Apurba Pokharel
11627243

The overall structure

The code will be divided into classes.

1. Graph

- Generates the graph, randomly or as per requirement 1.
- Returns the flow and capacity of each edge.
- Can plot the graph if visualization is needed.
- The graph code is referred from Dr. Russel's tutorial with slight changes as necessary.

2. Agent

- The brains of the entire operation.
- Has function for the heuristic function.
- Has function to optimize the flow of a path selected via heuristic value.
- A final method to calculate the optimized flow of the network.

The Pseudocodes are:

1. For the heuristic function

a. The heuristic function for this problem or in particular for any path associated with the graph in the problem will be, the minimum value, which is the difference between flow in edges in a path. The difference value will be the difference between the maximum allowable flow and the current flow rate through an edge.
b. For example: For path between nodes A – B – C – D, the heuristic function will return the minimum value which is the difference between the max and current flow in the edges between A-B, B-C, and C-D.

The code may end up looking something like this:

```
def heuristicFunction(self):
    # self.graph.path holds the all paths from sink to source
    for p in self.graph.path:
        minn = sys.maxsize
        # once inside the path iterate over each edges i.e the connectors between
        nodes
        for u,v in p:
            search_tuple = (u,v)
            result_tuple = self.graph.getEdgeFlowAndCapacity(search_tuple)
            remaining_flow = result_tuple[1] - result_tuple[0]
```

```
            if remaining_flow == 0:
                minn = 0
                break
            else:
                minn = min(minn, remaining_flow)
        heuristic_value_list.append(minn)
```

Once this function operates we will have the heuristic value (min flow difference) for each path.
Once this is computed we can move into the children generation.

2. For the successor function

a. Once the heuristic value for each path between the source and sink is known the rest will come easily.
b. The path with the highest heuristic value will be selected for expansion.
c. In the expansion process, each path edges will increase its flow by the heuristic value. Since, the heuristic value will be the minimum difference value, the law of conservation will be preserved as well as all the edges will have flow less than or equal to it's maximum flow.
d. Once the path's flow has been increased the children nodes are generated.
e. For the children node generation, the path with heuristic value 0 can be discarded and not considered and the rest of the paths will be passed to the function in 1 and the process will repeat until there is no more children available for expansion. Meaning all the children path will have an heuristic of 0.

The code may end up looking something like this

```
# max_heuristic_child_list is the path that has been selected for expansion
# increase_factor is the heuristic value for this path
def optimizeHeuristic(self, max_heuristic_child_list, increase_factor):
        for u, v in max_heuristic_child_list:
            search_tuple = (u, v)
            # the current flow is stored in index 0 and the max capacity in index 1
            result_tuple = self.graph.getEdgeFlowAndCapacity(search_tuple)
            new_flow = result_tuple[0] + increase_factor
            self.graph.updateFlow(u,v, new_flow, result_tuple[1])
```

3. The overall pseudocode

1. Generate the graph as needed.

2. From the sink nodes generate all paths that can be taken to reach back to the source node.

3. Once all these paths have been generated, iterate over each path.

    a. For each path, pass this path into the heuristic function.
    b. The heuristic function will iterate over each nodes in this path and return the heuristic value for the associated path.

4. Once heuristic for each path has been calculated, find the path with the maximim heuristic value. Select this path for expansion.

5. Repeat the steps in 2,3,4 until no new path can be generated. Meaning the heuristic value for the children node will be zero.

The code may end up looking like this

passing source and sink can be skipped as the source and sink nodes will always be 0 and the sink will be the (max no of nodes – 1).

```
def optimize(self, source, sink):

    while(1):
        # 1. compute heuristic function on each path
        res = self.heuristicFunction()

        # 3. optimize the flow for the chosen path
        if res == ():
            print("Done optimizing")
            break
        else:
            max_heuristic_child_list = res[0]
            heuristic = res[1]
            self.optimizeHeuristic(max_heuristic_child_list, heuristic)
```

All of the codes and algorithms are suspectable to change. This is the early thought process and may change with further testing and refinement.