CS 170 Project Reflection

JaeHyun Park

3035075252

I came up with three different but similar algorithms to generate my outputs. The main idea behind my three algorithms is as follows: find the graph with maximum shortest path from a tree of graphs where the top of the tree is the original graph, and at each level down the tree either the number of vertices or edges decreases by one. In the tree at each level, removed vertex or edge is chosen from the shortest path of its parent graph; that is for instance if the shortest path of a parent graph is 0-3-5-7, you can either remove one of the nodes, {3, 5} or one of the edges, {(0,3), (3,5), (5,7)} as long as the new graph is connected after removal. But I found out that constructing the full tree took too long, so I decided to use heuristic variable(s) to minimize the size of the tree.

For my first algorithm, 'solve' function in solver_ver1.py, I remove edges first then vertices. For removing edges, I use one heuristic variable, HTC, which is the height of the tree to merge; for instance if HTC=10 and the input graph G has 50 nodes, the algorithm constructs the tree of graphs mentioned above of height 10 with G being the ancestor of all graphs in the tree. Then the algorithm merges the tree to a single graph with max shortest path, say H. Then the algorithm constructs another tree of graphs of height 10 with H being the ancestor of all graphs in the tree and so on until the sum of the height of the trees reach 50. Note that for my first algorithm, the tree and the heuristic are only used when removing edges. Now for removing vertices, I use the same approach but use for-loop and HTC is always set to 1.

For my second algorithm, 'solve' function in solver_ver2.py, I go with the same idea as the previous algorithm of repeatedly constructing the tree of height HTC and merging it into a single graph. But this time, the same HTC value is applied to both vertices and edges, and I get two graphs, one by removing edges first then vertices and the other one by removing vertices first then edges. Then I return the one that has a larger shortest path length.

My third algorithm, ‘solve’ function in solver_ver3.py, is identical to my second algorithm except the fact that this time it uses two heuristic values, HTC_VERTICAL and HTC_HORIZONTAL where HTC_VERTICAL constrains the height of a tree while HTC_HORIZONTAL constraining the tree’s width.

As far as the performance goes, my second algorithm with HTC=3 to 5 (HTC=5 for small, HTC=4 for medium, HTC=3 for large) had the best results overall but my third algorithm with HTC=4 to 6 (HTC=6 for small, HTC=5 for medium, HTC=4 for large) also produced optimal paths for numerous input graphs. One peculiar thing to note is that larger HTC value did not necessarily produce a better result for all input graphs.

I think this is a decent approach because the shortest path length of a graph can increase only when either vertices or edges from the shortest path are removed so the previous shortest path no longer exists. Also by using heuristics I was able to successfully run my algorithms on my local machines since the use of heuristics constrained the size of a tree and reduced the running time significantly. But I do have a regret about this project; I would have been able to produce better results if I mixed the instances of removing edges and vertices.

For the project, I used two of my laptops, M1 Macbook Air and 2019 Thinkpad. I mainly used M1 Macbook Air to generate my outputs because it was more responsive and lasted a longer battery than my other laptop. I used 2019 Thinkpad to write code and gather outputs basically for non-intensive tasks.