CSCI 5451 Homework 3 Report

Benjamin Chu

November 2023

1 Implementation

For this problem, we are given the following process:

1. The first process reads data from the file. It then decomposes this data and distributes the data to the other processes.
2. Each process analyzes the non-local edges in its section of the graph.
3. Each process determines which other processes contain the non-local vertices corresponding to those edges.
4. The processes communicate to determine which processes need to send what data.
5. The processes perform transfer of non-local labels and updates of local labels until convergence.
6. The results are gathered to a single process which writes them to disk.

And for step 5, we are given the following algorithm:

1. Initialize by assigning a unique label to each node.
2. Until convergence (when no labels change):
   (a) Exchange non-local labels.
   (b) For each local node n:
   - Set the new label of n as the maximum (or minimum) label between its neighbors.

For this assignment, since the graphs have few edges relative to the number of possible edges, we store the graph with a sparse matrix representation. We use the following struct to store the graph:

```
typedef struct {
    int num_nodes, num_edges;
    int *counts;
    int *offsets;
    int *edges;
} graph_t;
```

The array `edges` stores all the edges of the graph. `counts` stores the number of edges for each vertex, and `offsets` determines where the edges for each vertex are stored in the `edges` array. Then, the jth edge of node i is stored at `edges[offsets[i]+j]`.

For the decomposition, a simple implementation would be to assign each process a range with an approximately equal number of points. However, in the given dataset, the lower indexed vertices tend to have much more edges than the higher indexed edges. In this case, decomposing this data this way could result in some processes having to handle significantly more edges than the others. We can balance the load by using the following steps:

1. Compute a target size for each range as the number of edges divided by the number of processes.
2. Iterate over the nodes of the graph, keeping a running total of the edge counts.
3. Each time the running total approaches the target size, create another range ending at the current index.

This produces ranges where the total number of edges in each range is approximately equal.

For steps 2-3, each process analyzes the non-local edges to determine what labels will be sent where. We do so in two passes over the edge list. On the first pass, each process counts its edges to determine how many edges need to be sent to each other process. Then, on the second pass, each process determines which edges will be sent to the other processes.

For step 4, each process scatters its data computed from steps 2-3 among the other processes. The data each process receives this way will determine what data it will be receiving in step 5.

During step 5, each process can access non-local labels by using a binary search to search its array of received labels, for a time complexity of $O(\log n)$. We can improve on this by saving an array of pointers alongside our edge array, where each of those pointers points to the memory location of the corresponding label. This way, the $O(\log n)$ time only occurs once during preprocessing, and it will take $O(1)$ time to access each label while running the algorithm.

2 Timing Results

Processes	1000 nodes		10000 nodes		100000 nodes		1000000 nodes	
	Steps 2-5	Step 5	Steps 2-5	Step 5	Steps 2-5	Step 5	Steps 2-5	Step 5
1	0.0006s	0.0003s	0.0055s	0.0023s	0.0494s	0.0223s	0.8214s	0.5547s
2	0.0008s	0.0002s	0.0074s	0.0017s	0.0981s	0.0218s	1.6083s	0.3272s
4	0.0007s	0.0002s	0.0064s	0.0014s	0.0797s	0.0182s	1.0675s	0.1615s
8	0.0021s	0.0005s	0.0089s	0.0022s	0.0445s	0.0084s	0.4682s	0.0784s
16	0.0056s	0.0014s	0.0081s	0.0018s	0.0285s	0.0056s	0.2280s	0.0419s