CMS/CS/EE 144 - Homework 1 Analysis Report

AI Use Declaration

I manually implemented the initial version of the crawler. I used AI assistance to optimize the crawler by parallelizing it across 5 workers and to help visualize the network structure. For the analysis, I used AI to identify the correct NetworkX functions for calculating diameter and clustering coefficients, as well as to clean histograms and CCDFs. Finally, AI was used to help format this LaTeX report.

1.1.1 Code & Execution

Libraries: The crawler uses `networkx` for graph management, `matplotlib` for plotting, and `concurrent.futures` for parallel execution.

How to run:

1. Run the crawler: `python crawler.py`
   This generates the graph, saves it to `caltech_web_graph.pkl`, and creates the network visualization.

2. Run the analysis: `python analysis.py`
   This calculates metrics and generates the histograms/CCDFs.

```
from fetcher3 import fetch_links
import networkx as nx
import matplotlib.pyplot as plt
import concurrent.futures
import pickle

start_url = "https://www.caltech.edu"
num_nodes_limit = 2000
MAX_WORKERS = 5  # Polite parallelism (approx 5 concurrent requests)

G = nx.DiGraph()
G.add_node(start_url)

to_visit = [start_url]
visited = set()

with concurrent.futures.ThreadPoolExecutor(max_workers=MAX_WORKERS) as executor:
    while len(visited) < num_nodes_limit and to_visit:
```

```
        # 1. Select a batch of URLs to visit (BFS)
        batch = []
        while len(batch) < MAX_WORKERS and to_visit:
            url = to_visit.pop(0)
            if url not in visited:
                batch.append(url)
                visited.add(url)

        if not batch:
            break

        # 2. Parallel Fetch
        future_to_url = {executor.submit(fetch_links, url): url for url in batch}

        # 3. Process results as they complete
        for future in concurrent.futures.as_completed(future_to_url):
            current_url = future_to_url[future]
            try:
                links = future.result()
                if links:
                    for link in links:
                        if "caltech.edu" in link:
                            # Add edge (safe: main thread does this)
                            G.add_edge(current_url, link)
                            # Add to queue if not seen (approximate check, set handles exact)
                            if link not in visited:
                                to_visit.append(link)
            except Exception as e:
                pass # safely ignore fetch errors
            print(f"Crawled: {len(visited)} | Nodes: {G.number_of_nodes()} | Queue: {len(to_visit)}", end='\r')

unvisited_nodes = [node for node in G.nodes() if node not in visited]
G.remove_nodes_from(unvisited_nodes)

print(f"\nFinal Graph: {G.number_of_nodes()} nodes, {G.number_of_edges()} edges")

# Save the graph
with open('caltech_web_graph.pkl', 'wb') as f:
    pickle.dump(G, f)
print("Graph saved to caltech_web_graph.pkl")

# plot the graph
plt.figure(figsize=(10, 10))
nx.draw(G,
        pos=nx.spring_layout(G, k=0.15, iterations=20), # k controls spacing
        node_size=10,
        width=0.1,
        arrowsize=5,
        with_labels=False,
```

```
        node_color='blue',
        edge_color='gray',
        alpha=0.6)
plt.title("Caltech Web Crawl")
plt.axis('off')
plt.savefig('caltech_web_graph.png', dpi=500)
```

Listing 1: Crawler Implementation

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

import os

# Graph loading moved to analyze_dataset function

def plot_histogram(data, title, xlabel, ylabel, filename):
    plt.figure()
    plt.hist(data, bins=range(min(data), max(data) + 2, 5), align='left', rwidth=0.8)
    plt.yscale('log')
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.savefig(filename)
    plt.close()

def plot_ccdf(data, title, xlabel, ylabel, filename):
    # Filter out zeros for log-log plot (reciprocal of infinite is 0, log(0) is undef)
    # or just shift them if 0 is meaningful. For degrees, 0 is possible.
    # Usually we plot data > 0 for power law checks.
    data = np.array(data)
    data = data[data > 0]

    if len(data) == 0:
        print(f"Warning: No positive data for {title}")
        return

    data_sorted = np.sort(data)
    # Calculate CCDF: P(X >= x)
    # rank i (0-based) corresponds to N-i elements >= x
    # y = (N - i) / N
    n = len(data_sorted)
    y = np.arange(n, 0, -1) / n

    plt.figure()
    plt.loglog(data_sorted, y, marker='.', linestyle='none')
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.grid(True, which="both", ls="-", alpha=0.2)
    plt.savefig(filename)
    plt.close()

```

```python
def analyze_dataset(pickle_file, label, file_prefix=''):
    print(f"\n--- Analyzing {label} ({pickle_file}) ---")

    try:
        with open(pickle_file, 'rb') as f:
            G = pickle.load(f)
        print(f"Graph loaded with {G.number_of_nodes()} nodes and {G.number_of_edges()} edges.")
    except FileNotFoundError:
        print(f"File {pickle_file} not found. Skipping.")
        return

    # 1.3 Histograms & 1.4 CCDF
    out_degrees = [G.out_degree(n) for n in G.nodes()]
    in_degrees = [G.in_degree(n) for n in G.nodes()]

    prefix = file_prefix

    plot_histogram(out_degrees, f"Out-Degree Distribution ({label})", "Out-Degree", "Frequency", f"{prefix}hist_out_degree.png")
    plot_histogram(in_degrees, f"In-Degree Distribution ({label})", "In-Degree", "Frequency", f"{prefix}hist_in_degree.png")

    plot_ccdf(out_degrees, f"CCDF of Out-Degree ({label})", "Out-Degree", "P(X >= x)", f"{prefix}ccdf_out_degree.png")
    plot_ccdf(in_degrees, f"CCDF of In-Degree ({label})", "In-Degree", "P(X >= x)", f"{prefix}ccdf_in_degree.png")

    # 1.5 Clustering Coefficients - Treat as undirected
    G_undirected = G.to_undirected()

    # Handle disconnected components if necessary for diameter
    if not nx.is_connected(G_undirected):
        largest_cc = max(nx.connected_components(G_undirected), key=len)
        G_comp = G_undirected.subgraph(largest_cc)
        print(f"Graph is not connected. Using largest component ({len(largest_cc)} nodes) for diameter.")
    else:
        G_comp = G_undirected

    avg_clustering = nx.average_clustering(G_undirected)
    overall_clustering = nx.transitivity(G_undirected)

    print(f"Average Clustering Coefficient: {avg_clustering}")
    print(f"Overall Clustering Coefficient: {overall_clustering}")

    # 1.6 Diameter
    try:
        diameter = nx.diameter(G_comp)
        avg_diameter = nx.average_shortest_path_length(G_comp)
        print(f"Maximal Diameter: {diameter}")
        print(f"Average Diameter: {avg_diameter}")
    except Exception as e:
        print(f"Could not calculate diameter: {e}")
        diameter = "N/A"
        avg_diameter = "N/A"
```

```
97
98      # Write results
99      mode = 'a'
100     with open("analysis_results.txt", mode) as f:
101         f.write(f"\n--- {label} ---\n")
102         f.write(f"Nodes: {G.number_of_nodes()}\n")
103         f.write(f"Edges: {G.number_of_edges()}\n")
104         f.write(f"Average Clustering Coefficient: {avg_clustering
    :.4f}\n")
105         f.write(f"Overall Clustering Coefficient: {
    overall_clustering:.4f}\n")
106         f.write(f"Maximal Diameter: {diameter}\n")
107         f.write(f"Average Diameter: {avg_diameter if isinstance(
    avg_diameter, str) else f'{avg_diameter:.4f}'}\n")
108
109 # Clear results file
110 with open("analysis_results.txt", "w") as f:
111     f.write("Analysis Report Summary\n")
112
113 # Run analysis for both
114 # Parallel (original names maintained by empty prefix)
115 analyze_dataset('caltech_web_graph.pkl', 'Parallel', '')
116 # Sequential (prefixed names)
117 analyze_dataset('caltech_web_graph_np.pkl', 'Sequential', 'seq_')
```

Listing 2: Analysis Implementation

1.1.2 Selection Policy

We utilized a **Breadth-First Search (BFS)** selection policy. The crawler maintains a FIFO (First-In-First-Out) queue of URLs.

Strengths:

- **Trap Avoidance:** BFS is less susceptible to "infinite depth" traps (e.g., infinite calendar next-page links) compared to Depth-First Search (DFS).
- **Relevance:** It prioritizes nodes closer to the root (*caltech.edu*), which are typically more significant than deeply nested pages.
- **Parallelism Friendly:** The level-by-level exploration naturally supports our parallel batch processing approach.

Weaknesses:

- **Memory Overhead:** The frontier queue can grow large quickly as it stores all discovered-but-not-visited links.

1.1.3 Network Visualization

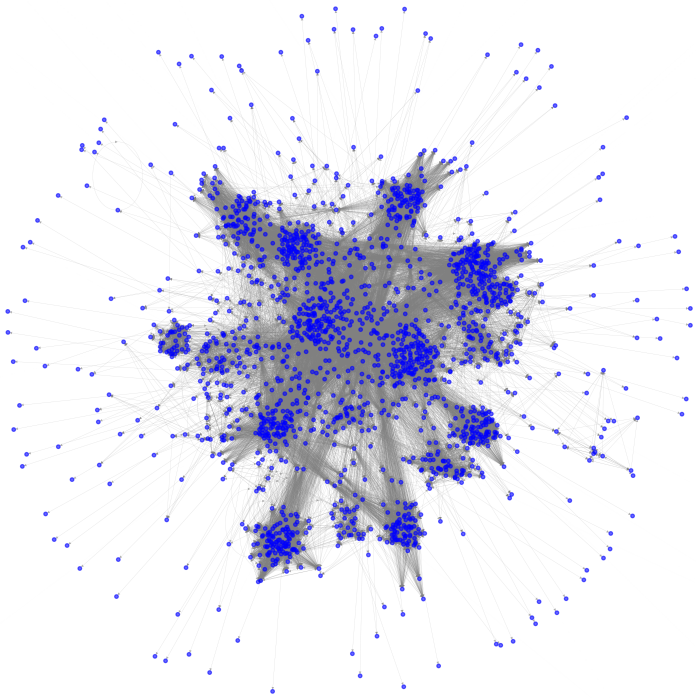

Figure 1: Visualization of the crawled Caltech web graph (Parallel).

1.1.4 Degree Distributions

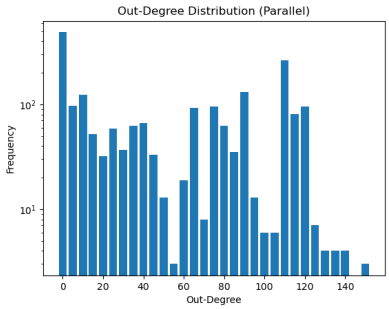

Figure 2: Out-Degree Histogram

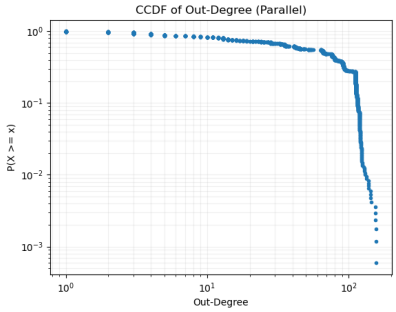

Figure 3: Out-Degree CCDF (Log-Log)

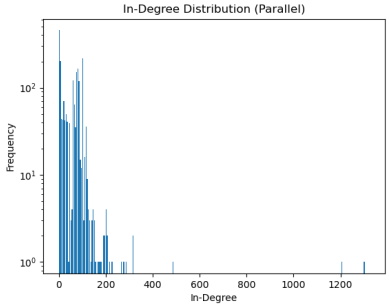

Figure 4: In-Degree Histogram

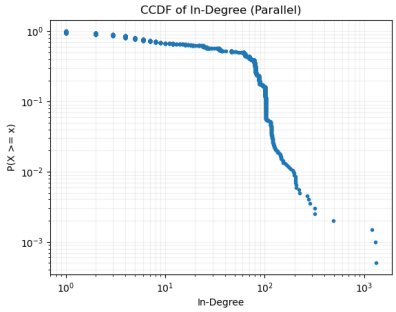

Figure 5: In-Degree CCDF (Log-Log)

1.1.5 Clustering Coefficients

The following values were calculated treating edges as undirected:

- **Average Clustering Coefficient:** 0.7867
- **Overall Clustering Coefficient:** 0.5905

1.1.6 Diameter Metrics

Calculated on the largest connected component of the undirected graph:

- **Maximal Diameter:** 4
- **Average Diameter:** 2.4183

1.1.7 Comparison with Collaboration Networks

Task: A comparison of the degree distribution, the clustering coefficients, and the diameters with those in Problem 2. Can you describe the similarities and differences regarding the "universal" properties between collaboration network and the web graph?

- **Similarities:**
- **Differences:**

1.1.8 Sequential Analysis (Retry)

The Cumulative Distribution Functions (CDFs) from the initial parallel crawl did not exhibit a clear power law behavior, which was unexpected for a web graph. Suspecting that the parallel execution might have influenced the sampling or traversal order non-deterministically, I implemented a sequential version of the crawler to verify the results.

```
from fetcher3 import fetch_links
import networkx as nx
import matplotlib.pyplot as plt
import pickle

start_url = "https://www.caltech.edu"
num_nodes_limit = 2000

G = nx.DiGraph()
G.add_node(start_url)

to_visit = [start_url]
visited = set()

while len(visited) < num_nodes_limit and to_visit:
    current_url = to_visit.pop(0)

    if current_url in visited:
        continue

    visited.add(current_url)

    try:
        links = fetch_links(current_url)
        if links:
            for link in links:
                if "caltech.edu" in link:
                    G.add_edge(current_url, link)
                    if link not in visited:
                        to_visit.append(link)
    except Exception:
        pass # Ignore fetch errors

    print(f"Crawled: {len(visited)} | Nodes: {G.number_of_nodes()}
    | Queue: {len(to_visit)}", end='\r')
```

```
35
36 # Clean up nodes that weren't visited
37 unvisited_nodes = [node for node in G.nodes() if node not in
       visited]
38 G.remove_nodes_from(unvisited_nodes)
39
40 print(f"\nFinal Graph: {G.number_of_nodes()} nodes, {G.
       number_of_edges()} edges")
41
42 # Save the graph with a different name
43 with open('caltech_web_graph_np.pkl', 'wb') as f:
44     pickle.dump(G, f)
45 print("Graph saved to caltech_web_graph_np.pkl")
46
47 # Plot the graph
48 plt.figure(figsize=(10, 10))
49 nx.draw(G,
50         pos=nx.spring_layout(G, k=0.15, iterations=20),
51         node_size=10,
52         width=0.1,
53         arrowsize=5,
54         with_labels=False,
55         node_color='green',
56         edge_color='gray',
57         alpha=0.6)
58 plt.title("Caltech Web Crawl (Sequential)")
59 plt.axis('off')
60 plt.savefig('caltech_web_graph_np.png', dpi=500)
```

Listing 3: Sequential Crawler Implementation

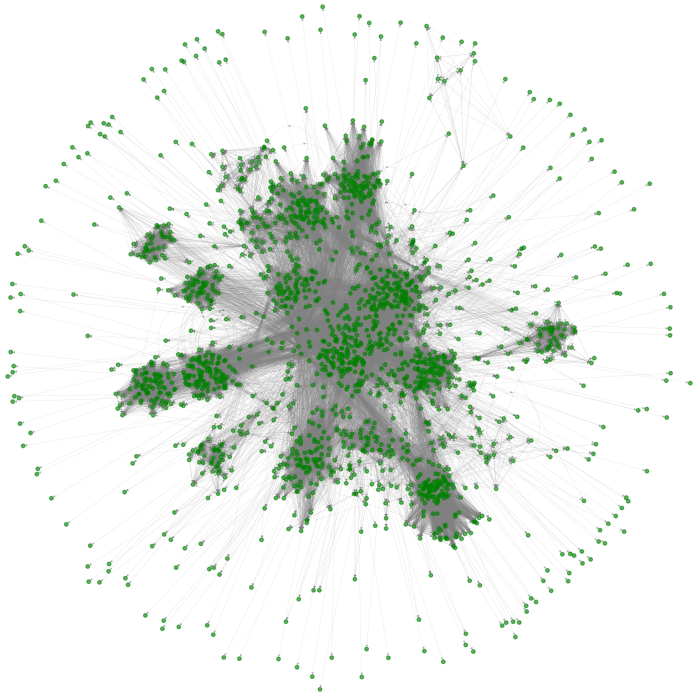

Figure 6: Visualization of the Caltech web graph (Sequential).

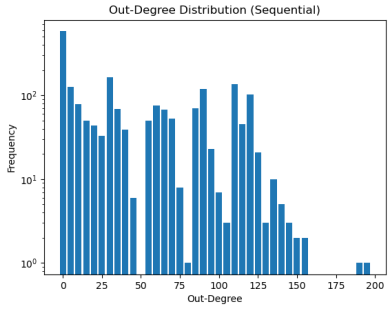

Figure 7: Sequential Out-Degree Histogram

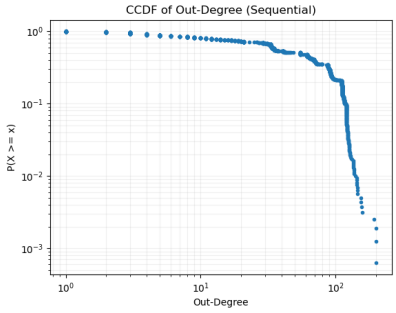

Figure 8: Sequential Out-Degree CCDF (Log-Log)

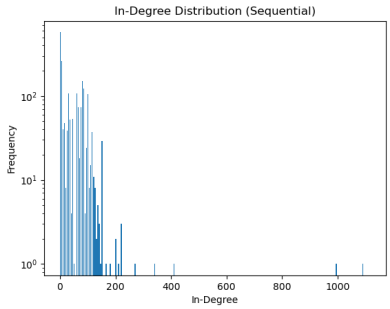

Figure 9: Sequential In-Degree Histogram

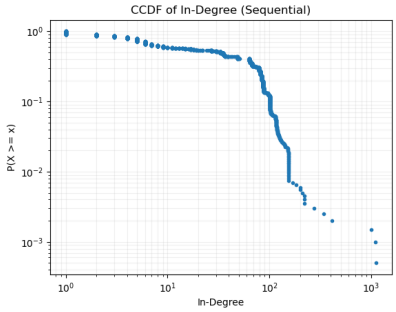

Figure 10: Sequential In-Degree CCDF (Log-Log)

The metrics for the sequential graph are:

- **Average Clustering Coefficient:** 0.7526
- **Overall Clustering Coefficient:** 0.6106
- **Maximal Diameter:** 4
- **Average Diameter:** 2.5954

Conclusion: The results from the sequential crawler are very similar to the parallel version. The graph structure, degree distributions, and clustering coefficients show no significant differences, suggesting that the parallel implementation correctly captures the graph properties and the observed distributions are intrinsic to the crawled subgraph.

1.2(a) Flight to the Queen Bee

The following paths were discovered on music-map.com. In both instances, the first path found was also the shortest.

- **Nikolai Rimsky-Korsakov → Beyoncé**
  Path: Nikolai Rimsky-Korsakov → Mozart → Distant Fires Burning → Chanyeol → BTS → Arctic Monkeys → Ed Sheeran → Adele → Beyoncé
- **Lyle Mays → Taylor Swift**
  Path: Lyle Mays → John Taylor → Mick Fleetwood → U2 → The Beatles → One Direction → Olivia Rodrigo → Taylor Swift

1.2(b) Know your Professors

The following paths were found using the "Co-authorship Path" tool on csauthors.net.

1. **Eric Mazumdar → Rudolf E. Kálmán** (Distance: 5)
   Source: https://csauthors.net/distance/eric-mazumdar/rudolf-e-kalman
   - Eric Mazumdar co-authored **10 papers** with Adam Wierman
   - Adam Wierman co-authored **8 papers** with Linqi Guo
   - Linqi Guo co-authored **4 papers** with Guangyi Shi
   - Guangyi Shi co-authored **1 paper** with Kazuo Toraichi
   - Kazuo Toraichi co-authored **2 papers** with Rudolf E. Kálmán
2. **Georgia Gkioxari → Paul Erdős** (Distance: 3)
   Source: https://csauthors.net/distance/georgia-gkioxari/paul-erdos
   - Georgia Gkioxari co-authored **12 papers** with Jitendra Malik
   - Jitendra Malik co-authored **2 papers** with Fan Chung Graham
   - Fan Chung Graham co-authored **7 papers** with Paul Erdős

1.2(d) Citations

I attempted to find paths for this section for over 30 minutes using multiple tools, including *Inciteful* and large language models like *Gemini*.

- **Inciteful:** This tool was problematic because it did not strictly follow the directed edge requirement (A cites B). Instead, it often connected papers if they simply cited the same source, resulting in invalid paths for this specific task. It found a distance of 3, but this was based on undirectional logic.
- **Gemini / LLMs:** Gemini proposed paths such as *Zipf (1936) → Simon (1955) → Price (1965) → Granovetter (1973)* and *Kalman (1960) → Bryson & Ho (1969) → Rumelhart et al. (1986) → LeCun et al. (1998).* However, verifying these specific citation links manually was extremely difficult and time-consuming, as it required accessing the full text of each paper to confirm the bibliography entries.

Due to these difficulties and the unreliability of automated tools for strict directed citation verification, I was unable to definitively confirm a valid path.