Project

In this project we test and analyze the behavior of a graphs CPL (Characteristic path length) and connectivity when certain edges are removed. We achieve that by calculating the shortest paths for every possible combination of i and j nodes and then removing the edge that is most frequently used.

Pseudocode for main function and data structures:

This is the main function:

```
runGraphs(currentGraph) {
        if(numOfNodes == 1) {return;}

        for (all the nodes) calcDijkstra(node);

        calcCPL;
        sortEdges;
        removeEdge;

        if(isConnected) {
            newGraph;
            newGraph.initGraph(currentGraph);

            runGraphs(newGraph);
        }else{
            if(numOfNodes == 2) {return;}

            newGraph1
            newGraph2
            newGraph1.initGraph(currentGraph);
            newGraph2.initGraph(currentGraph);

            runGraphs(newGraph1);
            runGraphs(newGraph2);
}}
```

Here we can see a function that calculates the shortest paths via a Dijkstra algorithm, the CPL, removing the most used edge, whether the graph is still connected, decides if the graph is to be split and the "next" graph/graphs is processed recursively.

Before diving in depth on how some of these methods work let's see the data structures that we use:

- A one-dimensional array of type (class Node) that represents all the nodes of the graph
- A two-dimensional adjacency array of type Boolean that shows the connections of every node to every other node
- A two-dimensional array of type int that keeps track of the use of every edge

Also, this is the class that represents the node:

```
class node {int node; int[]neighbours; double distance; boolean verified;}
```

The field "node" keeps the number that identifies the node, "neighbours" keeps the numbers of every node's neighbours and distance-verified are used in the Dijkstra calculations.

Algorithms in depth:

The methods that are worth mentioning and use some kind of algorithm are the "calcDijkstra", "sortEdges" and "isConnected".

1) "calcDijkstra":

```
calcDijkstra(start) {
   queue.add(starting node);
   while(!queue.isEmpty()) {
      currentNode = node closer to the given node;
      if(!currentNode.verified) {
         for (go through all the neighbours) {
            if (neighbour.verified) {
                if (neighbour.distance > currentNode.distance + 1) {
                   neighbour.distance = currentNode.distance + 1;}
                queue.add(neighbour);
         }}
         numOfVertices(currentNode);
         currentNode.verified = true;
 }}}
```

This is just a standard Dijkstra algorithm with a priority-queue which by itself has a complexity of $O((V + E)logV)$ where V stands for vertices/nodes and E for edges. There is also (marked in red) another algorithm nested in the while loop that finds which edges were used for the shortest path of the "start" node and the one that is verified immediately after.

This is the pseudocode:

```
numOfVertices(node) {
  while(node.distance != 0) {
    for(the neighbours of the node) {
       if (neighbour.verified && neighbour.distance == node.distance - 1) {
            edges[node][sNode]++; (This is the array mentioned in the data structures)
            edges[sNode][node]++;
            node = sNode;
            break;
}}}}
```

Now let's calculate this algorithms complexity:

Assuming that the while loop operates X times and the for loop K times the number of operation should be X*K*4 (4 operations in the for loop). Since the for-loop compares all the neighbours of the chosen node, k is the average degree of a node or $K = 2E/V$. Also, the

while loop operates until we reach the starting node thus it describes the shortest path from the node given or in general the CLP. This means that $X = (expected\ CPL) = log(V)/log(pV)$ where p is the probability of the current node being connected to some other or $p = degree/V$. The number of operations is:

$$op = \frac{\log(V)}{\log(\frac{2E}{V})} \times \frac{2E}{V} \times 4$$

And the complexity:

$$O\left(\frac{\log(V)}{\log\left(\frac{E}{V}\right)} \times \frac{E}{V}\right)$$

In conclusion the total complexity of calcDijkstra where numOfVertices is called V times is:

$$O\left((V + E)logV + V \times \left(\frac{\log(V)}{\log\left(\frac{E}{V}\right)} \times \frac{E}{V}\right)\right) => O\left((V + E)logV + \frac{\log(V)}{\log\left(\frac{E}{V}\right)} \times E\right)$$

2) "Sortedges" is just a merge sort of all the edges to find which is the most frequent, so it has a complexity of:

$$O(E \times log(E))$$

3) "Isconnected uses same exact process as "calcDijkstra" with of the calculation of the edges, so it has a complexity of:

$$O\big((V + E)logV\big)$$

To sum up, the cost of initializing the new graphs is:

1) Copying the two-dim array of connection thus $op = V^2$
2) Creating the new nodes $op = V \times avg\ degree(neighbours) = V \times \frac{2E}{V} = 2E$

Total complexity of the main function "rungraphs" is:

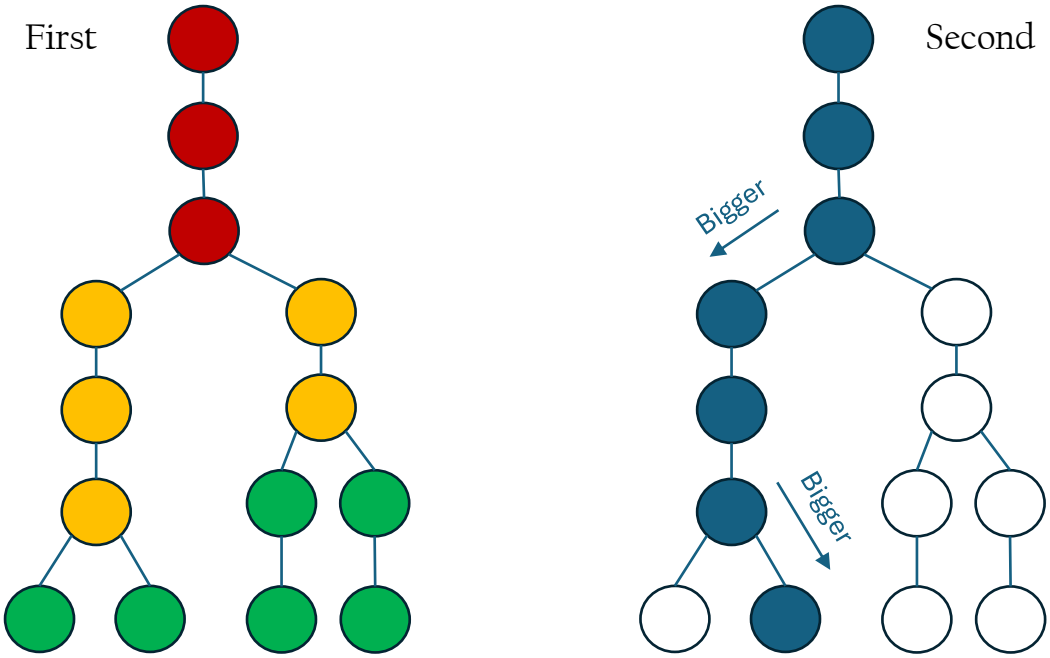
$$O\left\{\left((V + E)logV + \frac{\log(V)}{\log\left(\frac{E}{V}\right)} \times E\right) \times V + E \times log(E) + (V + E)logV + V^2 + E\right\}$$

Statistics and plots for the evolution of the program

We are provided with three tests and for every one of them we will show three plots, two describing the average CPL and one showing the largest sub-graph every time the graph is split. Now let's describe what the first two statistics show exactly:

1)First plot

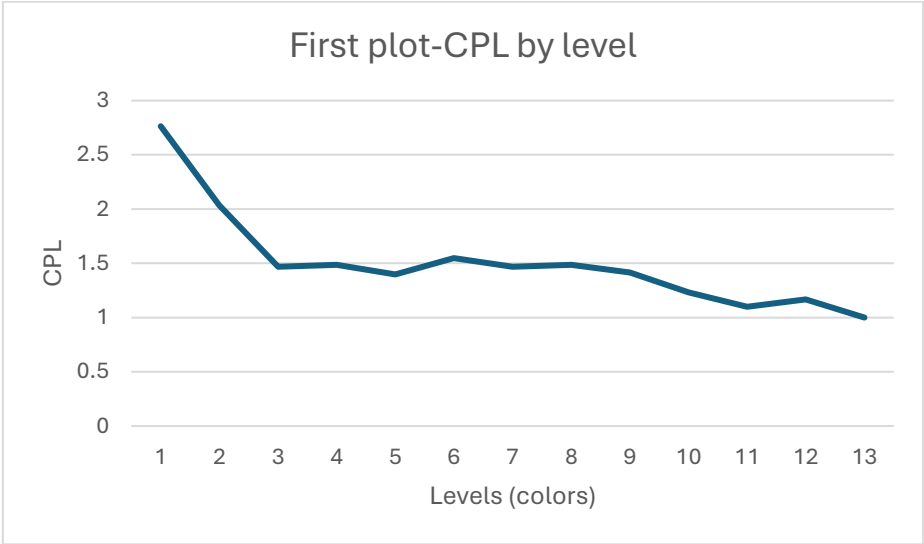
Every circle is a graph and as we go down the tree the new circles are graphs with the most used edge removed or the graph split in two. Every color represents a level, and the plot shows the average CPL of all the graphs of the same color as we go down.



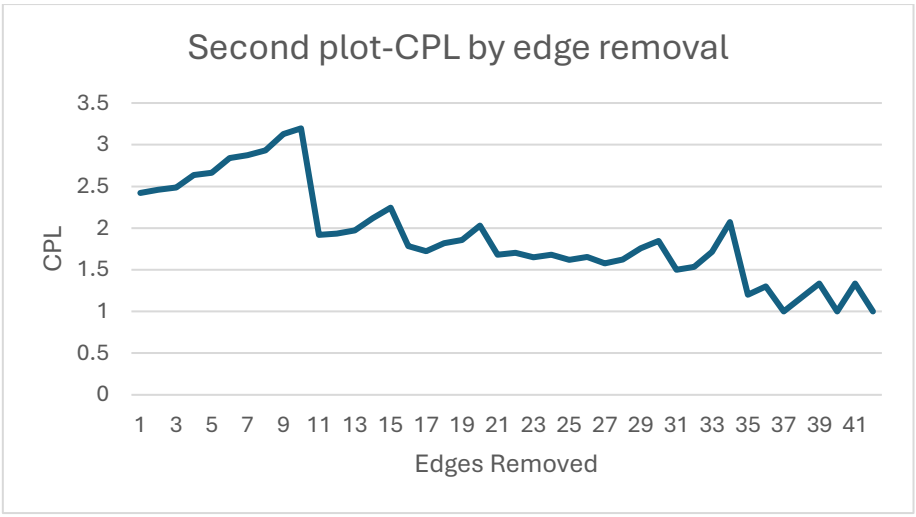
2)Second plot

It shows the progression of the CPL following the removal of edges and when the graph is to be split, we follow the bigger sub-graph.

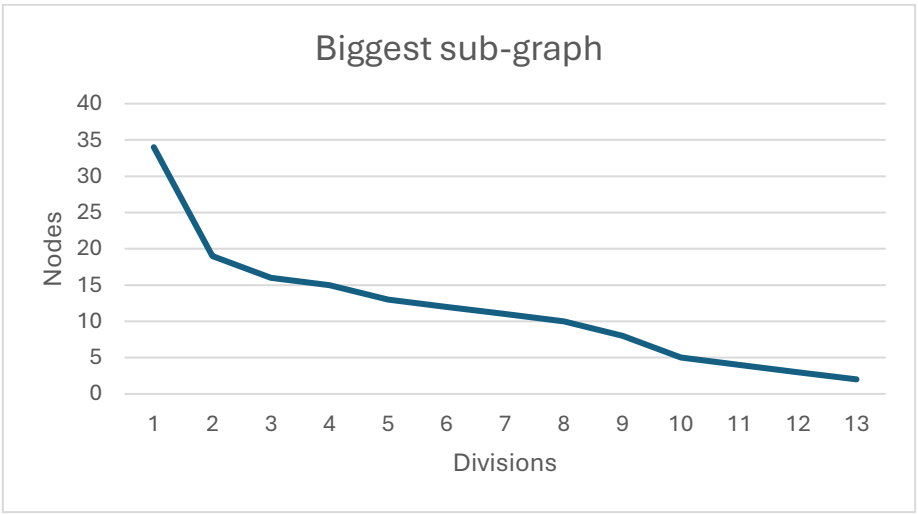
First test



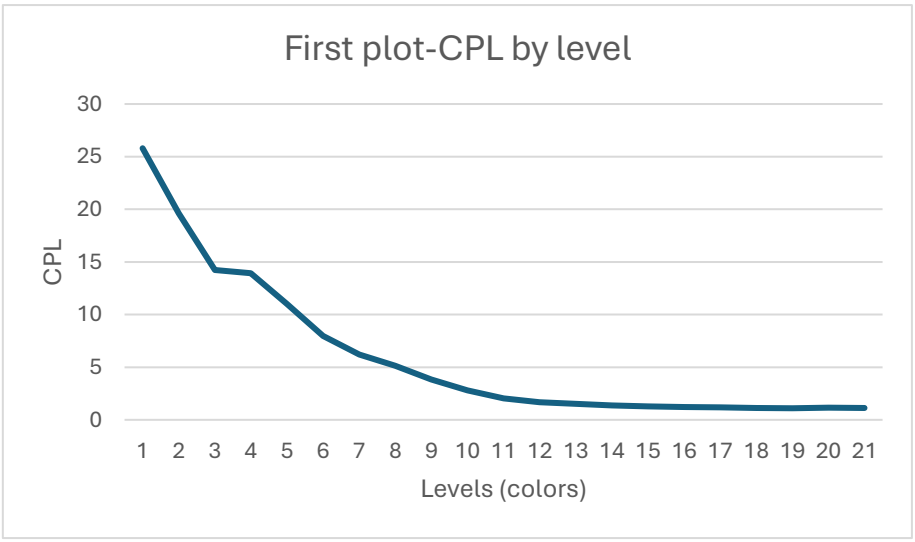
We see that the CPL steadily decreases as the graph is being dividing into smaller sub-graphs



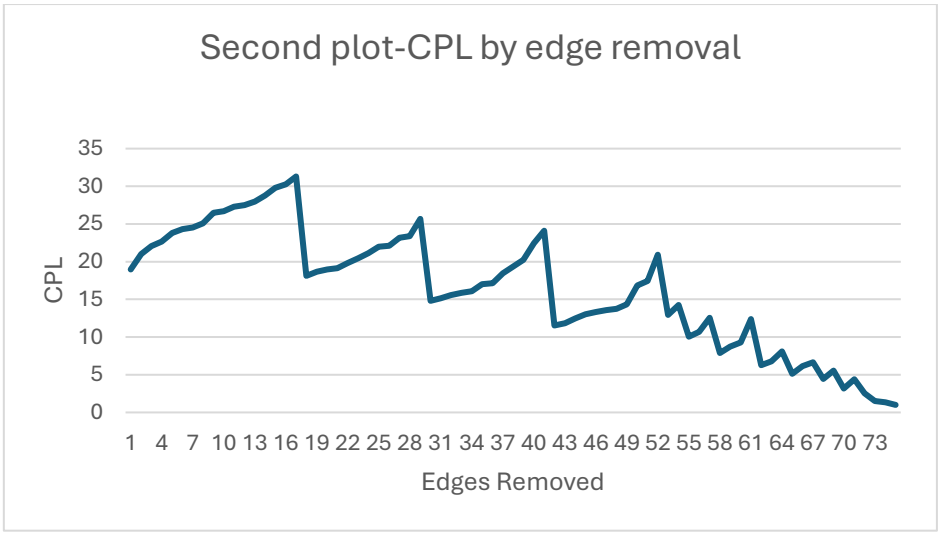
We observe that the CPL increases while edges are removed and the graph does not split but when it splits there is a sudden drop



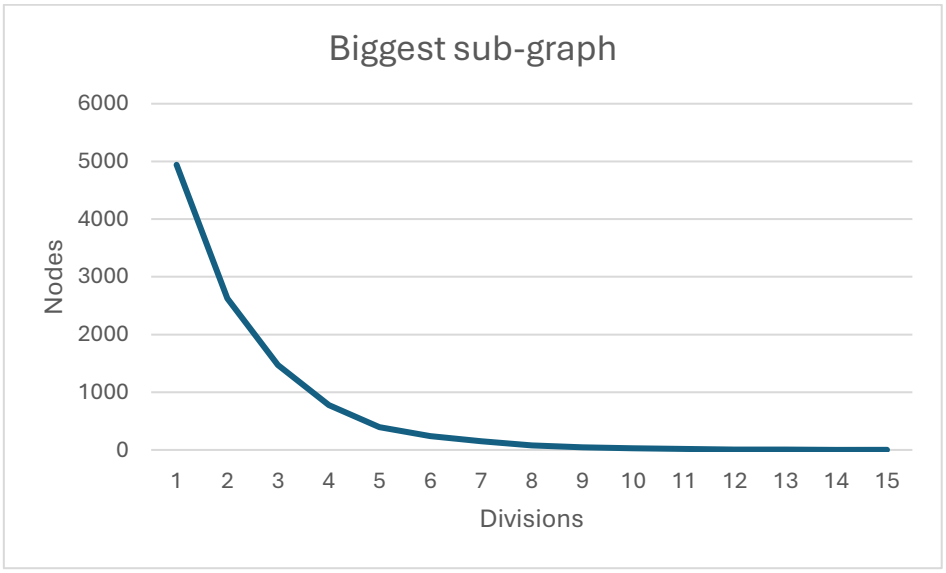
Second test



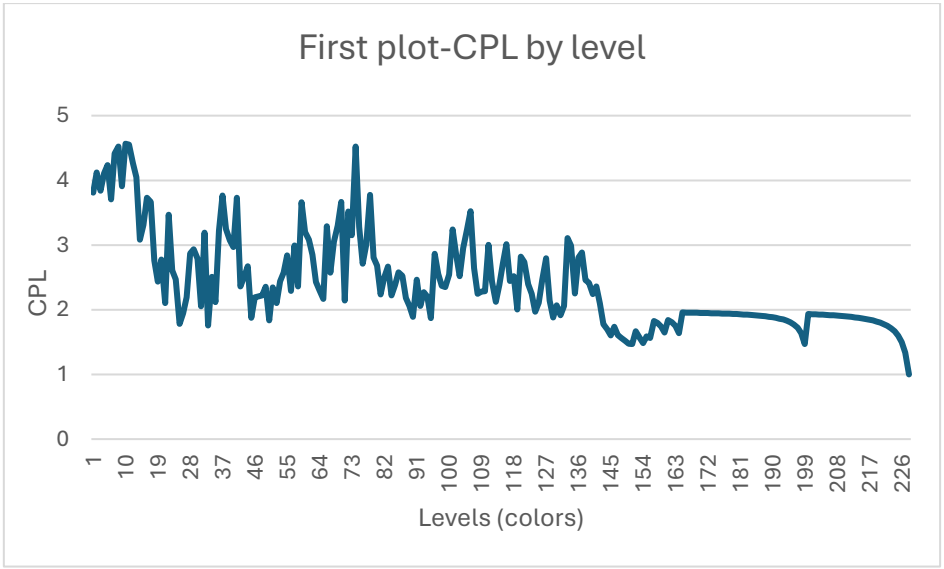
Similar to the first test.



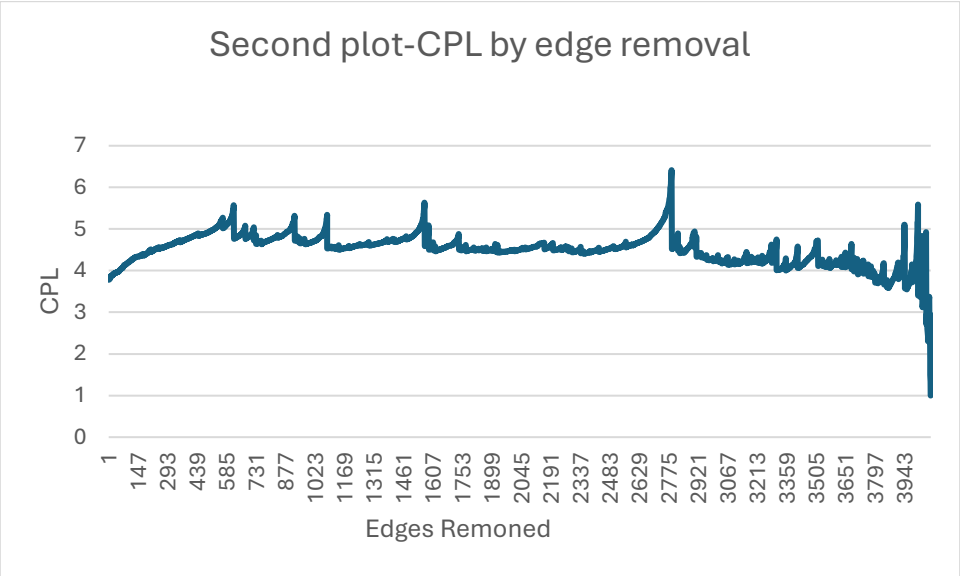
Similar to the first test again

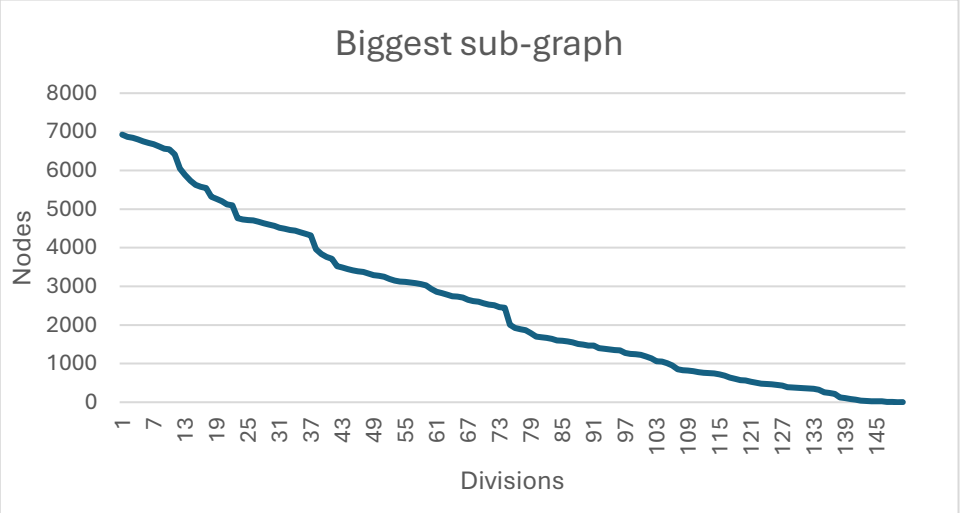


Third test

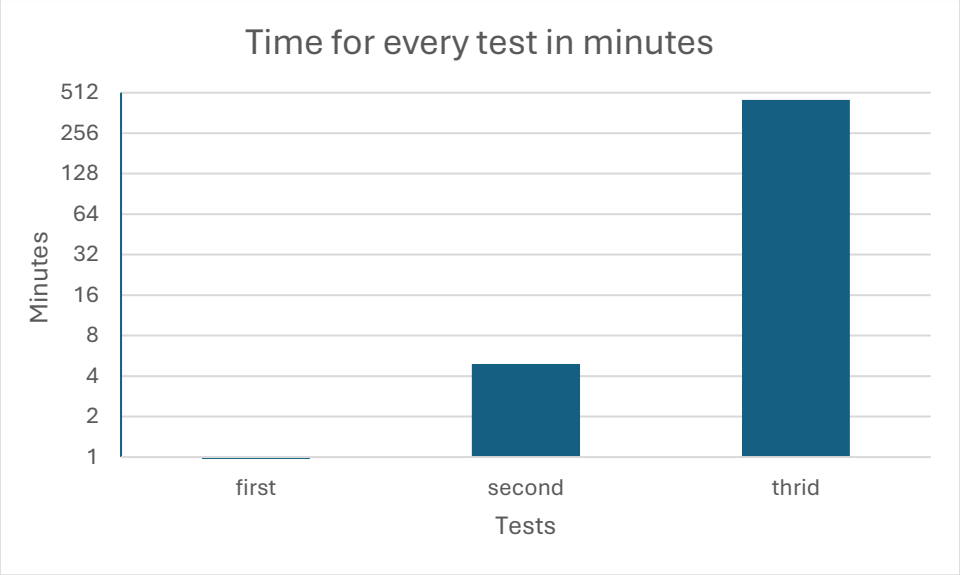


In this test it seems that we CPL decreases but not as steadily as the previous two tests. This probably happens because every time the graph slits the two subgraphs are not equal in size and the bigger one is almost the size of the original (we can see that in the "Biggest sub-graph" plot where the decrease is almost linear). This means that the sub-graph has almost as many nodes but might have fewer edges depending on how it is spilt and that drives the CPL temporarily up immediately after the division.





Time to complete the tests


Time for every test in minutes
512
256
128
64
32
16
8
4
2
1
Minutes
first
second
thrid
Tests