Report on Optimisations for Assignment 2

Rishabh Kumar

March 17, 2025

1 Introduction

This report describes the performance and clarity improvements made to the `template.cpp` file. The code uses MPI to parallelize the computation of a degree-based centrality (or related centrality metric) across partial graph data. Below, we highlight the main areas of optimization.

2 Optimisations Implemented

2.1 Unordered Map for Adjacency and Color Storage

Instead of using `std::map`, the code employs `std::unordered_map<int, std::vector<int>>` to store adjacency information, and similarly for color lookups. An unordered map typically has average-case `O(1)` insertion and lookup, making it faster than a sorted map, especially for large graphs.

2.2 Collective Data Gathering

The code uses `MPI_Allgatherv` (or similar) to gather adjacency information and color data from all MPI processes in a single call. Fewer collective operations can help reduce overall MPI overhead.

2.3 Precomputed Color Index Lookup

After extracting the unique set of colors and sorting them, the code builds a `color_to_idx` table (`unordered_map<int,int>`). This eliminates repeated searches (e.g., `lower_bound`) in inner loops.

2.4 Memory Reservation

Wherever feasible, the code calls `.reserve(...)` on vectors to minimize repeated allocations during push-backs. This is applied in areas like adjacency construction and final result assembly.

2.5 Efficient Extraction of Top k Results

At the final step (on rank 0), the code sorts each color bucket of `(vertex, score)` pairs to retrieve the top k scores per color. If k is much smaller than the total size of a color bucket, one can switch to `nth_element` to partially select the top k elements in $\mathcal{O}(n)$ time, followed by a smaller sort of size k in $\mathcal{O}(k \log k)$ time. This is more efficient than a $\mathcal{O}(n \log n)$ full sort for large n.