Social Network Analysis for Computer Scientists – Assignment 2

Benard Adala Wanyande (s4581733)

Leiden University, Leiden, The Netherlands

Reference

All definitions and notation follow the assignment instructions [1] by F. Takes.

1.1

The node clustering coefficient, denoted as $c(v)$, quantifies the fraction of existing connections among the neighbors of a given node v relative to all possible connections among those neighbors. For the average clustering coefficient $C(G)$ of a network to equal zero, each node's clustering coefficient $c(v)$ must also be zero. This situation arises when the network contains no triangles, or cycles of length three.

A tree network is one type of network with an average clustering coefficient of zero. A tree is a connected graph that contains no cycles; consequently, it cannot contain any triangles. For any node v in a tree with degree greater than one, none of its neighbors are connected to each other, making the number of edges among the neighbors zero. This results in $c(v) = 0$ for every node, and therefore $C(G) = 0$. A simple path or line graph represents a specific example of a tree.

Another type of network with $C(G) = 0$ is a bipartite network. In a bipartite graph, the set of vertices can be divided into two disjoint subsets, U and V, such that every edge connects a vertex in U to one in V, and there are no edges within either subset. Because any triangle would require a connection within one of these subsets, which is not allowed, bipartite graphs contain no triangles and therefore have an average clustering coefficient of zero.

In contrast, a complete graph, also known as a clique, represents a network in which the average clustering coefficient equals one. In a complete graph, every pair of distinct vertices is connected by a unique edge. For any node v, all of its neighbors are interconnected, forming a complete subgraph. The number of edges between the neighbors of v is given by

$$\frac{\deg(v) \cdot (\deg(v) - 1)}{2}.$$

The numerator in the definition of $c(v)$ equals twice this value, resulting in $\deg(v) \cdot (\deg(v) - 1)$, which matches the denominator exactly. Hence, $c(v) = 1$ for every node, and the average clustering coefficient is $C(G) = 1$.

1.2

This can be proven by contradiction.

Statement to Prove There exists a node $v \in V$ such that $c(v) \geq C(G)$.

Definition of Average Clustering Coefficient The average clustering coefficient is defined as:

$$C(G) = \frac{1}{n} \sum_{u \in V} c(u), \quad \text{where } n = |V|.$$

Proof by Contradiction Let's assume the opposite of the statement is true. The negation of "there exists a v such that $c(v) \geq C(G)$" is "for all $v \in V$, $c(v) < C(G)$".

1. Assume that for every node v in the network, $c(v) < C(G)$.
2. We can sum this inequality over all n nodes in V:

$$\sum_{v \in V} c(v) < \sum_{v \in V} C(G)$$

3. Since $C(G)$ is a constant value for the entire graph, summing it n times is equivalent to multiplying it by n:

$$\sum_{v \in V} C(G) = n \cdot C(G)$$

4. Substituting this back into the inequality, we get:

$$\sum_{v \in V} c(v) < n \cdot C(G)$$

5. Now, let's rearrange the original definition of $C(G)$ by multiplying both sides by n:

$$n \cdot C(G) = \sum_{v \in V} c(v)$$

6. By substituting this equality into our derived inequality from the previous step, we arrive at a contradiction:

$$\sum_{v \in V} c(v) < \sum_{v \in V} c(v)$$

Conclusion The statement that a sum is strictly less than itself is a logical impossibility. Therefore, our initial assumption ("for all $v \in V$, $c(v) < C(G)$") must be false. This proves that the original statement is true: there must exist at least one node v for which $c(v) \geq C(G)$.

1.3

The degree of all nodes in V_L in the original bipartite graph G_0 must be 1.

In the bipartite projection G, two nodes from V_L are connected if they share at least one common neighbor in V_R in the graph G_0. Consider a single node $r \in V_R$ with degree k in G_0. This node r is connected to k distinct nodes in V_L, which we denote as the set $N(r)$. In the projection G, every node in $N(r)$ is connected to every other node in $N(r)$ because they all share the common neighbor r. Consequently, the set of nodes $N(r)$ forms a clique of size k in G, and therefore, they all belong to the same connected component.

The problem states that the connected component size distribution of G is identical to the degree distribution of V_R. This implies a one-to-one correspondence: each node $r \in V_R$ with degree k must produce a distinct connected component of size k in G.

For the components generated by two different nodes, say $r_1 \in V_R$ and $r_2 \in V_R$, to remain separate or disjoint, there must be no path in G connecting the neighbors of r_1 (the set $N(r_1)$) to the neighbors of r_2 (the set $N(r_2)$). A path would exist if any node in $N(r_1)$ were also a neighbor of another node in V_R that could eventually link to $N(r_2)$. The simplest way to ensure that the components remain disjoint is to require the neighbor sets $N(r)$ to be disjoint for all $r \in V_R$; in other words, $N(r_1) \cap N(r_2) = \emptyset$ for any distinct $r_1, r_2 \in V_R$.

If the neighbor sets in V_L for every node in V_R must be disjoint, then any given node $v \in V_L$ can be a neighbor to at most one node in V_R. Since the graph is connected, and each node must have at least one neighbor, every node $v \in V_L$ in the bipartite graph G_0 must have degree exactly 1.

If $\deg(v) = 1$ for all $v \in V_L$, then each $r \in V_R$ with degree k generates a component of size k in G that is completely isolated from the components generated by other nodes in V_R, thereby satisfying the stated condition.

1.4

This relationship can be derived by counting the total number of edges in the graph in two different ways.

Definitions

- Let V_1 and V_2 be the two sets of nodes (partitions).
- $n_1 = |V_1|$ is the number of nodes of type one.
- $n_2 = |V_2|$ is the number of nodes of type two.
- E is the set of edges in the graph, and $m = |E|$ is the total number of edges.
- k_1 is the mean degree of nodes in V_1.
- k_2 is the mean degree of nodes in V_2.

Sum of Degrees The total number of edges m can be expressed as the sum of degrees of the nodes in one partition. Since every edge in a bipartite graph connects a node from V_1 to a node in V_2, each edge contributes exactly one to the degree sum of V_1 and exactly one to the degree sum of V_2. Therefore, the sum of the degrees of all nodes in V_1 is equal to the total number of edges:

$$\sum_{v \in V_1} \deg(v) = m$$

Similarly, the sum of the degrees of all nodes in V_2 is also equal to the total number of edges:

$$\sum_{u \in V_2} \deg(u) = m$$

Relating to Mean Degree The mean degree of a set of nodes is the sum of their degrees divided by the number of nodes.

- For V_1:

$$k_1 = \frac{\sum_{v \in V_1} \deg(v)}{n_1}$$

- For V_2:

$$k_2 = \frac{\sum_{u \in V_2} \deg(u)}{n_2}$$

Derivation of the Relationship From the previous step, we can express the sum of degrees in terms of the mean degree:

$$\sum_{v \in V_1} \deg(v) = n_1 \cdot k_1$$

$$\sum_{u \in V_2} \deg(u) = n_2 \cdot k_2$$

Since both sums are equal to m, we can set them equal to each other:

$$n_1 \cdot k_1 = n_2 \cdot k_2$$

To show the desired relationship, we simply solve for k_2:

$$k_2 = \frac{n_1}{n_2} \cdot k_1$$

This completes the proof.

1.5

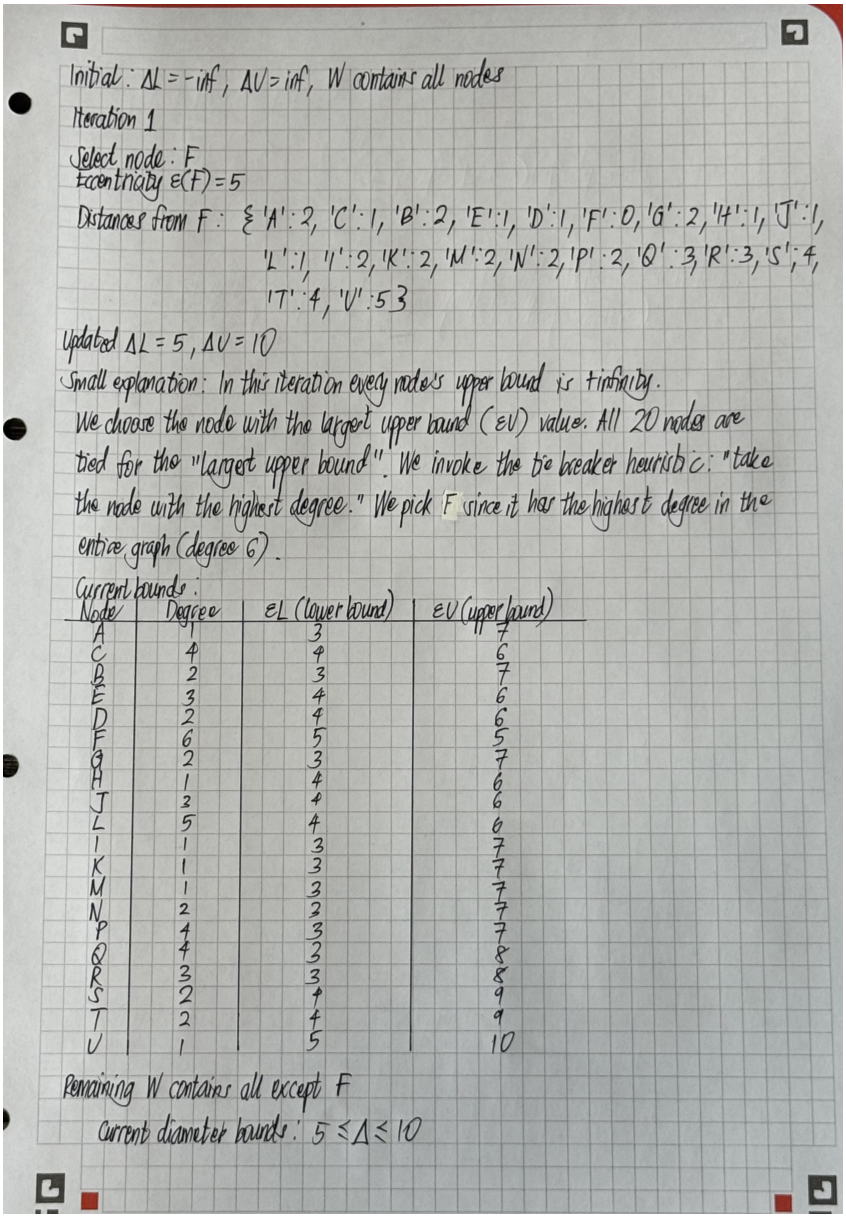
If the node rankings from degree centrality and betweenness centrality are identical, it implies that a node's number of direct connections is perfectly correlated with its importance as a bridge on shortest paths. This occurs in specific, highly structured networks.

One such network is the star graph. A star graph consists of a single central node connected to $n-1$ peripheral or leaf nodes, with no connections between the peripheral nodes themselves. In terms of degree centrality, the central node has a degree of $n-1$, while each peripheral node has a degree of 1. The ranking is therefore unambiguous: the central node ranks highest, followed by all peripheral nodes, which are tied for the lowest rank. For betweenness centrality, the shortest path between any two peripheral nodes must pass through the central node, meaning the central node lies on $\binom{n-1}{2}$ shortest paths and thus has very high betweenness centrality. Peripheral nodes, on the other hand, never lie on any shortest paths between other nodes and therefore have a betweenness centrality of 0. The ranking for both measures is therefore identical: the central node ranks first, and all peripheral nodes are tied for last place.

Another example is the complete graph, or clique. In a complete graph, every node is connected to every other node. Consequently, each node has the same degree, $n-1$, making all nodes tied for first place in degree centrality. For betweenness centrality, the shortest path between any two nodes s and t is simply the direct edge (s,t), with no other node lying on that path. Thus, the betweenness centrality of every node in a complete graph is 0. Because all nodes have identical values for both degree and betweenness centrality, the rankings are also identical—though in this case, trivially so, as all nodes share the same rank.

Question 2 - Handwritten Solutions

The following pages present the scanned handwritten solution for this section.



Initial: $\Delta L = -\inf$, $\Delta U = \inf$, W contains all nodes

Iteration 1

Select node: F

Eccentricity $\varepsilon(F) = 5$

Distances from F: {'A': 2, 'C': 1, 'B': 2, 'E': 1, 'D': 1, 'F': 0, 'G': 2, 'H': 1, 'J': 1, 'L': 1, 'I': 2, 'K': 2, 'M': 2, 'N': 2, 'P': 2, 'Q': 3, 'R': 3, 'S': 4, 'T': 4, 'V': 5}

Updated $\Delta L = 5$, $\Delta U = 10$

Small explanation: In this iteration every node's upper bound is infinity. We choose the node with the largest upper bound (εU) value. All 20 nodes are tied for the "largest upper bound". We invoke the tie breaker heurisbic: "take the node with the highest degree." We pick F since it has the highest degree in the entire graph (degree 6).

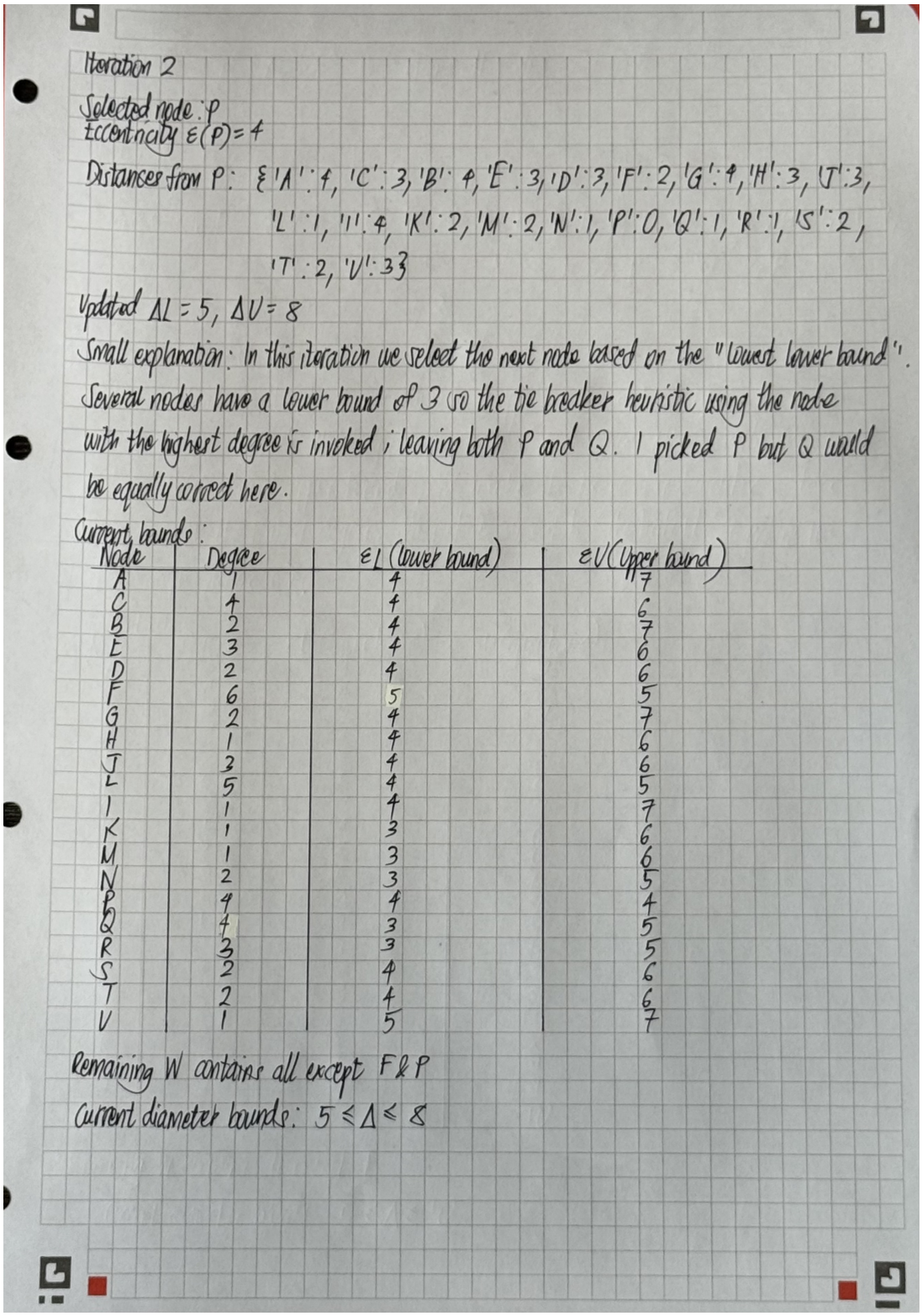
Current bounds:

Node	Degree	εL (lower bound)	εU (upper bound)
A	1	3	7
C	4	4	6
B	2	3	7
E	3	4	6
D	2	4	6
F	6	5	5
G	2	3	7
H	1	4	6
J	3	4	6
L	5	4	6
I	1	3	7
K	1	3	7
M	1	3	7
N	2	3	7
P	4	3	7
Q	4	3	8
R	3	3	8
S	2	4	9
T	2	4	9
V	1	5	10

Remaining W contains all except F

Current diameter bounds: $5 \leq \Delta \leq 10$

Fig. 1: Page 1.



Iteration 2

Selected node: P
Eccentricity $\varepsilon(P) = 4$

Distances from P: {'A': 4, 'C': 3, 'B': 4, 'E': 3, 'D': 3, 'F': 2, 'G': 4, 'H': 3, 'J': 3, 'L': 1, 'I': 4, 'K': 2, 'M': 2, 'N': 1, 'P': 0, 'Q': 1, 'R': 1, 'S': 2, 'T': 2, 'V': 3}

Updated $\Delta L = 5$, $\Delta U = 8$

Small explanation: In this iteration we select the next node based on the "lowest lower bound". Several nodes have a lower bound of 3 so the tie breaker heuristic using the node with the highest degree is invoked; leaving both P and Q. I picked P but Q would be equally correct here.

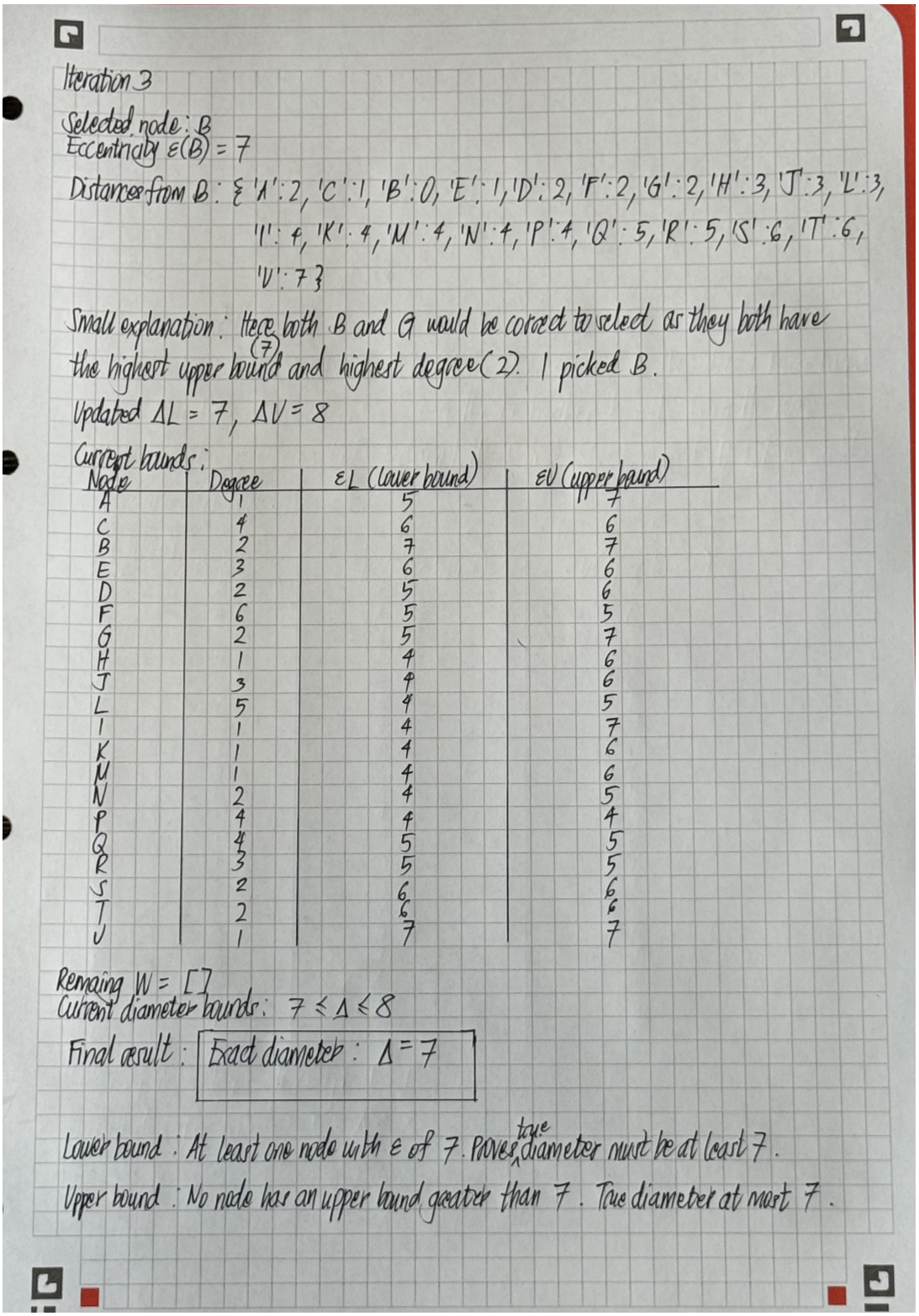
Current bounds:

Node	Degree	εL (lower bound)	εU (upper bound)
A	1	4	7
C	4	4	6
B	2	4	7
E	3	4	6
D	2	4	6
F	6	5	5
G	2	4	7
H	1	4	6
J	3	4	6
L	5	4	5
I	1	4	7
K	1	3	6
M	1	3	6
N	2	3	5
P	4	4	4
Q	4	3	5
R	3	3	5
S	2	4	6
T	2	4	6
V	1	5	7

Remaining W contains all except F & P

Current diameter bounds: $5 \leq \Delta \leq 8$

Fig. 2: Page 2.



Iteration 3

Selected node: B
Eccentricity $\varepsilon(B) = 7$

Distances from B: {'A': 2, 'C': 1, 'B': 0, 'E': 1, 'D': 2, 'F': 2, 'G': 2, 'H': 3, 'J': 3, 'L': 3, 'I': 4, 'K': 4, 'M': 4, 'N': 4, 'P': 4, 'Q': 5, 'R': 5, 'S': 6, 'T': 6, 'V': 7}

Small explanation: Here both B and G would be correct to select as they both have the highest upper bound (7) and highest degree (2). I picked B.

Updated $\Delta L = 7$, $\Delta V = 8$

Current bounds:

Node	Degree	εL (Lower bound)	εU (Upper bound)
A	1	5	7
C	4	6	6
B	2	7	7
E	3	6	6
D	2	5	6
F	6	5	5
G	2	5	7
H	1	4	6
J	3	4	6
L	5	4	5
I	1	4	7
K	1	4	6
M	1	4	6
N	2	4	5
P	4	4	4
Q	3	5	5
R	3	5	5
S	2	6	6
T	2	6	6
V	1	7	7

Remaining W = []
Current diameter bounds: $7 \leq \Delta \leq 8$

Final result: **Exact diameter: $\Delta = 7$**

Lower bound: At least one node with ε of 7. Proves true diameter must be at least 7.

Upper bound: No node has an upper bound greater than 7. True diameter at most 7.

Fig. 3: Page 3.

Reflection on Algorithm Efficiency The implementation of the Bounding Diameters algorithm on the given graph demonstrated notable computational efficiency. The algorithm successfully completed its execution in only three iterations.

This outcome can be compared to that of the naive approach for computing graph diameters, which relies on an all-pairs shortest path (APSP) computation. In the naive method, a breadth-first search (BFS) is performed from each node in the network. Given that the graph in question consists of 20 nodes, this would require a total of 20 BFS computations to exhaustively determine all shortest paths and identify the maximum among them. In contrast, the Bounding Diameters algorithm achieved the same result using only four BFS computations. This represents a substantial reduction in computational effort, highlighting the efficiency of the algorithm's bounding mechanism.

The key to this improvement lies in the algorithm's use of eccentricity bounds, which are iteratively refined as each node is processed. By updating these bounds and eliminating nodes that no longer influence the maximum distance estimation, the algorithm rapidly narrows the search space. Consequently, it avoids redundant BFS operations while still guaranteeing convergence to the exact diameter.

The Bounding Diameters algorithm was able to determine the exact diameter of seven using only four BFS executions, as opposed to the twenty required by the naive APSP method. This efficiency demonstrates the practical advantage of bounding-based strategies in diameter estimation, particularly for larger graphs where exhaustive search methods become computationally prohibitive.

3.1

Extraction of the Mention Graph

The mention graph was successfully extracted from the input Twitter datasets. To ensure full reproducibility, the datasets were obtained programmatically using the `requests` API in Python, directly from the URLs provided in the assignment instructions. Both the `twitter-small.tsv` and `twitter-larger.tsv` datasets were downloaded and stored locally if they did not already exist. The complete code implementation can be found in **Appendix A**.

The primary objective of the implementation was to generate an adjacency list in which, for each user, all other users they mentioned were recorded, together with the count of how many times each mention occurred. The output is a `.csv` file representing a weighted directed edge list in the following format:

Table 1: Example of adjacency list (weighted edge list) output format.

Source	Target	Weight
aeneas	achilles	1
aeneas	hector	3
hector	achilles	2

The original input data in the `twitter-small.tsv` and `twitter-larger.tsv` files were structured as tab-separated values (TSV) with each row consisting of a timestamp, username, and tweet content, as illustrated below.

Table 2: Example of raw Twitter dataset format.

Timestamp	User	Tweet Content
2009-07-05 14:07:18	aeneas	Hi @achilles, how are you? #old
2009-07-05 14:12:45	hector	Fighting @achilles again...
2009-07-05 14:13:57	aeneas	Cheers @hector! Great battle!

The extraction process was implemented in two modular functions: `download_file(url, local_filename)` and `extract_mention_graph(input_tsv_path, output_csv_path)`.

File Download and Reproducibility. The first helper function, `download_file`, takes the dataset URL and a local filename as arguments. It verifies whether the file already exists locally, and if not, downloads it using the Python `requests` package. This guarantees that the extraction process can be fully reproduced on any system, independent of manual file acquisition.

Mention Graph Extraction. The second helper function, `extract_mention_graph`, performs the parsing and construction of the weighted mention graph. The dataset is read line by line, ensuring scalability to larger files and efficient memory management. The mention graph was represented internally as a nested dictionary using the `defaultdict` structure from Python's `collections` module:

```
{sender: {mentioned_user: count}}
```

This representation was chosen for its efficiency in both user addition and mention count incrementation. Type hints were incorporated to enhance code clarity and ensure parsing integrity.

Mention Extraction via Regular Expressions. To accurately identify mentions, a dedicated regular expression was designed and compiled as follows:

```
@([A-Za-z0-9_]{1,15})\b
```

The logic behind this pattern is summarized in Table 3.

Table 3: Regular expression logic for valid Twitter username extraction.

Component	**Description**
`@`	Matches the literal '@' symbol indicating a mention.
`[A-Za-z0-9_]{1,15}`	Captures a sequence of 1–15 alphanumeric or underscore characters, consistent with Twitter's username policy.
`\b`	Enforces a word boundary to prevent partial matches within email addresses (e.g., `user@example.com`).

Data Quality Monitoring. During parsing, counters were maintained for both the number of processed lines and the number of malformed lines. Malformed lines—those not adhering to the three-field TSV structure—were skipped, and their count recorded for diagnostics. Additional safeguards were implemented for case sensitivity and Unicode decoding. All usernames were normalized to lowercase to ensure that `@UserA` and `@usera` referred to the same node in the graph.

Expected Issues and Mitigation. Throughout the extraction process, multiple potential issues were identified and systematically addressed, as summarized in Table 4.

Table 4: Identified challenges and mitigation strategies during mention graph extraction.

Issue	**Reason/Example**	**Solution Implemented**
Case insensitivity	Usernames such as `@UserA` and `@usera` are equivalent.	All usernames converted to lowercase during parsing.
Defining valid usernames	Avoid matching partial strings in email addresses or long tokens.	Strict regex enforcing Twitter username rules and word boundaries.
Text mining considerations	Avoid false positives for non-mention '@' symbols.	Pattern ensures '@' appears at token start and followed by valid username characters.
Malformed or corrupted lines	Missing tab-separated fields.	Line skipped and counter incremented; process continued without interruption.
Scalability and performance	Large dataset size (`twitter-larger.tsv`).	Line-by-line parsing with memory-efficient data structures.
Unicode/encoding problems	Tweets containing emojis or non-ASCII text.	Files read in UTF-8 with error replacement to prevent crashes.
Tabs inside tweet content	Tweets may contain tabs beyond the delimiter.	Split limited to first two tabs (`split('\t', 2)`).
Multiple mentions and duplicates	Same user mentioned multiple times per tweet.	Each occurrence counted as one directed mention edge (incremental weight).

Final Results. After iterative refinements and multiple runs, the parser achieved zero malformed lines, indicating that all data quality and parsing issues were successfully resolved. The final weighted edge list files were generated for both `twitter-small.tsv` and `twitter-larger.tsv`, confirming that the mention graph extraction was both accurate and fully reproducible.

3.2 and 3.6

Network Characterization and Structural Analysis

Following the successful extraction of the mention graphs, a structural analysis was conducted using the `NetworkX` library in Python to characterize their topological properties. Both the `twitter-small` and `twitter-larger` datasets were analyzed to compare the structure and connectivity patterns of the mention networks. The key metrics obtained are summarized in Table 5.

Table 5: Topological properties of the extracted Twitter mention graphs.

Metric	**twitter-small**	**twitter-larger**
Number of Nodes (Users)	85,239	470,323
Number of Edges (Mentions)	154,089	1,309,987
Weakly Connected Components	2,857 (Giant: 75,114 nodes)	10,717 (Giant: 439,997 nodes)
Strongly Connected Components	79,850 (Largest: 4,857 nodes)	412,521 (Largest: 54,889 nodes)
Density	2.12e-05	5.92e-06
Avg. Clustering Coefficient	0.0479	0.0727
Avg. Distance (Giant Component)	6.2425	5.3767

The degree distributions for both mention graphs were plotted on a log-log scale, as shown in Figures 4 and 5. The approximately linear trend observed in both indegree and outdegree distributions suggests a power-law behavior, characteristic of scale-free networks. This implies the presence of "hub" nodes—users who are either highly mentioned (high indegree) or mention many others (high outdegree).

An interesting observation is the appearance of a plateau in the outdegree distribution of both graphs, roughly spanning between $10^{0.5}$ and 10^{1}. This plateau indicates a region where a significant proportion of users mention between 3 and 10 others—a behavior consistent with typical Twitter interaction patterns. In social communication contexts, this suggests a saturation point in user engagement, where the majority of users actively interact with a small but stable group rather than a continuously growing set of new users. Such plateaus are commonly seen in online social networks where user interaction tends to be constrained by cognitive and temporal limits (see, e.g., Dunbar's number effect).

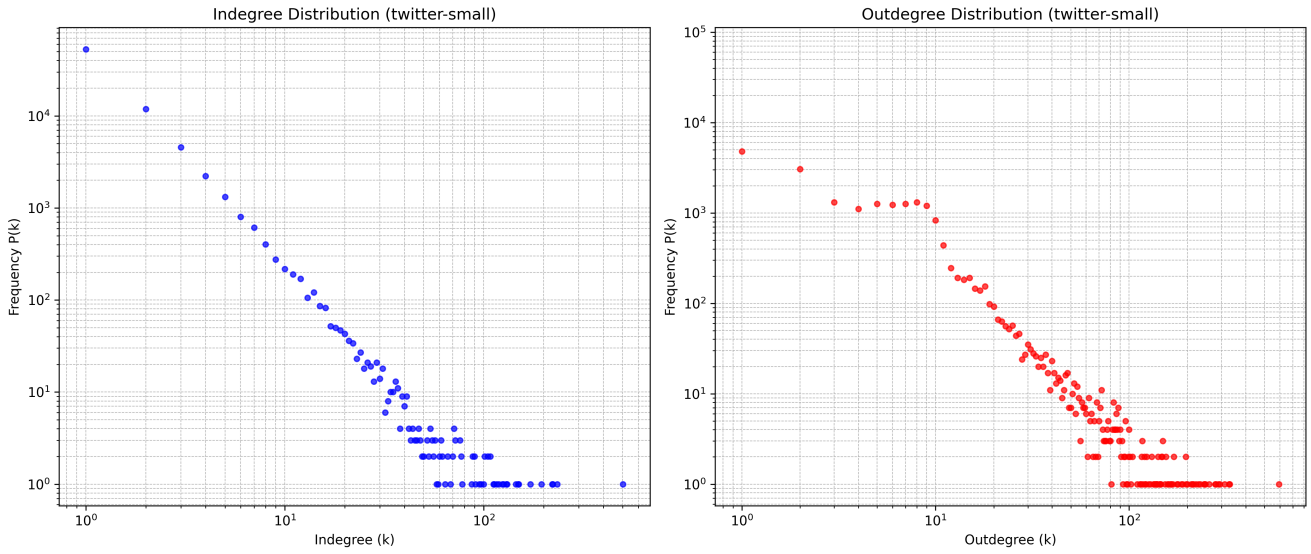


Fig. 4: Indegree and outdegree distributions for the `twitter-small` mention graph on a log-log scale.

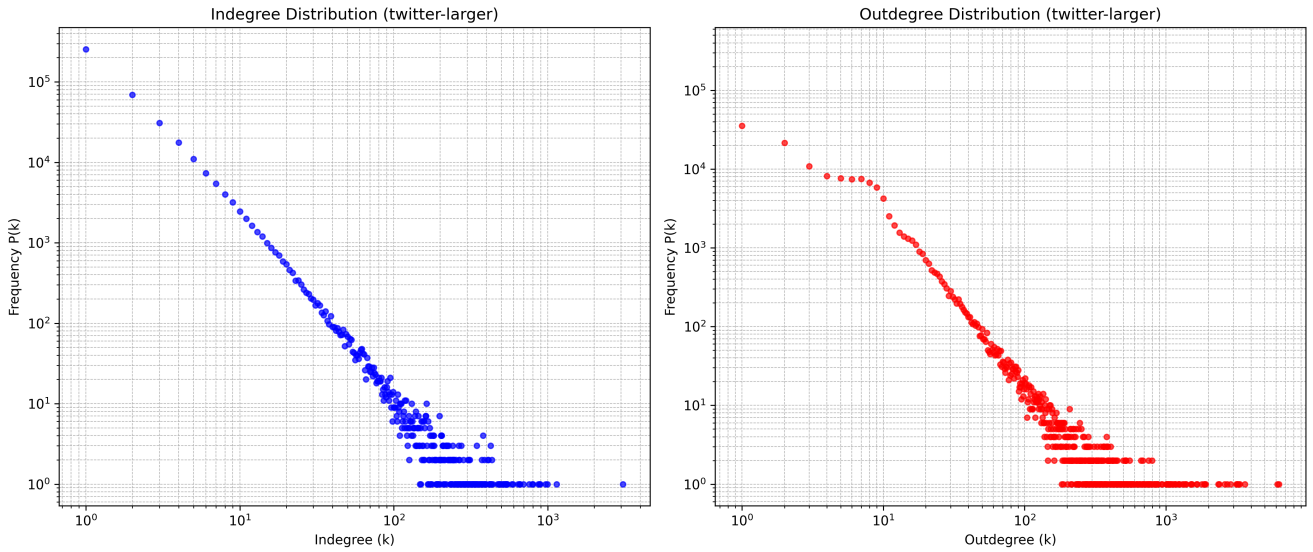
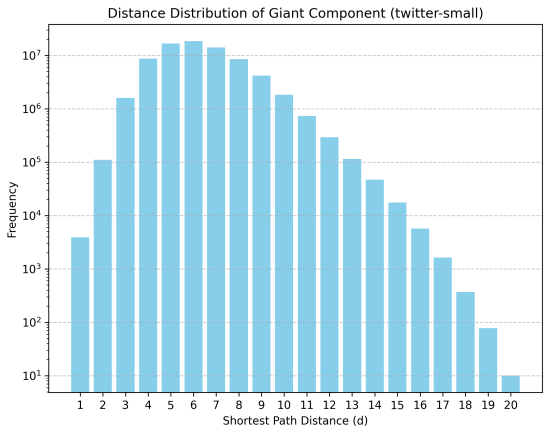


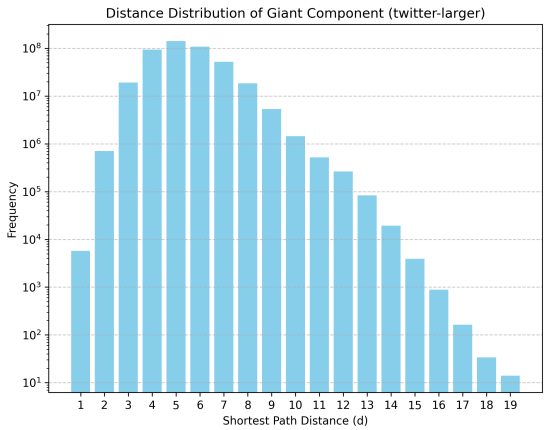
Fig. 5: Indegree and outdegree distributions for the `twitter-larger` mention graph on a log-log scale.

The distribution of shortest path distances within the undirected giant components was further analyzed for both datasets, as shown in Figures 6a and 6b. Both graphs exhibit the "small-world" property, characterized by short average path lengths and the presence of tightly clustered local communities. Despite the significant difference in size, both networks maintain average path lengths between 5 and 6, implying that information or influence can traverse the network with only a few intermediaries.

Interestingly, the larger mention graph exhibits a slightly smaller average path length (5.38) compared to the smaller one (6.24), suggesting that the addition of more nodes and edges increased the overall connectivity of the graph. This observation aligns with the densification phenomenon observed in evolving social networks, where the network becomes increasingly interconnected as it grows.



(a) Shortest path distance distribution in the giant component of the `twitter-small` graph.



(b) Approximated shortest path distance distribution in the giant component of the `twitter-larger` graph.

Fig. 6: Comparison of shortest path distance distributions between `twitter-small` and `twitter-larger` mention graphs.

3.3

Top 20 Users Based On 3 Different Centrality Measures

To identify the most influential users within the network, we computed three distinct centrality measures. The analysis was performed on the directed mention graph, with specific considerations for handling directionality for each measure. The top 20 users according to Indegree, Closeness, and Betweenness centrality are presented in Table 6.

Table 6: Top 20 Users by Centrality in the Analyzed Twitter Graph.

Rank	Indegree Centrality	Closeness Centrality	Betweenness Centrality
1	mashable	mashable	tamaraschilling
2	matt_369	matt_369	americandream09
3	tweetmeme	starlingpoet	centerpet
4	starlingpoet	holdemtalkradio	straightstreet
5	holdemtalkradio	bonniestwit	lauralassiter
6	bonniestwit	tamaraschilling	modelsupplies
7	tamaraschilling	thelifehackpost	medic_ray
8	emarketingguru	americandream09	elocio
9	centerpet	lorimoreno	doc_remy
10	thelifehackpost	centerpet	jhillstephens
11	sharonhayes	faithgoddess7	bacieabbracci
12	elocio	modelsupplies	josephranseth
13	jonnerz	americanwomannn	bobcallahan
14	jason_pollock	billzucker	drjennifer
15	americanwomannn	jason_pollock	nachhi
16	peterfacinelli	emarketingguru	sharonhayes
17	fridayluv	weizenbaum	teddy_salad
18	straightstreet	tap29	dpbkmb
19	faithgoddess7	sharonhayes	fizzyduck
20	americandream09	nurul54	thelifehackpost

Handling Directionality The directed nature of the mention graph (user A mentions user B) is fundamental to the interpretation of centrality. Our approach, reflected in the Python script, accounts for this as follows:

- **Degree Centrality:** We specifically use **indegree centrality**. In a mention network, a user's indegree reflects the number of times they are mentioned by others, serving as a direct measure of their popularity, visibility, or prestige. Outdegree, by contrast, would simply measure a user's activity. The script correctly calls `nx.in_degree_centrality(G)`.
- **Closeness Centrality:** This measure is calculated on the directed graph, quantifying how efficiently a user can be *reached* by others. A high closeness score suggests a user is well-positioned to receive information quickly from many parts of the network, placing them in the "center" of conversational flows.
- **Betweenness Centrality:** This measure quantifies a user's role as a "broker" or "bridge." It is calculated by counting how often a user lies on the directed shortest paths between other pairs of users. A user with high betweenness connects otherwise disparate communities. As this calculation is computationally intensive, the script approximates the value for large graphs by sampling a subset of nodes, a standard practice for performance.

Discussion of Results The rankings in Table 6 reveal a fascinating and complex structure of influence within this network. A key observation is the significant divergence between the lists.

- **Indegree and Closeness Centrality** show some overlap at the very top (e.g., `mashable`, `matt_369`), indicating that some popular users are also centrally located. However, the lists quickly diverge, suggesting that being

popular doesn't automatically mean a user is in the absolute "heart" of the network. The presence of accounts like `tweetmeme` and `mashable` points to a core of news aggregators and media, while personal accounts like `starlingpoet` and `holdemtalkradio` indicate influential individuals within specific hobbyist communities.

– **Betweenness Centrality** presents a nearly completely different set of users. The top brokers, such as `tamaraschilling`, `americandream09`, and `centerpet`, are not the most popular users. This is a critical insight: in this network, the roles of "celebrity" (high indegree) and "connector" (high betweenness) are largely decoupled. These high-betweenness users are indispensable to the network's cohesion, bridging conversations between different communities (e.g., connecting the "pet lovers" community to a general social group), even if they are not famous in their own right. Their influence is structural rather than based on sheer popularity.

Comparison of Ranking Similarity To formally quantify the similarity between these rankings, we applied **Kendall's Rank-Order Correlation Coefficient (τ)**. This non-parametric statistic measures the correspondence between two ranked lists. A τ of 1 indicates perfect agreement, 0 indicates no correlation, and -1 indicates perfect disagreement. The correlation was calculated between the top-20 lists for each pair of centrality measures.

Table 7: Kendall's Tau (τ) Correlation Between Top 20 Centrality Rankings.

Ranking Pair	**Kendall's Tau (τ)**
Indegree vs. Closeness	0.4316
Indegree vs. Betweenness	-0.0105
Closeness vs. Betweenness	0.0421

Interpretation: The Kendall's Tau results provide a stark, quantitative confirmation of our qualitative observations:

1. **Indegree vs. Closeness ($\tau = 0.4316$):** There is a moderate positive correlation. This is expected and shows that popular users tend to be more centrally located than a random user, but the relationship is not exceptionally strong.
2. **Indegree vs. Betweenness ($\tau \approx -0.01$):** There is effectively **zero correlation**. This number powerfully reinforces that the most popular users are not the ones bridging communities. The network's structure does not rely on its "stars" to hold it together.
3. **Closeness vs. Betweenness ($\tau \approx 0.04$):** Similarly, there is **zero correlation**. The users who are most central to information reception are not the ones responsible for connecting disparate groups.

This analysis reveals a decentralized network where influence is multifaceted. Unlike networks dominated by a few central hubs, here the critical functions of popularity, information centrality, and community bridging are distributed among different sets of users.

article graphicx subcaption [section]placeins booktabs

3.4 Community Detection Analysis

To analyze the meso-scale organization of the mention graph, a community detection algorithm was applied to the giant component. This process aimed to partition the network into distinct clusters of users who interact more densely with each other than with the rest of the network, thereby revealing underlying thematic and social structures.

Methodology

The analysis was conducted using **Gephi**. The giant weakly connected component was first isolated to focus on the network's core. The **Modularity** algorithm was then executed to partition the nodes. The algorithm yielded **110 distinct communities** with a high modularity score of **0.655**, indicating a statistically significant and well-defined community structure.

For visualization, the **OpenOrd** layout algorithm was applied, as it is designed to spatially separate clusters in the visual space. Nodes were colored according to their *community assignment (Modularity Class)* and sized proportionally to their **In-Degree**, making the most influential users in each community visually prominent. The subsequent manual analysis focused on the four largest and most coherent communities.

Results and Discussion

The analysis reveals a network with a clear **core–periphery structure**, where a few large, densely interconnected communities form the central mass, surrounded by numerous smaller, more specialized groups. The overall structure is visualized in Figure 7. A detailed investigation into the most prominent of these communities reveals a diverse landscape of conversations, a selection of which are summarized in Table 8.

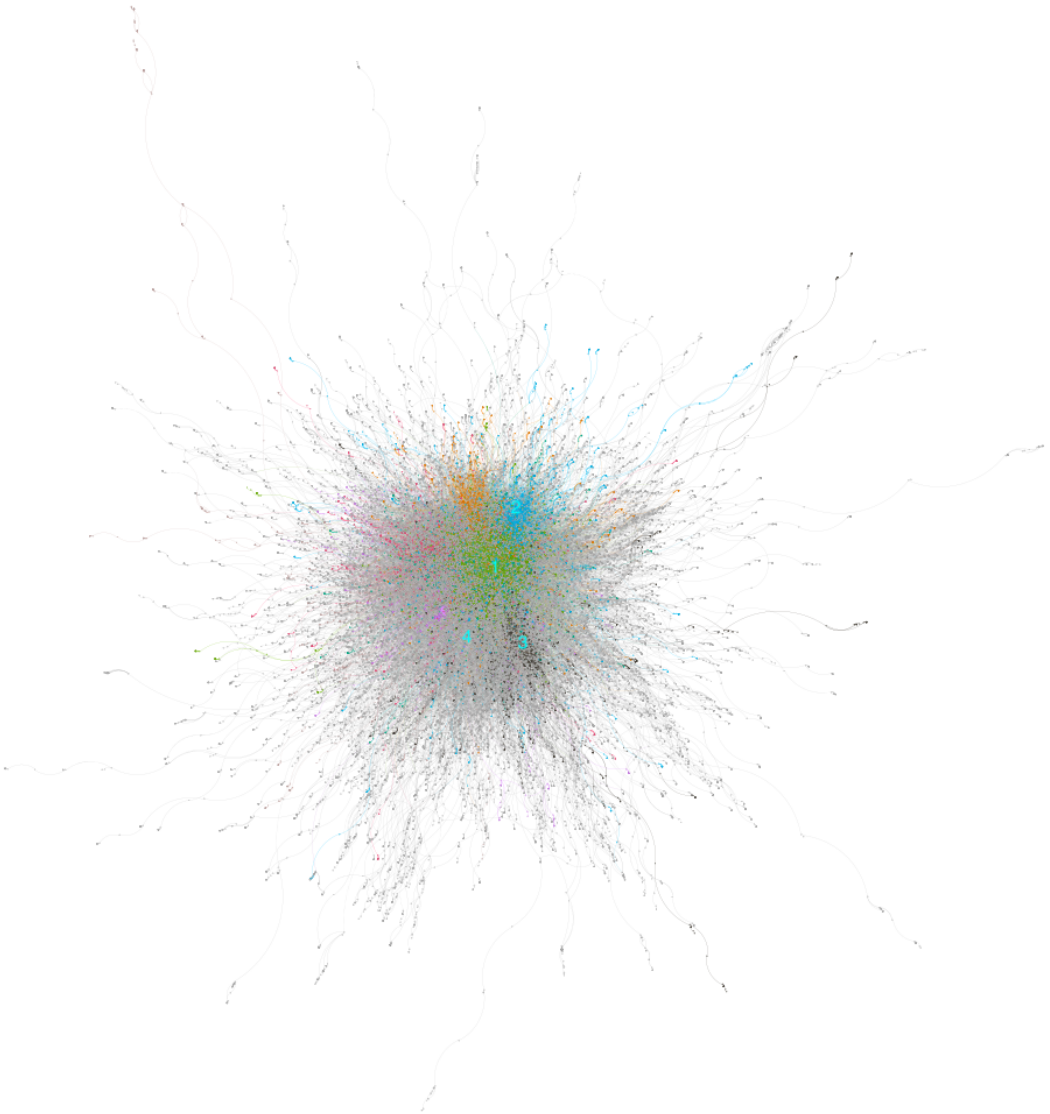


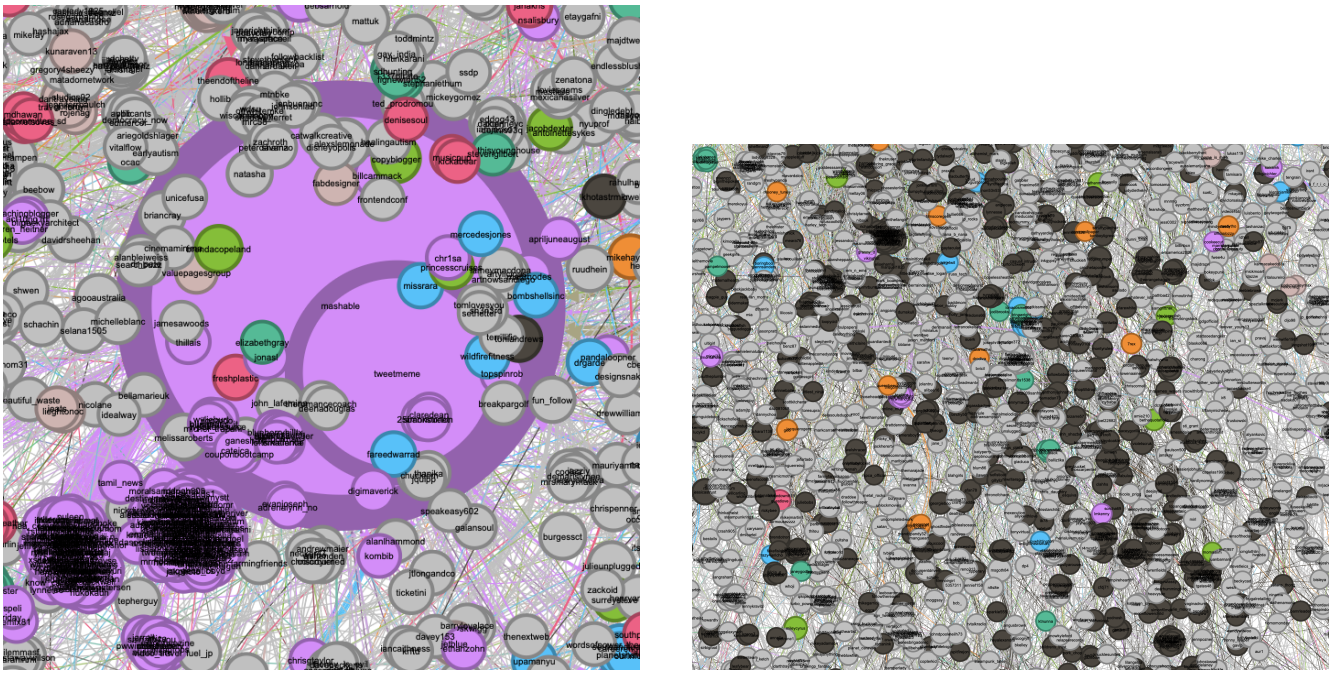
Fig. 7: The giant component of the mention graph. Nodes are colored by community and sized by In-Degree. Major thematic communities, as identified in the manual analysis, are annotated.

Table 8: Summary and interpretation of key detected communities.

Rank	Approx. Size	Interpreted Theme	Top Members (by In-Degree)
1	[Approx. 1,200] users	Tech News & Social Media Marketing	`mashable, tweetmeme, jonasm, elizabethgray, freshplastic`
2	[Approx. 950] users	E-commerce & Online Shopping	`etsy, stylehive, shopstyle, shopittome, thinkgeek`
3	[Approx. 900] users	Breaking News & Mainstream Media	`breakingnews, cnnbrk, nytimes, aolnews, usatoday`
4	[Approx. 750] users	Tech Influencers & Blogging	`guykawasaki, scobleizer, chrisbrogan, darrenrowse, techcrunch`

Deep Dive into Key Communities A more granular examination of individual communities confirms the principle of **homophily**, where users cluster around shared interests, professions, or causes.

Community 1: Tech News & Social Media Marketing. This community, visualized in Figure 8a, is the largest cluster in the network. Its thematic focus was determined by examining its most central nodes. The most frequently mentioned users include `mashable`, a major digital media platform focused on technology and social media, and `tweetmeme`, an early and influential service that aggregated popular links on Twitter. The common thread among these leaders is the dissemination of technology news and social media marketing trends, suggesting that the conversation within this cluster is dominated by industry professionals, marketers, and tech enthusiasts.



(a) Community 1: Tech News & Social Media

(b) Community 2: E-commerce & Shopping

Fig. 8: Zoomed-in visualizations of the two largest communities, highlighting their internal structure and central nodes.

Community 2: E-commerce & Online Shopping. The second-largest cluster, shown in Figure 8b, is focused on online retail and consumerism. This interpretation is supported by its most prominent members, such as `etsy`,

a global marketplace for handmade and vintage items, and `shopittome`, a personal shopping service. Unlike the first community, which revolves around information flow, this group is centered on commercial activity, brand promotions, and consumer interests, forming a distinct conversational space for shoppers and online retailers.

Community 3: Breaking News & Mainstream Media. This large community is anchored by major news organizations. Its central nodes include the accounts of `cnnbrk` (CNN Breaking News), `nytimes` (The New York Times), and `breakingnews`. These nodes act as powerful information hubs, broadcasting current events to a wide audience. The dense connections around them likely represent public discussion and reaction to global and national news stories, separate from the more niche tech conversations.

Community 4: Tech Influencers & Blogging. While also technology-focused, this community is distinct from the Mashable cluster. It is organized around influential individuals rather than corporate media entities. Key members include `guykawasaki` (a venture capitalist and author), `scobleizer` (Robert Scoble, a prominent tech blogger), and `chrisbrogan` (a marketing consultant). The conversations here are likely more centered on personal branding, thought leadership, and opinion-driven analysis within the tech industry.

Overall, the community detection analysis effectively decomposed the giant component into a **"network of niches"**. The high modularity score and clear visual separation of clusters confirm that the network is not a single homogeneous conversation. Instead, it consists of several large, thematically focused communities alongside numerous smaller, specialized groups. This structure demonstrates how online social networks facilitate the formation of distinct conversational arenas, enabling both broad and highly specialized interactions to coexist.

3.5

Analysis of Link Weight Distribution

The weight of a directed edge in the mention graph represents the total number of times one user mentioned another, quantifying the strength or frequency of their interaction. To examine the overall distribution of these interaction strengths across the network, the link weight distribution was analyzed for the `twitter-small` dataset. This analysis was conducted using a Python script, provided in Appendix 1, which extracts edge weights from the network's edgelist, counts their frequencies, and plots the results on a log-log scale. This approach follows standard practice for characterizing interaction patterns in large social networks, where values often span several orders of magnitude.

The resulting distribution, shown in Figure 9, reveals a clear structural pattern. When plotted on logarithmic axes, the data points form an approximately straight line with a steep negative slope. This visual pattern suggests that as the link weight increases, its frequency decreases rapidly. In other words, repeated interactions between the same pair of users are relatively rare compared to single mentions. The linear appearance on a log-log scale is characteristic of a power-law distribution, indicating that the mention network exhibits scale-free properties. Such a distribution implies that while most connections in the network are weak and occur only once, a small number of pairs interact very frequently, forming strong ties that act as outliers in an otherwise sparsely connected system.

Empirical results from the analysis support this interpretation. The majority of links, 137,620 in total, have a weight of one, meaning that most user mentions occur only once. The number of links with a weight of two drops sharply to 12,584, and by the time the weight reaches five, only 309 links remain. This rapid decay illustrates the heavy-tailed nature of the distribution, where weak ties dominate the structure of the network and strong ties are statistically rare. Unlike a normal distribution, which would have a typical or average interaction strength, this network exhibits no characteristic scale. Its structure is instead defined by a vast number of weak, transient connections alongside a very small number of persistent, high-frequency interactions.

This pattern aligns with well-established findings in social network theory. The prevalence of weak ties reflects the open and transient nature of communication on platforms such as Twitter, where users often mention others only once, for instance in replies, reposts, or citations. These weak ties play a crucial role in facilitating information diffusion across otherwise distant parts of the network, as discussed in Granovetter's seminal work on the strength of weak ties (1973). Conversely, the few high-weight edges represent sustained exchanges between users, suggesting more stable and recurring social relationships. The coexistence of many weak ties and few strong ties is a hallmark

of large-scale social systems, supporting the interpretation that the mention graph follows a power-law, or heavy-tailed, distribution of interaction strengths.

The Python implementation of this analysis, detailed in Appendix 1, reads the edgelist file, computes the frequency of each unique link weight, and generates the log-log plot using the `matplotlib` library. The log-log representation enables effective visualization of the power-law behavior, emphasizing both the dominance of weak ties and the rarity of strong connections in the social structure.

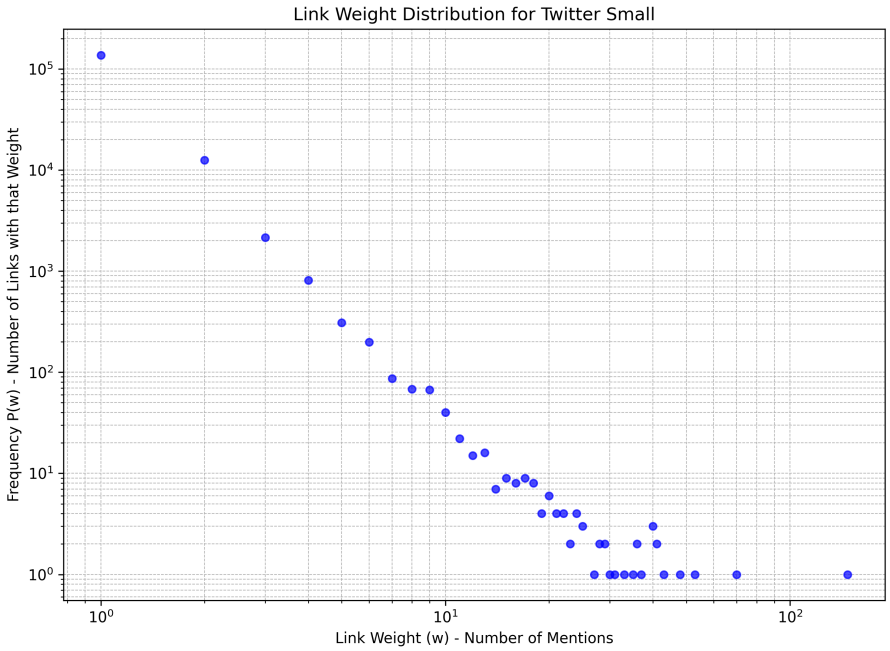


Fig. 9: Distribution of link weights on a log-log scale for the `twitter-small` mention graph. The x-axis represents the link weight (number of mentions), while the y-axis shows the frequency of links with that weight.

1 3.7

The Twitter interaction network was analyzed using a 10% uniform sample to manage its large scale, resulting in a subgraph of 796,852 nodes and 1,641,042 edges. Extrapolating from this sample, the full graph is estimated to contain approximately 8 million nodes and 16.4 million edges, as detailed in Table 9. The analysis reveals that the network exhibits the classic characteristics of a large-scale social network, namely properties of being scale-free and having a small-world structure.

Degree Distributions and Scale-Free Nature

The indegree and outdegree distributions, plotted on a log-log scale in Figure 10, show a clear linear trend. This is characteristic of a **power-law distribution**, a hallmark of scale-free networks. This heavy-tailed distribution indicates a significant inequality in node connectivity:

- **Indegree:** A small number of users (hubs) have an extremely high indegree, signifying they are the recipients of a vast number of interactions (e.g., popular figures, news organizations). The vast majority of users, however, have very low indegrees.
- **Outdegree:** Similarly, a few users exhibit a very high outdegree, indicating they initiate a large number of interactions, while most users are far less active.

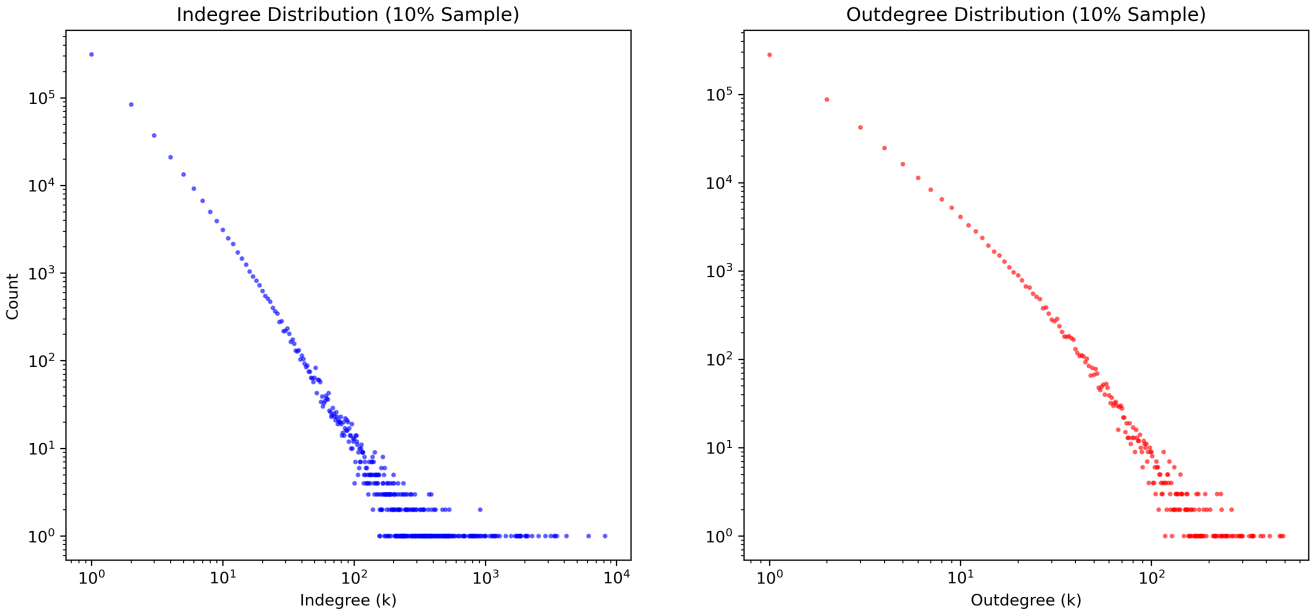


Fig. 10: Indegree (left) and Outdegree (right) distributions from a 10% sample of the network, plotted on a log-log scale. The linear trend is indicative of a power-law distribution.

Connectivity and Network Density

The network is **extremely sparse**, with a calculated density of approximately 2.58×10^{-6}. This is expected in large real-world networks where the number of actual connections is a tiny fraction of all potential connections. The analysis of connectivity reveals a **giant weakly connected component (WCC)** encompassing a majority of the nodes in the sample (636,051 nodes). This suggests that most users are reachable from one another if the direction of interactions is ignored. In contrast, the network is highly fragmented into millions of **strongly connected components (SCCs)**, with the largest found in the sample containing only 126,113 nodes. This indicates that mutual, reciprocal interaction paths are far less common than one-way information flow.

Table 9: Summary of key statistics for the Twitter interaction network, derived from a 10% sample.

Metric	**Value**
Number of Nodes (Estimated)	7,968,520
Number of Edges (Estimated)	16,410,420
Weakly Connected Components (Est.)	681,020
Giant WCC in sample	*636,051 nodes*
Strongly Connected Components (Est.)	6,457,950
Largest SCC in sample	*126,113 nodes*
Density (from sample)	2.58×10^{-6}
Avg. Clustering Coefficient (Approx.)	0.0522
Avg. Distance (Giant Comp., Approx.)	6.2173

Path Lengths and the Small-World Phenomenon

The distance distribution, shown in Figure 11, reveals that the network possesses the **small-world property**. The distribution of shortest path lengths within the giant component is centered around a clear peak, with an average path length of just **6.22**. This finding aligns with the well-known "six degrees of separation" concept, demonstrating that despite its vast size, any two users within the main component are connected by a surprisingly short chain of

interactions. This structure is critical for the rapid dissemination of information across the network. The average clustering coefficient of 0.0522, while modest, is significantly higher than that of a random graph of equivalent size, indicating the presence of local community structures where users tend to form tightly-knit groups.

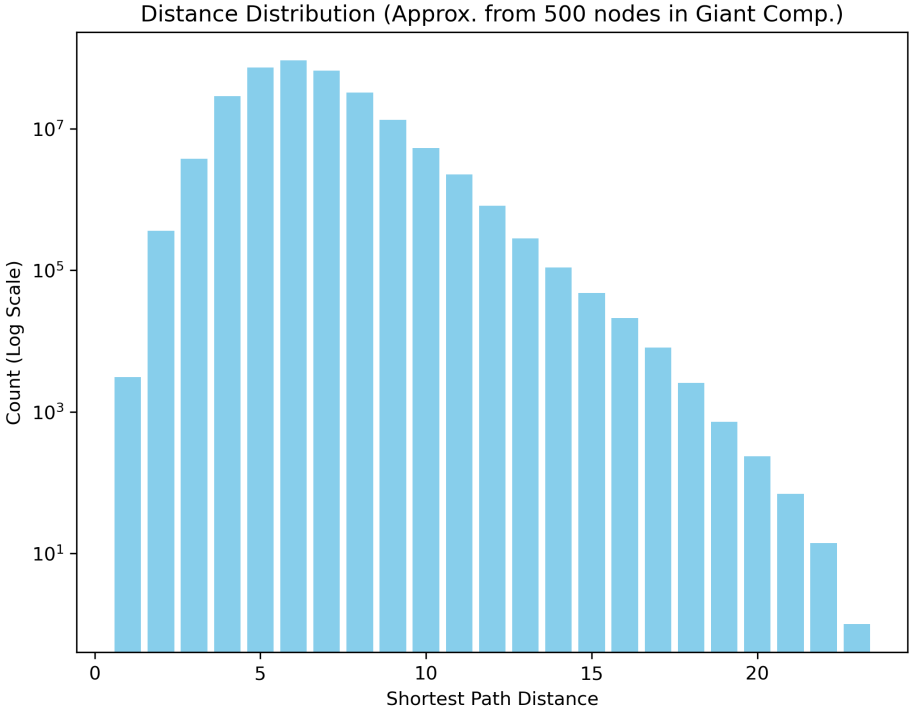


Fig. 11: Distribution of shortest path distances, approximated from 500 nodes in the giant component. The peak around 6-7 with a low average distance of 6.22 confirms the small-world property.

References

1. Frank W. Takes. Social network analysis for computer scientists – assignment 1. https://liacs.leidenuniv.nl/ takesfw/SNACS/snacs2025-assignment1.pdf, 2025.

Python Script for Mention Graph Extraction

```
import csv
import re
import os
import requests
from collections import defaultdict

def download_file(url, local_filename):
    """
    Downloads a file from a URL if it doesn't already exist locally.
    """
    if not os.path.exists(local_filename):
        print(f"Downloading {local_filename} from {url}...")
        try:
            with requests.get(url, stream=True) as r:
                r.raise_for_status()
                with open(local_filename, 'wb') as f:
                    for chunk in r.iter_content(chunk_size=8192):
                        f.write(chunk)
            print(f"Successfully downloaded {local_filename}")
        except requests.exceptions.RequestException as e:
            print(f"Error downloading {url}: {e}")
            return False
    else:
        print(f"{local_filename} already exists.")
    return True

def extract_mention_graph(input_tsv_path, output_csv_path):
    """
    Parses a Twitter TSV file to extract a weighted, directed mention graph
    and saves it as a CSV edge list.

    Args:
        input_tsv_path (str): The path to the input TSV file.
        output_csv_path (str): The path where the output CSV edge list will be saved.
    """
    print(f"\nProcessing {input_tsv_path} to generate mention graph...")

    # The graph is represented as a nested dictionary: {sender: {mentioned_user: count}}
    mention_graph = defaultdict(lambda: defaultdict(int))

    # This regex is designed to capture valid Twitter usernames mentioned in a tweet.
    username_regex = re.compile(r'@([A-Za-z0-9_]{1,15})\b')

    lines_processed = 0
    malformed_lines = 0

    try:
        with open(input_tsv_path, 'r', encoding='utf-8') as f:
            for line in f:
                lines_processed += 1

                parts = line.strip().split('\t')

                # --- Issue Handling: Malformed Lines ---
                if len(parts) != 3:
                    malformed_lines += 1
                    continue # Skip this line and move to the next one.

                timestamp, sender, content = parts

                # --- Issue Handling: Case Sensitivity ---
                sender = sender.lower()

                # Use the pre-compiled regex to find all mentions in the tweet content.
                mentions = username_regex.findall(content)

                if not mentions:
                    continue # Skip tweets with no mentions.

                for mentioned_user in mentions:
                    mentioned_user_lower = mentioned_user.lower()

                    # Increment the weight of the directed edge from sender -> mentioned_user.
                    mention_graph[sender][mentioned_user_lower] += 1

    except FileNotFoundError:
        print(f"Error: The file {input_tsv_path} was not found.")
        return
```

```
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return

    print(f"Finished processing. Total lines: {lines_processed}, Malformed lines skipped: {malformed_lines}")

    # --- Step 3: Output the adjacency list as a weighted edge list CSV ---
    print(f"Writing graph to {output_csv_path}...")
    edge_count = 0
    with open(output_csv_path, 'w', newline='', encoding='utf-8') as f:
        writer = csv.writer(f)
        writer.writerow(['Source', 'Target', 'Weight'])

        for sender, targets in mention_graph.items():
            for target, weight in targets.items():
                writer.writerow([sender, target, weight])
                edge_count += 1

    print(f"Successfully created edge list with {edge_count} unique edges.")


if __name__ == "__main__":
    files_to_process = {
        'twitter-small.tsv': 'https://liacs.leidenuniv.nl/~takesfw/SNACS/twitter-small.tsv',
        'twitter-larger.tsv': 'https://liacs.leidenuniv.nl/~takesfw/SNACS/twitter-larger.tsv'
    }

    for filename, url in files_to_process.items():
        if download_file(url, filename):
            output_filename = filename.replace('.tsv', '-edgelist.csv')
            extract_mention_graph(filename, output_filename)
```

Weight Distribution Analysis Code

```
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
import os

def analyze_weights(dataset_name):
    """
    Analyzes and plots the link weight distribution from an edgelist file.
    """
    edgelist_path = f'{dataset_name}-edgelist.csv'
    if not os.path.exists(edgelist_path):
        print(f"Error: {edgelist_path} not found. Please run the extraction script first.")
        return

    print(f"\n--- Analyzing Weight Distribution for {dataset_name} ---")

    # 1. Load the data and extract weights
    df = pd.read_csv(edgelist_path)
    weights = df['Weight']

    # 2. Count the frequency of each weight
    weight_counts = Counter(weights)

    # Sort weights for plotting
    sorted_weights = sorted(weight_counts.keys())
    counts = [weight_counts[w] for w in sorted_weights]

    print("Top 5 most frequent weights:")
    for w in sorted_weights[:5]:
        print(f"  - Weight {w}: Occurs {weight_counts[w]:,} times")

    # 3. Create the log-log plot
    plt.figure(figsize=(10, 7))
    plt.loglog(sorted_weights, counts, 'bo', markersize=5, alpha=0.7)
    plt.title(f'Link Weight Distribution for {dataset_name.replace("-", " ").title()}')
    plt.xlabel('Link Weight (w) - Number of Mentions')
    plt.ylabel('Frequency P(w) - Number of Links with that Weight')
    plt.grid(True, which="both", ls="--", linewidth=0.5)

    output_filename = f'{dataset_name}_weight_distribution.png'
    plt.savefig(output_filename, dpi=300)
    print(f"Saved weight distribution plot to {output_filename}")
```

```
if __name__ == '__main__':
    analyze_weights('twitter-small')
```