Homework 2: Visualizing Networks

Maurice D. Hanisch

January 17, 2026

AI Declaration

All the first versions of the code have been written without AI. They were then debugged with AI. AI helped format this TeX file and help with the plots as well.

1. Working with real data

1. a) Network Statistics and Distributions

I analyzed the largest connected component of the researcher collaboration network ($n = 4158$). Its degree distribution and metrics are shown below:

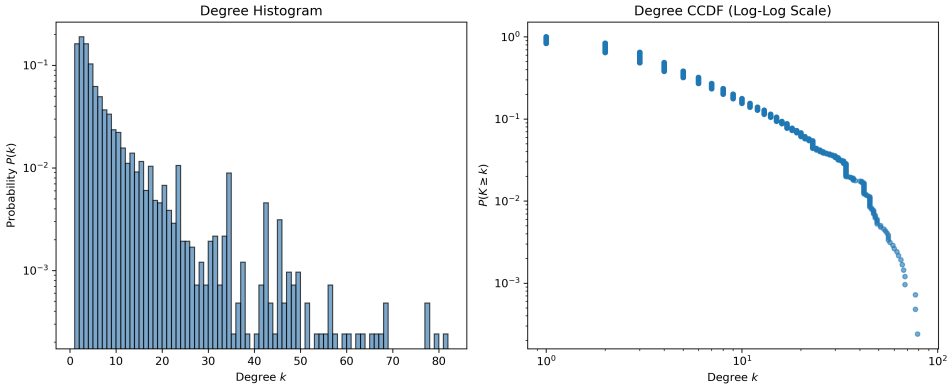


Figure 1: Degree Histogram (left) and Complementary Cumulative Distribution Function (right) for the co-authorship network.

The calculated metrics for the largest connected component are:

- **Average Clustering Coefficient:** 0.5569

- **Overall Clustering Coefficient:** 0.6289
- **Maximal Diameter:** 17
- **Average Diameter:** 6.0494

1. b) Triangles and Erdős–Rényi Parameter

The number of triangles T in the network is $47,779$.

To derive the expected number of triangles in an Erdős–Rényi graph $G(n,p)$, I consider all possible triplets of nodes. There are $\binom{n}{3}$ such triplets. For any triplet $\{u,v,w\}$ to form a triangle, the edges (u,v), (v,w), and (w,u) must all exist. Since edges in $G(n,p)$ occur independently with probability p, the probability that all three edges exist is $p \cdot p \cdot p = p^3$.

Let $X_{i,j,k}$ be an indicator random variable that is 1 if the nodes $\{i,j,k\}$ form a triangle and 0 otherwise. The total number of triangles is $T = \sum_{1 \le i < j < k \le n} X_{i,j,k}$. By the linearity of expectation:

$$\mathbb{E}[T] = \sum_{1 \le i < j < k \le n} \mathbb{E}[X_{i,j,k}] = \binom{n}{3} p^3$$

Setting $\mathbb{E}[T] = T = 47,779$ and $n = 4158$, we calculate p:

$$47,779 = \binom{4158}{3} p^3$$

$$\implies p \approx 0.015862$$

1. c) Erdős–Rényi Model Fit

The Erdős–Rényi model does not fit this graph well. In $G(n,p)$, node degrees follow a Binomial distribution $B(n-1,p)$, which looks like a Gaussian at this scale. The actual co-authorship network is much more skewed with a heavy tail. A few authors have many papers and many connections, while the ER model stays clustered near the mean.

I do not need the histogram to see that ER is a poor fit. Collaboration networks cluster because a paper with k authors forms a clique, creating many triangles. In a $G(n,p)$ graph, the clustering coefficient is much lower. The actual clustering coefficient here is 0.5569. The ER model cannot replicate this inherent structure.

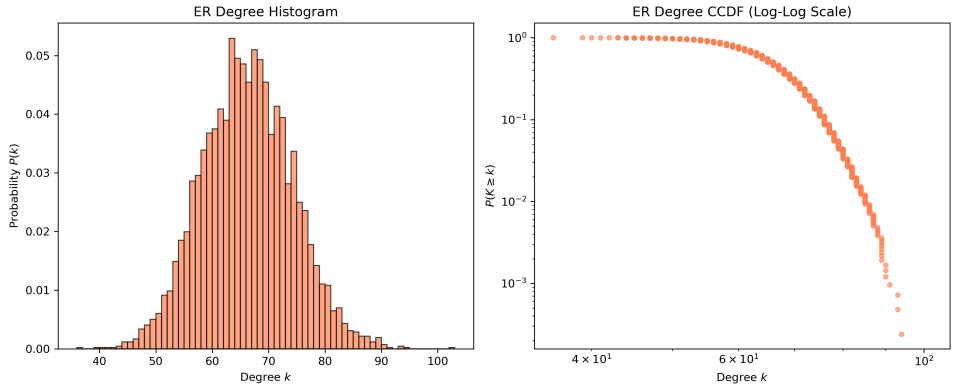


Figure 2: Degree Histogram (left) and CCDF (right) for a generated Erdős–Rényi graph ($n = 4158, p = 0.015862$).

2. Visualize your network

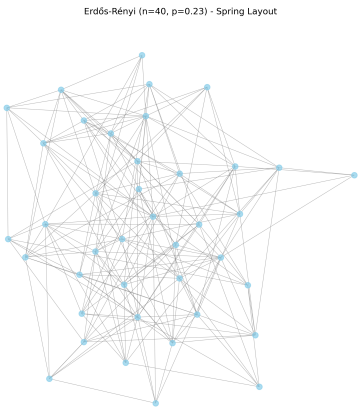
For the Erdős–Rényi and SSBM models, I sampled networks with the base parameters ($n = 40$ and $n = 30$), but I also included examples with $n = 100$ nodes (Figure 4 and Figure 6). With the larger n, the expected structure of the graph models becomes much more apparent than with the smaller examples—specifically the uniform density of the ER model and the distinct community separation in the SSBM.

2. a) Erdős-Rényi Visualization

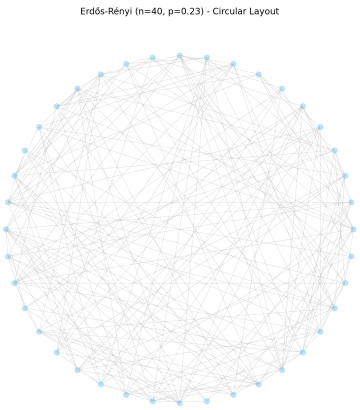
I compared a spring layout to a circular layout for the $n = 40$ ER graph. The spring layout (Figure 3a) provides a clear view of the connectivity, whereas the circular layout (Figure 3b) is less effective for recognizing local structure. For the $n = 100$ version (Figure 4), I balanced the alpha and line width to manage the high edge density.

2. b) SSBM Visualization

For the SSBM, I implemented a visualization that colors nodes by their cluster ID. I also differentiated edge colors: gray for intra-cluster edges and salmon for inter-cluster edges. The spring layout (Figure 5a) clearly separates the communities. I also included a random layout (Figure 5b) as a contrast; this layout is largely useless for analysis as it provides no information about the community structure.



(a) Spring Layout (n=40)



(b) Circular Layout (n=40)

Figure 3: Erdős-Rényi visualizations showing layout comparison.

2. c) The Web

I visualized the web graph for $n = 100$ (Figure 7a) and $n = 300$ (Figure 8). I also attempted a radial/shell layout (Figure 7b), but it proved ineffective as the directed edge structure became impossible to recognize. For the $n = 300$ graph, which is quite dense, I significantly reduced the alpha and edge width so that the underlying connections could still be discerned.

Code Implementation

The following code generates and visualizes these networks. It is imported directly from the notebook.

Code Listing: Visualization Generation

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

SEED = 42

def ER_G(n, p, seed=42):
    rng = np.random.RandomState(seed)
```

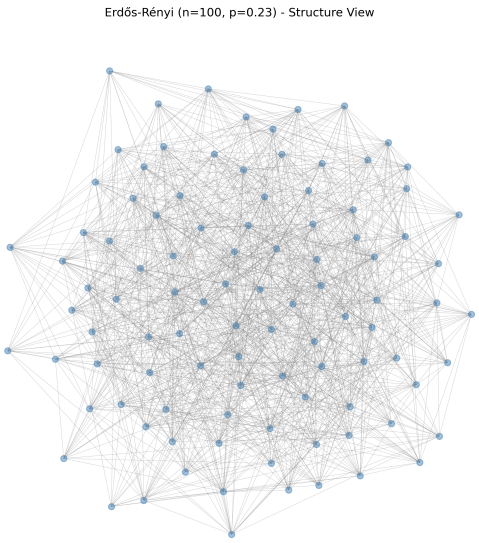
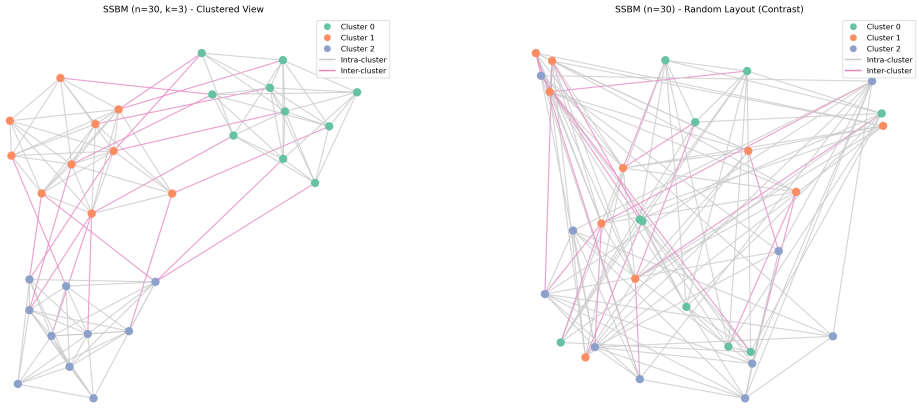


Figure 4: Erdős-Rényi with $n = 100$ showing dense structure.

```
    G = nx.Graph()
    G.add_nodes_from(range(n))
    for i in range(n):
        for j in range(i + 1, n):
            if rng.rand() < p:
                G.add_edge(i, j)
    return G

G_er_40 = ER_G(40, 0.23, SEED)

# Plot 1: Spring Layout (Effective)
plt.figure(figsize=(8, 8), dpi=300)
pos = nx.spring_layout(G_er_40, k=0.5, seed=SEED)
nx.draw(G_er_40, pos, node_size=60, node_color='skyblue', edge_color='gray', width=0.5, alpha=0.7)
plt.title("Erdős-Rényi (n=40, p=0.23) - Spring Layout")
os.makedirs('latex/figs', exist_ok=True)
plt.savefig('latex/figs/2a_ER_spring.png', bbox_inches='tight')
```



(a) Spring Layout (with clustering) (b) Random Layout

Figure 5: SSBM visualizations showing the effect of clustering.

```
plt.close()

# Plot 2: Circular Layout (Contrasting)
plt.figure(figsize=(8, 8), dpi=200)
pos_circ = nx.circular_layout(G_er_40)
nx.draw(G_er_40, pos_circ, node_size=50, node_color=
    'skyblue', edge_color='gray', width=0.3, alpha
    =0.5)
plt.title("Erdős-Rényi (n=40, p=0.23) - Circular
    Layout")
plt.savefig('latex/figs/2a_ER_circular_contrast.png'
    , bbox_inches='tight')
plt.close()

# ER with n=100 to show dense structure better
G_er_100 = ER_G(100, 0.23, SEED)

plt.figure(figsize=(10, 10), dpi=300)
pos = nx.spring_layout(G_er_100, k=0.8, seed=SEED)
# Balanced alpha and width for high density
nx.draw(G_er_100, pos, node_size=40, node_color='
    steelblue', edge_color='gray', width=0.2, alpha
    =0.3)
```

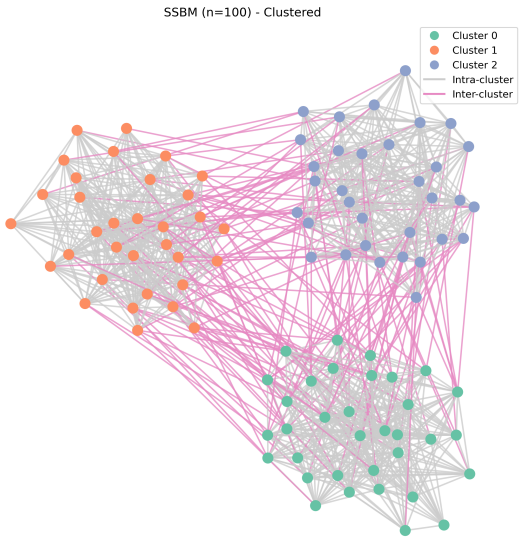


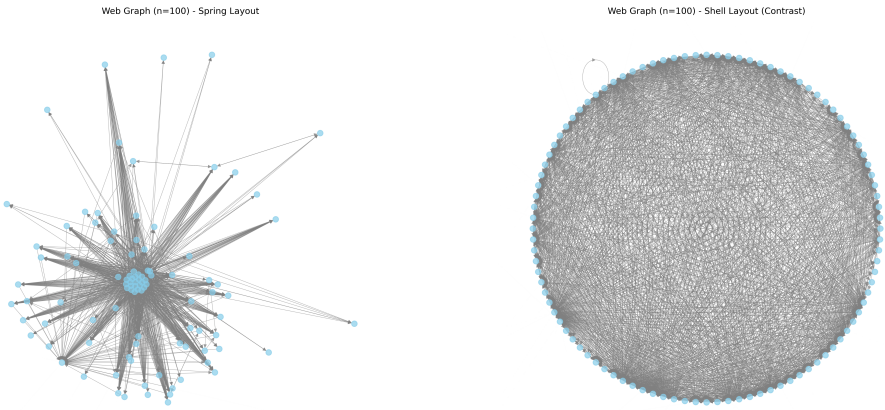
Figure 6: SSBM ($n = 100$) showing clear separation of 3 clusters.

```
plt.title("Erdős-Rényi (n=100, p=0.23) - Structure View")
plt.savefig('latex/figs/2a_ER_100.png', bbox_inches='tight')
plt.close()

import matplotlib.lines as mlines

def SSBM_G(n, k, A, B, seed=42):
    rng = np.random.RandomState(seed)
    G = nx.Graph()
    G.add_nodes_from(range(n))

    # Assign clusters
    nodes = np.array(range(n))
    rng.shuffle(nodes)
    cluster_size = n // k
    labels = np.zeros(n, dtype=int)
    for i in range(k):
        labels[nodes[i*cluster_size:(i+1)*cluster_size]] = i
```



(a) n=100 (Spring) (b) n=100 (Radial/Shell)

Figure 7: Web graph visualizations.

```
62
63      for i in range(n):
64          for j in range(i + 1, n):
65              prob = A if labels[i] == labels[j] else
    B
66              if rng.rand() < prob:
67                  G.add_edge(i, j)
68
69      return G, labels
70
71  def plot_colored_ssbm(G, labels, title, filename,
    layout='spring'):
72      plt.figure(figsize=(10, 10), dpi=300)
73
74      if layout == 'spring':
75          pos = nx.spring_layout(G, seed=SEED, k=0.5)
76      else:
77          pos = nx.random_layout(G, seed=SEED)
78
79      # Color nodes by cluster
80      colors = ['#66c2a5', '#fc8d62', '#8da0cb'] # Set
     2 colors
81      node_colors = [colors[labels[i]] for i in G.
    nodes()]
```

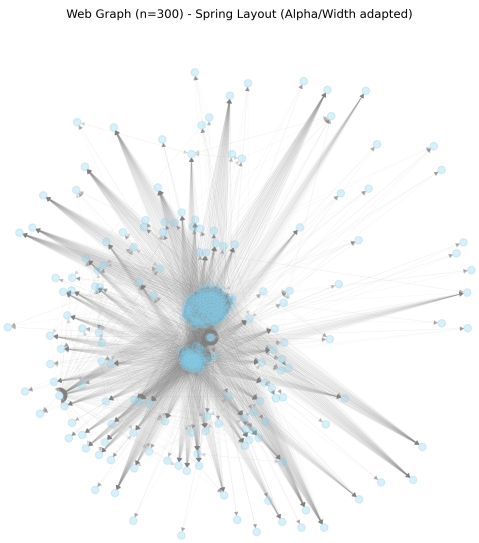


Figure 8: Web graph with $n = 300$ using balanced alpha and width.

```
82
83      # Color edges: light gray for intra-cluster,
        salmon for inter-cluster
84      edge_colors = []
85      for u, v in G.edges():
86          if labels[u] == labels[v]:
87              edge_colors.append('#cccccc') # Light
        gray intra
88          else:
89              edge_colors.append('#e78ac3') # Pinkish
        inter
90
91      nx.draw_networkx_nodes(G, pos, node_size=100,
        node_color=node_colors)
92      nx.draw_networkx_edges(G, pos, width=1.5,
        edge_color=edge_colors, alpha=0.8)
93
94      # Create legend
95      cluster_handles = [mlines.Line2D([], [], color='
        white', marker='o', markerfacecolor=c, markersize
        =10, label=f'Cluster {i}') for i, c in enumerate(
```

```
    colors)]
    edge_handles = [
        mlines.Line2D([], [], color='#cccccc',
    linewidth=2, label='Intra-cluster'),
        mlines.Line2D([], [], color='#e78ac3',
    linewidth=2, label='Inter-cluster')
    ]
    plt.legend(handles=cluster_handles +
    edge_handles, loc='upper right')

    plt.title(title)
    plt.axis('off')
    os.makedirs('latex/figs', exist_ok=True)
    plt.savefig(filename, bbox_inches='tight')
    plt.close()

# Generate and plot SSBM
G_ssbm_30, labels_30 = SSBM_G(30, 3, 0.8, 0.05, SEED
    )
plot_colored_ssbm(G_ssbm_30, labels_30, "SSBM (n=30,
     k=3) - Clustered View", "latex/figs/2
    b_SSBM_spring.png", layout='spring')

# Random layout for contrast - WITH edge coloring
plot_colored_ssbm(G_ssbm_30, labels_30, "SSBM (n=30)
     - Random Layout (Contrast)", "latex/figs/2
    b_SSBM_random_contrast.png", layout='random')

# Larger SSBM check
G_ssbm_100, labels_100 = SSBM_G(100, 3, 0.5, 0.05,
    SEED)
plot_colored_ssbm(G_ssbm_100, labels_100, "SSBM (n
    =100) - Clustered", "latex/figs/2b_SSBM_100.png",
     layout='spring')

# SSBM with n=100 to show clustering better
G_ssbm_100, labels_100 = SSBM_G(100, 3, 0.7, 0.1,
    SEED)
plot_colored_ssbm(G_ssbm_100, labels_100, "SSBM (n
    =100, k=3) - Structure View", "2b_SSBM_100")

import pickle

```

```
with open('data/caltech_web_graph.pkl', 'rb') as f:
    G_web = pickle.load(f)

# Use list to index nodes
nodes = list(G_web.nodes())
G_web_100 = G_web.subgraph(nodes[:100])
G_web_300 = G_web.subgraph(nodes[:300])

# 1. Spring Layout for n=100
plt.figure(figsize=(8, 8), dpi=300)
pos_spring = nx.spring_layout(G_web_100, k=0.5, seed
    =SEED)
nx.draw(G_web_100, pos_spring, node_size=60,
    node_color='skyblue', edge_color='gray', width
    =0.5, alpha=0.7)
plt.title("Web Graph (n=100) - Spring Layout")
os.makedirs('latex/figs', exist_ok=True)
plt.savefig('latex/figs/2c_Web100_spring.png',
    bbox_inches='tight')
plt.close()

# 2. Radial/Shell Layout for n=100 (Contrast)
plt.figure(figsize=(8, 8), dpi=300)
pos_shell = nx.shell_layout(G_web_100)
nx.draw(G_web_100, pos_shell, node_size=60,
    node_color='skyblue', edge_color='gray', width
    =0.5, alpha=0.7)
plt.title("Web Graph (n=100) - Shell Layout (
    Contrast)")
plt.savefig('latex/figs/2c_Web100_radial_contrast.
    png', bbox_inches='tight')
plt.close()

# Web 300 - Spring with balanced alpha/width
plt.figure(figsize=(8, 8), dpi=300)
pos = nx.spring_layout(G_web_300, k=0.4, seed=SEED)
# Very low alpha and thin lines for dense web graph
nx.draw(G_web_300, pos, node_size=60, node_color='
    skyblue', edge_color='gray', width=0.1, alpha
    =0.3)
plt.title("Web Graph (n=300) - Spring Layout (Alpha/
    Width adapted)")
```

```
plt.savefig('latex/figs/2c_Web300_spring.png',
    bbox_inches='tight')
plt.close()
```

3. The Navigation Paradox

3. a) & b) Graph Generation and Distance

I generated a connected Watts-Strogatz graph with $n = 1000$, $k = 10$, and $p = 0.1$. I implemented the ring distance function $d_{ring}(u, v) = \min(|u - v|, n - |u - v|)$.

3. c) Path Length Comparison

I compared the true shortest path length with the path length found by a greedy search for 100 random pairs.

- **Average Shortest Path Length:** 4.48
- **Average Greedy Path Length:** 12.99

The greedy search performs significantly worse than the optimal path, and in some cases, it can get stuck or take very long routes.

3. d) Why Greedy Search Fails

The greedy agent fails to utilize the random shortcuts effectively because it relies solely on local geometric information (the 1D ring distance) to make routing decisions.

In the Watts-Strogatz model, the "shortcuts" are added (or rewired) uniformly at random; they are not correlated with the underlying geometry of the ring. A greedy agent will only take a shortcut if it lands strictly closer to the target in terms of ring distance. This approach misses opportunities to take shortcuts that might initially appear to move away from or stay roughly equidistant to the target on the ring, but actually connect to a "highway" leading very close to the destination.

Code Implementation: Navigation

Code Listing: Navigation Simulation

```python
import numpy as np
import networkx as nx
from tqdm import tqdm

n = 1000
k = 10
p = 0.2
n_samples = 1000000

def d_ring(u, v, n=1000):
    dist = np.abs(u - v)
    return np.minimum(dist, n - dist)

def greedy_length(G, u, v):
    length = 0

    while u != v:
        if length >= n: # Prevent infinite loops
            # print(f"Warning: exceeded max length in greedy search from {u} to {v}")
            if (np.abs(u - v) != 1):
                print(f"Stuck at node {u} trying to reach {v}")
            return np.inf
        neighbors = np.array(list(nx.neighbors(G, u)))
        d_rings = d_ring(neighbors, v)
        u = neighbors[np.argmin(d_rings)]
        length += 1

    return length

G = nx.watts_strogatz_graph(n=n, k=k, p=p, seed=42)
print("is connected:", nx.is_connected(G))

uvs = np.random.choice(list(range(n)), (n_samples, 2), replace=True) # This doesn't exclude (u, u)

shortest_lens = []
```

```
greedy_lens = []
num_failed = 0
for i in tqdm(range(len(uvs))):
    shortest_lens.append(nx.shortest_path_length(G, uvs[i, 0], uvs[i, 1]))
    greedy_lens.append(greedy_length(G, uvs[i, 0], uvs[i, 1]))
    if greedy_lens[-1] == np.inf:
        num_failed += 1

print(f"Avg. shortest length: {np.mean(shortest_lens)}")
print(f"Avg. greedy length: {np.mean(greedy_lens)}")
print(f"Number of failed greedy searches: {num_failed} out of {n_samples} -> probability {num_failed / n_samples}")
```