SDSC3001 - Assignment 1

Question 5

Loading the Graph

In [1]:
```
def load_graph(file_path):
    edges = []
    with open(file_path, "r") as file:
        for line in file:
            if line.startswith("#"):
                continue  # Skip comment lines
            parts = line.strip().split()
            if len(parts) == 2:
                from_node, to_node = map(int, parts)
                edges.append((from_node, to_node))
    return edges
```

In [2]:
```
file_path = "com-dblp.txt"
graph_edges = load_graph(file_path)
```

Calculate the normalized degree

In [3]:
```
def calculate_normalized_degrees(graph_edges):
    node_degrees = {}
    for from_node, to_node in graph_edges:
        if from_node not in node_degrees:
            node_degrees[from_node] = 0
        if to_node not in node_degrees:
            node_degrees[to_node] = 0
        node_degrees[from_node] += 1
        node_degrees[to_node] += 1

    total_degrees = sum(node_degrees.values())
    normalized_degrees = {
        node: degree / total_degrees for node, degree in node_degrees.items()
    }
    return normalized_degrees
```

In [4]:
```
normalized_degrees = calculate_normalized_degrees(graph_edges)
```

Simulate a random walk

In [5]:
```
import random


def simulate_random_walk(graph_edges, num_steps, seed=42):
    random.seed(seed)

    neighbors = {}
    for from_node, to_node in graph_edges:
        if from_node not in neighbors:
            neighbors[from_node] = []
        if to_node not in neighbors:
            neighbors[to_node] = []
        neighbors[from_node].append(to_node)
        neighbors[to_node].append(from_node)

    current_node = random.choice(list(neighbors.keys()))
    visit_counts = {node: 0 for node in neighbors.keys()}

    for _ in range(num_steps):
        visit_counts[current_node] += 1
        current_node = random.choice(neighbors[current_node])

    return visit_counts
```

In [6]:
```
def calculate_empirical_frequencies(visit_counts, num_steps):
    empirical_frequencies = {
        node: count / num_steps for node, count in visit_counts.items()
    }
    return empirical_frequencies
```

In [7]:
```
import numpy as np


def calculate_l1_distance(vector1, vector2):
    return np.sum(
        np.abs(np.array(list(vector1.values())) - np.array(list(vector2.values())
    )
```

In [8]:
```
diff_num_steps = [1_000_000, 10_000_000, 20_000_000, 30_000_000, 40_000_000, 50_
for num_steps in diff_num_steps:
    visit_counts = simulate_random_walk(graph_edges, num_steps)
    empirical_frequencies = calculate_empirical_frequencies(visit_counts, num_st
    l1_distance = calculate_l1_distance(empirical_frequencies, normalized_degree
    print(f"Number of steps: {num_steps}, L1 distance: {l1_distance:.3f}")
```

```
Number of steps: 1000000, L1 distance: 0.585
Number of steps: 10000000, L1 distance: 0.192
Number of steps: 20000000, L1 distance: 0.136
Number of steps: 30000000, L1 distance: 0.112
Number of steps: 40000000, L1 distance: 0.097
Number of steps: 50000000, L1 distance: 0.087
```

Summary of findings and Conclusion

My main findings from this question is to observe the computed ℓ_1-distance values as M increases, as introduced during the lecture.

The ℓ_1-distance measures how close the result of a random walk is to becoming steady state. This measure shows us how well the actual number of times we visit each node during the random walk matches up with what we'd expect based on how many connections each node has.

The link between $\mathbf{n}$, which is the normalized degree vector, and $\mathbf{f}$, which is the empirical frequency vector, comes from how random walks act on graphs.

Each part of $\mathbf{n}$, normalized degree vector, written as $n_v = \frac{d_v}{D}$, indicates the chance of picking a node v if we randomly choose an endpoint of a graph edge. This shows what we expect for visits if the random walk completely follows the graph's setup, only looking at how connected each node is.

On the other hand, each part of empirical frequency vector $\mathbf{f}$, written as $f_v = \frac{m_v}{M}$, shows how often we actually visit node v in a random walk with M steps. This tells us the real outcome of what happens during the random walk.

When M gets bigger, $\mathbf{f}$ should get closer to $\mathbf{n}$, making the ℓ_1-distance smaller. This means the random walk is becoming more like what we expect based on the degrees of the nodes.