Assignment 2: Parallel Graph Centrality Computation Using MPI

COL730

Due Date: 11:59PM, Oct 2, 2024

Objective

In this assignment, you will work with social graphs to compute centrality measures in parallel using MPI. Specifically, you will implement algorithms to compute degree centrality and betweenness centrality of nodes in the graph, and then combine these measures to identify the top k most influential nodes in the graph.

Problem Statement

You are given a directed graph representing social connections, where nodes are individuals and directed edges represent relationships. Your task is to compute two types of centrality measures:

1. **Degree Centrality**: This measure indicates how connected a node is within the graph. For a node u, the degree centrality is the sum of its incoming and outgoing edges.
2. **Betweenness Centrality**: This measure indicates the importance of a node in terms of its ability to connect other nodes via the shortest paths. For a node v, it is calculated as the sum of the fraction of all-pairs shortest paths that pass through v.

After computing these centrality measures, you will normalize the values and then combine them to compute a combined centrality score for each node. Finally, you will use this combined score to find the top k most influential nodes in the graph.

Instructions

Step 1: Reading the Graph

You will start by reading the graph from a binary file named `gplus_combined.dat`. This file contains the edges of the graph, where each edge is represented by two strings (source and destination). Since the file is large, use MPI I/O functions to read the file in parallel. Each MPI process should handle a portion of the file, and data should be combined correctly to form the complete graph.

Step 2: Computing Degree Centrality

Definition: The degree centrality for a node u is the number of direct connections it has. In mathematical terms, for a directed graph:

$$\text{Degree Centrality}(u) = \text{In-degree}(u) + \text{Out-degree}(u)$$

Implementation: Each MPI process should compute the degree centrality for a subset of nodes. The degree centrality for each node can be calculated independently.

Step 3: Computing Betweenness Centrality

Definition: Betweenness centrality for a node v is calculated by summing the fraction of all-pairs shortest paths that pass through v. It is mathematically defined as:

$$\text{Betweenness Centrality}(v) = \frac{\sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}}{(n-1)(n-2)}$$

where σ_{st} is the total number of shortest paths from node s to node t, n is the total number of nodes in the directed graph, and $\sigma_{st}(v)$ is the number of those paths that pass through v. Note that $\sigma_{st} \neq \sigma_{ts}$ in case of a directed graph, and if t is not reachable from s, then ignore the ordered pair (s, t).

Implementation: Use a parallel version of Dijkstra's algorithm to compute shortest paths and calculate betweenness centrality. Distribute nodes and paths across MPI processes to parallelize the computation.

Step 4: Normalizing Centrality Scores

After computing the raw degree and betweenness centrality scores, normalize them to bring the values into a comparable range. Normalization can be done using the following formula:

$$\text{Normalized Centrality}(u) = \frac{\text{Centrality}(u) - \min(\text{Centrality})}{\max(\text{Centrality}) - \min(\text{Centrality})}$$

This ensures that all centrality scores are between 0 and 1, making them easier to combine.

Step 5: Combining Centrality Scores

To find the top k most influential nodes, combine the normalized degree and betweenness centrality scores. You can assign the following weights to the centrality measures:

$$\text{Centrality}(u) = 0.6 \times \text{Normalized Degree Centrality}(u) + 0.4 \times \text{Normalized Betweenness Centrality}(u)$$

Here, a higher weight is given to degree centrality because the number of direct connections often has a stronger influence on a node's importance in a social graph.

Step 6: Finding the Top k Nodes

Sort the nodes based on their combined centrality scores in descending order. Then, select the top k nodes with the highest combined scores as the most influential nodes in the graph. For breaking ties, the node with the lexicographically smaller label is ranked higher.

Input Format

You can download the dataset for this assignment from the link here.

The input file contains an edge on every line, and an edge is denoted by the ids of two nodes (directed). This dataset consists of edge data from Google+. It contains 107,614 nodes and 13,673,453 edges. There are also 2 smaller graphs (only for testing, not to be evaluated): twitter.dat, which contains 475 nodes and 13,289 edges, and sample.dat for which the assignment is solved at the end of the document. "k" will be given as a command line argument. You can read the input data vertices/edges as strings as they may not fit into 32-bit integers.

Output Format

- **Degree Centrality File**: Write the degree centrality for each node to `degree_centrality.txt`. On each line, write the node id, followed by the degree centrality separated by a single space.
- **Betweenness Centrality File**: Write the betweenness centrality for each node to `betweenness_centrality.txt`. On each line, write the node id, followed by the betweenness centrality separated by a single space.
- **Top k Nodes File**: Write the top k nodes based on combined centrality scores to `topk_nodes.txt`. Write the node ids in decreasing order of influence, one id per line.

Evaluation Criteria

- **Correctness**: The accuracy of the computed centrality measures and top k nodes.
- **Performance**: Efficiency and scalability of the parallel implementation with different numbers of threads (1, 2, 4, 8, 16, 32, 40).
- **Report**: Clarity and depth of performance analysis.

Deliverables

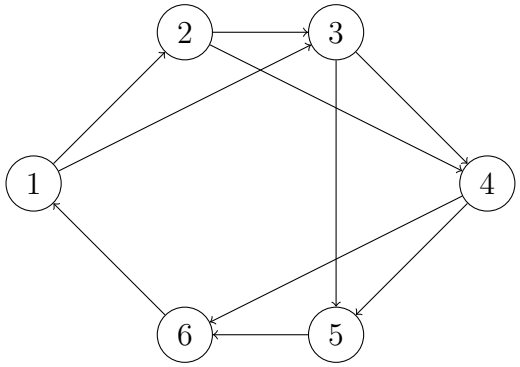
1. **Source Code**:
   - Submit a zip archive named `<entry_num>.zip` containing:
     - `main.cpp` and any other necessary `.cpp` and header files.
     - `Makefile` which compiles the code and generates an executable named `<entry_num>`.
     - The executable `<entry_num>` takes one argument "k".
2. **Performance Report**:
   - A `report.pdf` file containing runtime measurements for different numbers of MPI processes (1, 2, 4, 8, 16, 32, 40) for each of the provided graphs. Include a discussion of performance results and scalability analysis.

Solving an Example Graph

Let's manually compute the degree centrality, betweenness centrality, normalize them using float division, combine the scores, and find the top k nodes for the following graph:

Graph:

- Nodes: $\{1, 2, 3, 4, 5, 6\}$
- Edges: $\{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (6, 1)\}$



Step 1: Degree Centrality

For each node, we calculate the degree centrality as the sum of its in-degree and out-degree:

$$\begin{aligned}
\text{Degree Centrality}(1) &= 2 \text{ (out-degree)} + 1 \text{ (in-degree)} = 3 \\
\text{Degree Centrality}(2) &= 2 \text{ (out-degree)} + 1 \text{ (in-degree)} = 3 \\
\text{Degree Centrality}(3) &= 2 \text{ (out-degree)} + 2 \text{ (in-degree)} = 4 \\
\text{Degree Centrality}(4) &= 2 \text{ (out-degree)} + 2 \text{ (in-degree)} = 4 \\
\text{Degree Centrality}(5) &= 1 \text{ (out-degree)} + 2 \text{ (in-degree)} = 3 \\
\text{Degree Centrality}(6) &= 1 \text{ (out-degree)} + 2 \text{ (in-degree)} = 3
\end{aligned}$$

Step 2: Betweenness Centrality

For betweenness centrality, we calculate the shortest paths that pass through each node. Below are the betweenness centrality scores:

$$\begin{aligned}
&\text{Betweenness Centrality}(1) = 0.5 \\
&\text{Betweenness Centrality}(2) = 0.092 \\
&\text{Betweenness Centrality}(3) = 0.23 \\
&\text{Betweenness Centrality}(4) = 0.23 \\
&\text{Betweenness Centrality}(5) = 0.092 \\
&\text{Betweenness Centrality}(6) = 0.5
\end{aligned}$$

Note that all of the values have been normalised by (6-1)*(6-2) = 20

Step 3: Normalization

Degree Centrality Normalization:

$$\text{Normalized Degree Centrality}(u) = \frac{\text{Degree Centrality}(u) - \min(3,4)}{\max(3,4) - \min(3,4)}$$

$$\text{Normalized Degree Centrality}(1) = \frac{3-3}{4-3} = 0$$
$$\text{Normalized Degree Centrality}(2) = \frac{3-3}{4-3} = 0$$
$$\text{Normalized Degree Centrality}(3) = \frac{4-3}{4-3} = 1$$
$$\text{Normalized Degree Centrality}(4) = \frac{4-3}{4-3} = 1$$
$$\text{Normalized Degree Centrality}(5) = \frac{3-3}{4-3} = 0$$
$$\text{Normalized Degree Centrality}(6) = \frac{3-3}{4-3} = 0$$

Betweenness Centrality Normalization:

$$\text{Normalized Betweenness Centrality}(1) = 1$$
$$\text{Normalized Betweenness Centrality}(2) = 0$$
$$\text{Normalized Betweenness Centrality}(3) = 0.35$$
$$\text{Normalized Betweenness Centrality}(4) = 0.35$$
$$\text{Normalized Betweenness Centrality}(5) = 0$$
$$\text{Normalized Betweenness Centrality}(6) = 1$$

Combined Centrality Scores:

Using the weights $w_b = 0.4$ and $w_d = 0.6$, we compute the combined centrality score for each node:

$$C_{\text{combined}}(1) = 0.4 \times 1 + 0.6 \times 0 = 0.4$$
$$C_{\text{combined}}(2) = 0.4 \times 0 + 0.6 \times 0 = 0$$
$$C_{\text{combined}}(3) = 0.4 \times 0.35 + 0.6 \times 1 = 0.14 + 0.6 = 0.74$$
$$C_{\text{combined}}(4) = 0.4 \times 0.35 + 0.6 \times 1 = 0.14 + 0.6 = 0.74$$
$$C_{\text{combined}}(5) = 0.4 \times 0 + 0.6 \times 0 = 0$$
$$C_{\text{combined}}(6) = 0.4 \times 1 + 0.6 \times 0 = 0.4$$

Step 4: Top k Nodes

Finally, we sort the nodes based on the combined centrality scores to find the top k nodes.

If k = 2, nodes 3 and 4 are the most central nodes in the graph based on our combined centrality measure. If k=1, then node 3 is the most central node as the label is lexicographically smaller than 4.