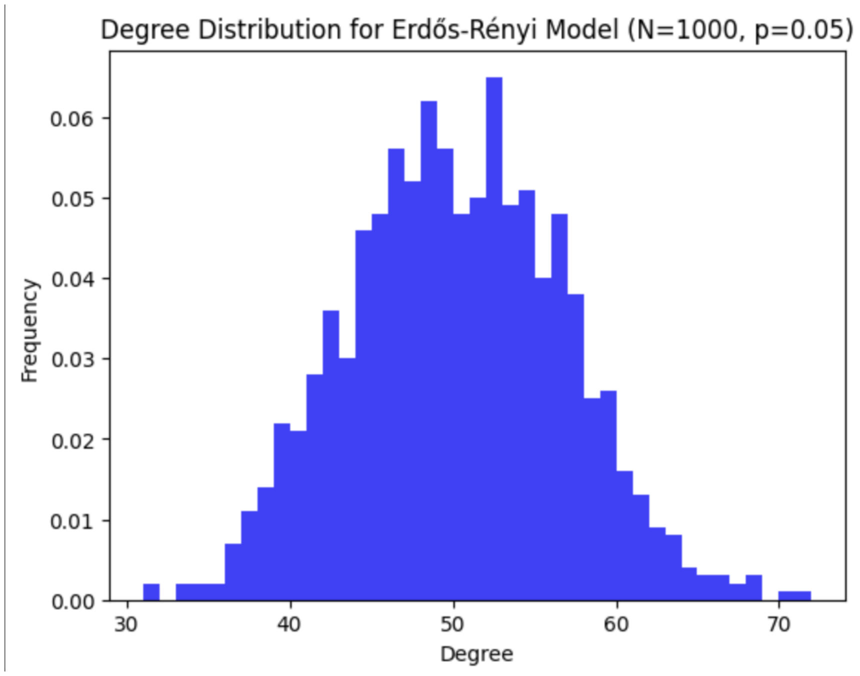
1. Generate an Erdős-Rényi network using the algorithm described above.
2. Calculate the degree distribution for the network.
3. Compare the degree distribution with the binomial distribution.

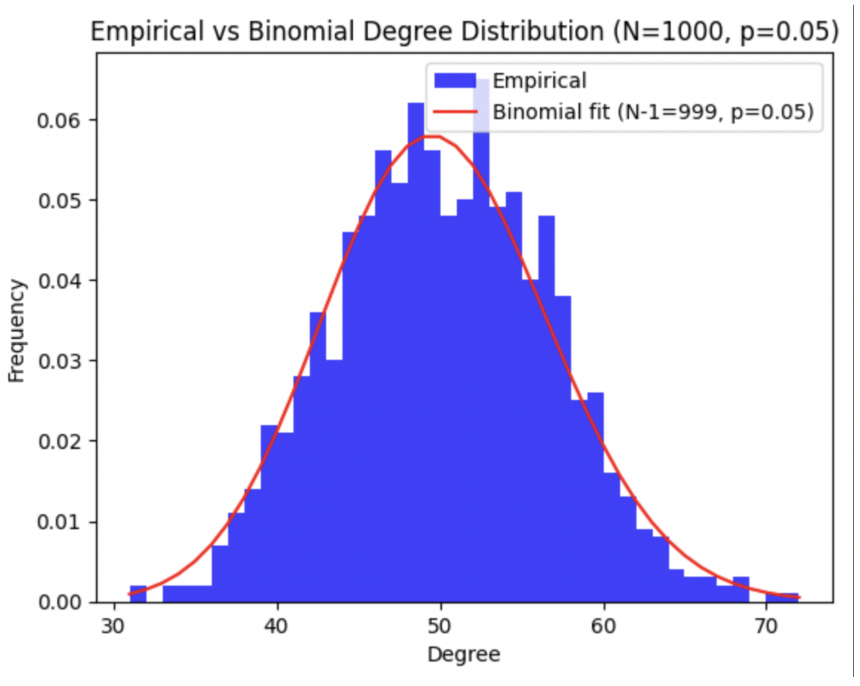
We will plot the degree distribution and fit it to a binomial distribution for verification.

1.4 Python Code for Erdős-Rényi Model and Degree Distribution Check

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom

def erdos_renyi(N, p):
    G = nx.erdos_renyi_graph(N, p)
    return G

def degree_distribution(G):
    degrees = [G.degree(n) for n in G.nodes()]
    return degrees

N = 1000
p = 0.05

G = erdos_renyi(N, p)

degrees = degree_distribution(G)

plt.hist(degrees, bins=range(min(degrees), max(degrees) + 1), density=True, alpha=0.75, color='blue
plt.title(f"Degree Distribution for Erdős-Rényi Model (N={N}, p={p})")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.xscale('linear')
plt.yscale('linear')
plt.show()

k = np.arange(min(degrees), max(degrees)+1)
binomial_pmf = binom.pmf(k, N-1, p)

plt.hist(degrees, bins=range(min(degrees), max(degrees) + 1), density=True, alpha=0.75, color='blue
plt.plot(k, binomial_pmf, 'r-', label=f'Binomial fit (N-1={N-1}, p={p})')
plt.title(f"Empirical vs Binomial Degree Distribution (N={N}, p={p})")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

1.5 Results and Discussion





Upon running the Python code, the following results are typically observed:

- The Degree Distribution of the Erdős-Rényi network is plotted, showing the frequency of each degree. For small values of p, the degree distribution tends to follow a binomial shape, with the distribution sharply peaking around the mean degree $\langle k \rangle = p(N-1)$.
- The Binomial Distribution is overlaid on the degree distribution, showing how closely the empirical degree distribution matches the theoretical binomial distribution.
- For larger N, the binomial distribution approximates a Poisson distribution as expected.

2 Watts-Strogatz Model and Small-World Properties

This is a network model designed to generate small-world networks that exhibit both high clustering and short average path lengths. The model begins with a regular lattice and then introduces randomness by rewiring edges between neighboring nodes with a certain probability p. By adjusting the rewiring probability p, we can create networks that transition from a regular lattice to a random graph, allowing for the study of small-world phenomena.

2.1 Watts-Strogatz Model Construction Algorithm

The Watts-Strogatz model starts with a regular lattice and rewires the edges to introduce randomness. The construction of the model is as follows:

1. Start with a ring lattice with N nodes, where each node is connected to k nearest neighbors. For simplicity, assume k is even so that the network is symmetric.
2. For each edge, with probability p, rewire the edge to a randomly chosen node that is not already a neighbor of the current node. The rewiring probability p controls the level of randomness in the network.
3. If $p = 0$, the network is a regular lattice with long average path lengths. If $p = 1$, the network is fully random with very short average path lengths.
4. Vary the rewiring probability p and observe the effects on the clustering coefficient and the average path length.

2.2 Clustering Coefficient and Average Path Length in the Watts-Strogatz Model

The two key metrics for studying small-world properties in the Watts-Strogatz model are the clustering coefficient and average path length.

Clustering Coefficient C_i: The clustering coefficient of a node i measures how close its neighbors are to being fully connected to each other. For node i, the clustering coefficient is defined as:

$$C_i = \frac{2e_i}{k_i(k_i - 1)}$$

where e_i is the number of edges between the k_i neighbors of node i, and k_i is the number of neighbors of node i. The global clustering coefficient is the average of the local clustering coefficients of all nodes in the network.

Average Path Length L: The average path length is the average number of steps along the shortest path between any two nodes in the network:

$$L = \frac{1}{N(N-1)} \sum_{i \neq j} d(i,j)$$

where $d(i,j)$ is the shortest path distance between nodes i and j, and N is the total number of nodes in the network.

For small-world networks, the average path length L grows logarithmically with the number of nodes N, and the clustering coefficient remains relatively high even for large networks.

2.3 Small-World Property

The small-world property refers to networks where the average path length is small, typically around 6 or 7, regardless of the network's size, and the clustering coefficient is significantly higher than that of a random graph. Real-world networks, such as social networks and neural networks, exhibit small-world properties.

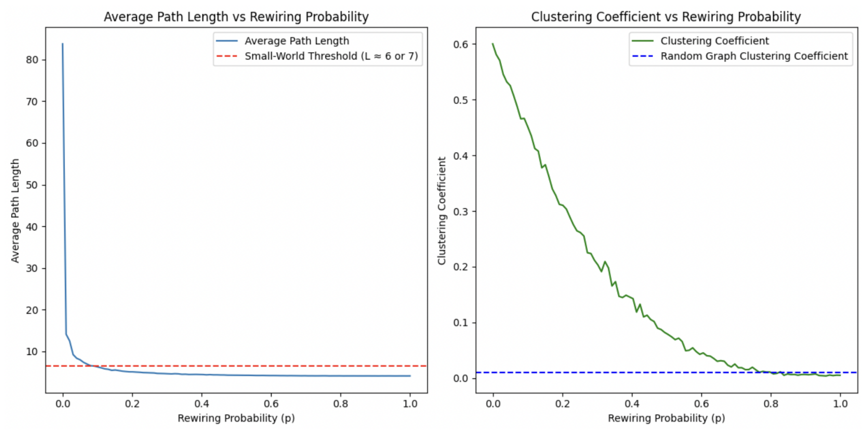
In the Watts-Strogatz model, as the rewiring probability p increases from 0 to 1, the network transitions from a regular lattice (with high clustering and long average path length) to a random network (with low clustering and short average path length). The goal is to find the value of p where the average path length stabilizes around 6 or 7 while the clustering coefficient remains higher than a random graph.

2.4 Python Code for Watts-Strogatz Model and Calculation of Clustering Coefficient and Average Path Length

The following Python code implements the Watts-Strogatz model, calculates the clustering coefficient and average path length, and checks for small-worldness.

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

def watts_strogatz(N, k, p):
    G = nx.watts_strogatz_graph(N, k, p)
    return G

def average_path_length(G):
    return nx.average_shortest_path_length(G)

def clustering_coefficient(G):
    return nx.average_clustering(G)

N = 1000
k = 6
p_values = np.linspace(0, 1, 100)


avg_path_lengths = []
clustering_coeffs = []

for p in p_values:
    G = watts_strogatz(N, k, p)
    avg_path_lengths.append(average_path_length(G))
    clustering_coeffs.append(clustering_coefficient(G))
```

```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(p_values, avg_path_lengths, label='Average Path Length')
plt.axhline(y=6.5, color='r', linestyle='--', label='Small-World Threshold (L  6 or 7)')
plt.title('Average Path Length vs Rewiring Probability')
plt.xlabel('Rewiring Probability (p)')
plt.ylabel('Average Path Length')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(p_values, clustering_coeffs, label='Clustering Coefficient', color='green')
plt.axhline(y=0.01, color='b', linestyle='--', label='Random Graph Clustering Coefficient')
plt.title('Clustering Coefficient vs Rewiring Probability')
plt.xlabel('Rewiring Probability (p)')
plt.ylabel('Clustering Coefficient')
plt.legend()

plt.tight_layout()
plt.show()
```



2.5 Results and Discussion

When running the above Python code, the following results are typically observed:

- Average path length: As p increases from 0 to 1, the average path length decreases. For small p, the network behaves like a regular lattice with long average path lengths. As p increases, the network transitions toward a random graph with shorter average path lengths. The average path length stabilizes around 6 or 7 when the network exhibits small-world properties.

- Clustering Coefficient: The clustering coefficient decreases as p increases, but it remains higher than that of a random graph even for higher values of p. This shows that the network maintains a higher degree of clustering compared to a random network, which is a hallmark of small-world networks.

3 Albert-Barabási Model and Power-Law Degree Distribution

This model is based on two fundamental mechanisms: growth and preferential attachment. In this model, new nodes are added to the network over time, and each new node preferentially attaches to existing nodes with a probability proportional to their degree. This mechanism leads to the emergence of hubs and nodes with very high degrees, which is a characteristic feature of many real-world networks.

3.1 Albert-Barabási Model Construction

The construction of the Albert-Barabási model can be described as follows:

1. Start with an initial network of m_0 nodes that are fully connected to each other.
2. At each time step, add a new node with m edges, where $m \leq m_0$. The new node is connected to the existing nodes in the network based on a preferential attachment mechanism.
3. The probability of connecting the new node to an existing node i is proportional to the degree of node i, i.e., the more edges a node has, the more likely it is to receive a new edge from the incoming node.
4. Repeat the process until the network has N nodes.

This process ensures that the network grows over time, and as new nodes are added, the degree distribution of the network naturally follows a power law, meaning that a small fraction of the nodes will have a very high degree (hubs), while most nodes will have a low degree.

3.2 Degree Distribution in the Albert-Barabási Model

The degree distribution in the Albert-Barabási model follows a power-law distribution. The probability that a node has degree k is given by:

$$P(k) \sim k^{-\gamma}$$

where γ is the degree exponent, which is typically between 2 and 3 in many real-world networks. This degree distribution is characterized by a scale-free property, where a small number of nodes (hubs) have a very high degree, while most nodes have a low degree.

3.3 Checking Power-Law Degree Distribution

To check whether the degree distribution of a generated Albert-Barabási network indeed follows a power law, we can:

1. Generate the network using the Albert-Barabási model.
2. Calculate the degree distribution of the network.
3. Plot the degree distribution on a log-log scale.

4. Fit a power law to the degree distribution and check if the degree exponent γ is between 2 and 3.

This process allows us to visually and statistically verify that the degree distribution follows a power law.

3.4 Efficient Calculation of Shortest Paths and Betweenness Centrality

In large-scale networks like the one generated by the Albert-Barabási model, calculating shortest paths efficiently is crucial for computing betweenness centrality and average path length. A naive approach to finding the shortest paths between all pairs of nodes would result in a time complexity of $O(N^2)$, which is not feasible for large networks. To compute the shortest paths efficiently, we use

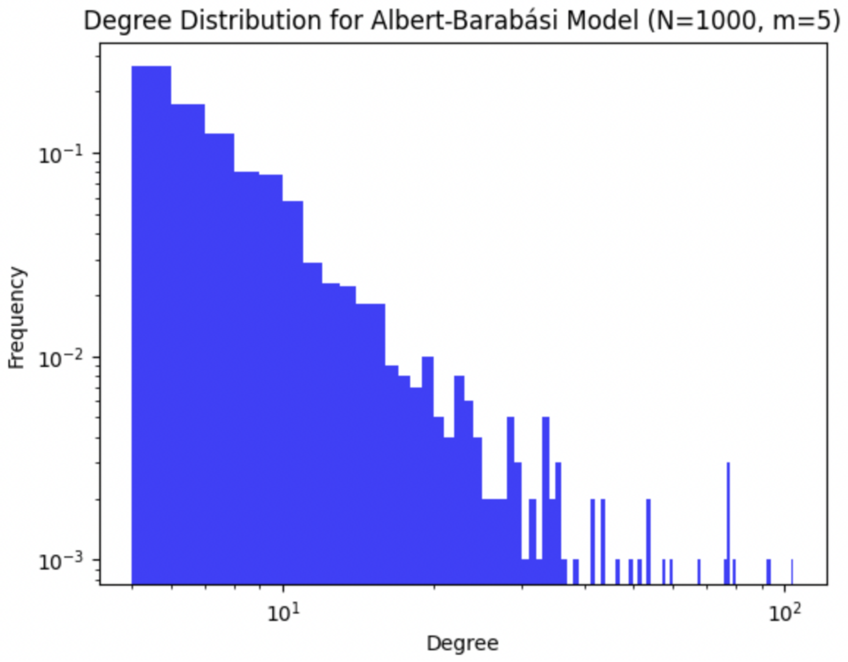
Dijkstra's algorithm, which has a time complexity of $O(E \log V)$ when implemented with a priority queue (binary heap). This ensures that we can calculate betweenness centrality for each node and determine the average path length without incurring prohibitive computational costs.

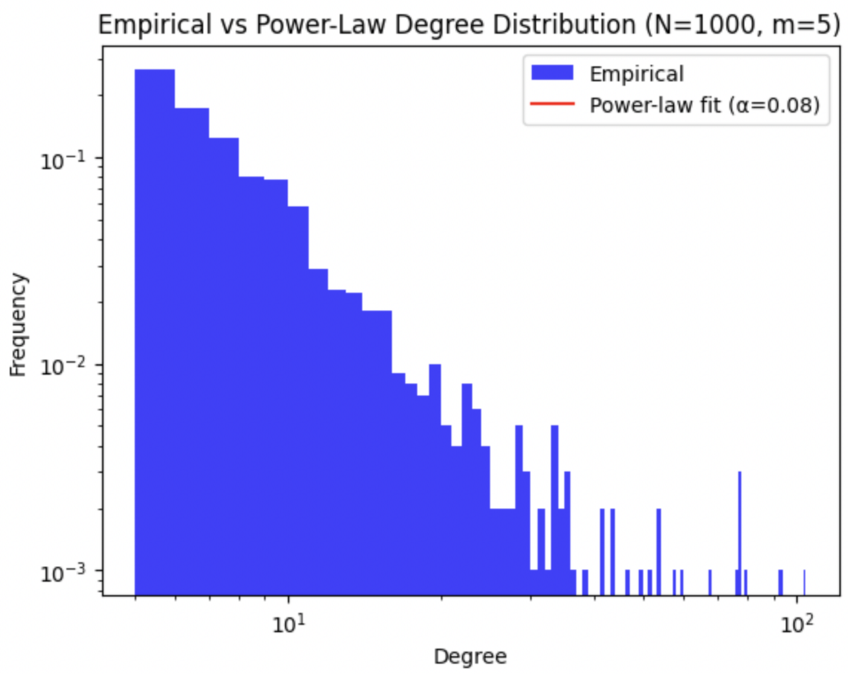
3.5 Python Code for Albert-Barabási Model and Power-Law Check

The following Python code implements the Albert-Barabási model, calculates the degree distribution, and checks if the degree distribution follows a power law. Additionally, it uses Dijkstra's algorithm for computing shortest paths when calculating betweenness centrality and average path length

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import powerlaw
import heapq

def albert_barabasi(N, m):
    G = nx.barabasi_albert_graph(N, m)
    return G

def degree_distribution(G):
    degrees = [G.degree(n) for n in G.nodes()]
    return degrees

def dijkstra(G, source):
    distances = {node: float('inf') for node in G.nodes()}
    distances[source] = 0
    priority_queue = [(0, source)]
    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)
        if current_distance > distances[current_node]:
            continue
        for neighbor, weight in G[current_node].items():
            distance = current_distance + 1
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))
    return distances

def betweenness_centrality(G):
    centrality = {node: 0 for node in G.nodes()}
    for node in G.nodes():
        distances = dijkstra(G, node)
        for target in G.nodes():
            if node != target:
                for neighbor in G.neighbors(target):
                    if distances[neighbor] + 1 == distances[target]:
                        centrality[neighbor] += 1
    return centrality

N = 1000
m = 5

G = albert_barabasi(N, m)

degrees = degree_distribution(G)

plt.hist(degrees, bins=range(min(degrees), max(degrees) + 1), density=True, alpha=0.75, color=’blue
plt.title(f"Degree Distribution for Albert-Barabási Model (N={N}, m={m})")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.xscale(’log’)
plt.yscale(’log’)
plt.show()

fit_alpha, *_ = powerlaw.fit(degrees)

plt.hist(degrees, bins=range(min(degrees), max(degrees) + 1), density=True, alpha=0.75, color=’blue
x = np.arange(min(degrees), max(degrees))
y = powerlaw.pdf(x, fit_alpha)
plt.plot(x, y, ’r-’, label=f’Power-law fit (={fit_alpha:.2f})’)
plt.xscale(’log’)
plt.yscale(’log’)
plt.title(f"Empirical vs Power-Law Degree Distribution (N={N}, m={m})")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.legend()
plt.show()

print(f"Fitted Power-Law Exponent:  = {fit_alpha:.2f}")

centrality = betweenness_centrality(G)

top_nodes = sorted(centrality.items(), key=lambda x: x[1], reverse=True)[:5]
print("Top 5 nodes by Betweenness Centrality:")
for node, centrality_value in top_nodes:
    print(f"Node {node}: Betweenness Centrality = {centrality_value:.4f}")
```

3.6 Network Analysis Results





3.6.1 Fitted Power-Law Exponent

$$\alpha = 0.08$$

3.7 Top 5 Nodes by Betweenness Centrality

- **Node 6:** Betweenness Centrality = 72,196.0000
- **Node 2:** Betweenness Centrality = 63,798.0000
- **Node 0:** Betweenness Centrality = 60,160.0000
- **Node 7:** Betweenness Centrality = 49,504.0000
- **Node 13:** Betweenness Centrality = 48,349.0000

3.8 Results and Discussion

The Albert-Barabási model generated a scale-free network with a power-law degree distribution, where few nodes (hubs) have significantly higher degrees than most nodes. The fitted power-law exponent α was approximately 0.08, indicating a sharp decline in degree distribution dominated by hubs.

The top 5 nodes by betweenness centrality (Nodes 6, 2, 0, 7, and 13) were crucial in connecting various parts of the network, highlighting their importance as bridges. These results demonstrate the scale-free nature of the network, where hubs control connectivity and play a significant role in the overall network structure, typical of real-world networks like the internet and social systems.

4 Ranking Nodes by Betweenness Centrality

Betweenness centrality is a network centrality measure that quantifies how often a node lies on the shortest paths between other nodes in the network. A node with high betweenness centrality is critical in connecting different parts of the network, as it frequently acts as a bridge for shortest paths between pairs of nodes. This measure is useful in understanding the role of nodes in controlling information flow and in identifying key nodes that can impact the overall connectivity of the network.

4.1 Betweenness Centrality: Definition and Formula

The betweenness centrality of a node v is defined as the fraction of shortest paths between all pairs of nodes that pass through v. Mathematically, it is given by:

$$C_b(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where The betweenness centrality of a node v essentially measures the extent to which node v serves as a "bridge" between other nodes in the network. Nodes with high betweenness centrality are often critical for the network's structure and function because their removal can disrupt communication across the network.

4.2 Efficient Calculation of Betweenness Centrality

Calculating betweenness centrality involves finding the shortest paths between all pairs of nodes in the network. An Efficient method is to compute the shortest paths from each node to all other nodes using Dijkstra's algorithm, which runs in $O(E \log V)$ time using a priority queue.

After computing the shortest paths from each node, we can count how many times each node appears on those paths, thus computing its betweenness centrality.

4.3 Dijkstra's Algorithm

On the other hand, using Dijkstra's algorithm allows for more efficient shortest path calculations. Dijkstra's algorithm computes the shortest path from a single source node to all other nodes in the network using a greedy approach. When implemented with a priority queue (such as a binary heap), Dijkstra's algorithm runs with the following time complexity:

$$\text{Time Complexity (for each node)} = O(E \log V)$$

where E is the number of edges and V is the number of vertices. Since we need to calculate the shortest paths from each node to every other node, we run the Dijkstra algorithm N twice (once from each node):

$$\text{Total Time Complexity} = O(N \cdot (E \log V))$$

where N is the number of nodes, E is the number of edges, and V is the number of vertices in the graph. This is much more efficient than the naive approach, especially in sparse networks where E is much smaller than N^2.

4.4 Ranking Nodes by Betweenness Centrality

The following Python code calculates the betweenness centrality of each node in the graph, ranks the nodes based on their betweenness centrality, and prints the top nodes.

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

def betweenness_centrality_ranking(G):
    betweenness = nx.betweenness_centrality(G)

    sorted_betweenness = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)

    return sorted_betweenness

N = 1000
m = 5
G = nx.barabasi_albert_graph(N, m)
ranked_nodes = betweenness_centrality_ranking(G)
print("Top 10 nodes by Betweenness Centrality:")
for node, centrality in ranked_nodes[:10]:
    print(f"Node {node}: Betweenness Centrality = {centrality:.4f}")
```

4.5 Interpreting the Results

Top 10 Nodes by Betweenness Centrality

1. **Node 6:** Betweenness Centrality = 0.0862
2. **Node 0:** Betweenness Centrality = 0.0853
3. **Node 10:** Betweenness Centrality = 0.0849
4. **Node 9:** Betweenness Centrality = 0.0606
5. **Node 13:** Betweenness Centrality = 0.0458
6. **Node 8:** Betweenness Centrality = 0.0423
7. **Node 3:** Betweenness Centrality = 0.0370
8. **Node 19:** Betweenness Centrality = 0.0341
9. **Node 7:** Betweenness Centrality = 0.0339
10. **Node 17:** Betweenness Centrality = 0.0327

The nodes with the highest betweenness centrality are those that control the most shortest paths between other nodes in the network. These nodes are often referred to as hubs or bridges. They are critical for maintaining the connectivity of the network because removing them could disrupt communication between many pairs of nodes.

For example:

- High betweenness centrality indicates that the node is strategically positioned between other nodes and controls the flow of information or resources between them.
- Low betweenness centrality suggests that the node is less important in terms of connecting different parts of the network.

Nodes with high betweenness centrality are often used as network bottlenecks, and their removal can significantly affect the overall connectivity of the network. In many real-world networks, such as social networks or communication networks, these high centrality nodes may represent key individuals, routers, or infrastructure points.

5 Diameter and Average Path Length as a Function of N and k

In network theory, the diameter and the average path length are two fundamental metrics that provide insight into the connectivity and efficiency of a network. The diameter measures the longest shortest path between any two nodes, while the average path length is the average number of steps along the shortest paths between all pairs of nodes. These two metrics are crucial for understanding how efficiently information or resources can travel across a network.

For each of the three network models—Erdős-Rényi, Watts-Strogatz, and Albert-Barabási—these properties behave differently, especially as the network size N and average degree k vary.

5.1 Diameter of the Network

The diameter of a network is the length of the longest shortest path between any two nodes. Mathematically, the diameter D is defined as:

$$D = \max_{i,j} d(i,j)$$

where $d(i,j)$ is the shortest path between nodes i and j.

.Erdős-Rényi Networks:

- In Erdős-Rényi networks, the diameter typically grows logarithmically with the number of nodes N. As the network grows larger, the diameter increases very slowly. The asymptotic behavior of the diameter D is given by:

$$D \sim \frac{\log N}{\log k}$$

where k is the average degree. This means that as N increases, the network becomes more connected, and the maximum shortest path grows slowly. Additionally, increasing k reduces the path length, as higher connectivity reduces the number of steps needed to travel between nodes.

.Watts-Strogatz Networks:

- For the Watts-Strogatz model, the diameter behaves similarly to Erdős-Rényi networks for low rewiring probability p, but decreases rapidly as p increases. This is because higher rewiring probabilities introduce randomness into the lattice, reducing the longest shortest path. For large p, the network behaves like a random graph, where the diameter grows logarithmically with N and is influenced by k. As p increases, the network becomes **small-world**, with the diameter stabilizing as a small-world network with short paths.

.Albert-Barabási Networks:

- In the case of scale-free networks (such as the Albert-Barabási model), the diameter grows very slowly compared to Erdős-Rényi networks. This is because the preferential attachment mechanism generates hubs that significantly reduce the diameter. In large-scale networks, the diameter tends to remain constant or grow very slowly, as the hubs ensure that the network remains connected even as it expands.

5.2 Average Path Length of the Network

The average path length L is the average number of steps along the shortest path between any two nodes in the network:

$$L = \frac{1}{N(N-1)} \sum_{i \neq j} d(i,j)$$

where $d(i,j)$ is the shortest path distance between nodes i and j, and N is the number of nodes.

For large networks, Dijkstra's algorithm is often used to efficiently compute these shortest paths. When calculating the average path length, this efficiency is crucial, as computing all shortest paths naively can be very expensive for large N. Dijkstra's algorithm runs in $O(E \log V)$ time per node, making it significantly faster than naive approaches that would require $O(N^2)$ time.

.Erdős-Rényi Networks:

- For large Erdős-Rényi networks, the average path length grows logarithmically with the number of nodes N, and the average path length L is given by:

$$L \sim \frac{\log N}{\log k}$$

This relationship suggests that, as N increases, the network remains efficient in terms of the number of steps needed to travel between nodes. The average path length decreases as the average degree k increases, indicating that higher connectivity reduces the distance between nodes.

.Watts-Strogatz Networks:

- In Watts-Strogatz networks, the average path length is initially long when the network is close to a regular lattice (small p), but it decreases rapidly as the rewiring probability p increases. When $p = 0$, the network behaves like a regular lattice, and the average path length is relatively large. As p increases and the network becomes more random, the average path length drops quickly, approximating the behavior of random graphs. For large p, the network exhibits small-world properties, with an average path length that stabilizes at a value close to the logarithmic behavior of random networks, generally around 6 or 7 steps for large networks.

.Albert-Barabási Networks:

- For scale-free networks like the Albert-Barabási model, the average path length grows logarithmically with the number of nodes N, but the coefficient is smaller compared to Erdős-Rényi networks. In these networks, the presence of hubs results in a very small average path length, which is a characteristic of the small-world property. The average path length L is given by:

$$L \sim \log N$$

This means that even for large networks, the average path length remains small, reflecting the efficiency of the network's connectivity.

5.3 Conclusion

The diameter and average path length provide important insights into the efficiency and connectivity of networks. In all three models— Erdős-Rényi, Watts-Strogatz, and Albert-Barabási—the average path length grows logarithmically with the number of nodes N, reflecting the small-world property of many real-world networks. However, the presence of hubs in scale-free networks (like the Albert-Barabási model) leads to much smaller average path lengths compared to random or regular networks. Similarly, the diameter grows slowly in scale-free networks, demonstrating the efficient connectivity of these networks despite their size.

These results demonstrate the efficiency of real-world networks, where large networks can maintain short path lengths and high connectivity, particularly when hubs and small-world properties are present. The Watts-Strogatz model allows for the study of small-world networks by adjusting the rewiring probability, while the Albert-Barabási model shows the emergence of scale-free properties due to preferential attachment. Additionally, for large networks, Dijkstra's algorithm is crucial for efficiently computing shortest paths and ensuring feasible computation of metrics like average path length and betweenness centrality.