PA3_report

1. Data Structures and algorithm I used
   a. Where I tested my code
      All of my code is run on the EDA Union Lab.
   b. Data Structures
      I used one-dimensional arrays (vector<int>) to store capacity, demand, and history (used for A* algorithm) for the 3D grid; whereas, nets are stored in a vector <Net> array. The size of these arrays is numLayers * xSize * ySize. Flattening the 3D coordinates (layer, x, y) into a 1D index improves cache locality compared to a 3D vector vector. We create an adjacency list vector<vector<Edge>> to represent the routing graph, where each vertex corresponds to a GCell. I used priority_queue (builtin heap) with a greater comparator (builtin > comparator) to implement the Min-Heap for the A* algorithm.
   c. Algorithm
      I apply A* algorithm instead of classical Dijkstra's Algorithm since we view efficiency important and also have to try our best to find the minimum overflow solution, though the function name in the code is still Dijkstra. Thus, the heuristic A* algorithm is our best option for the problem and get the best approximation. While we should also try to lower the length, we take f(n) = g(n) + h(n), where g(n) is past cost = current_cost + edge_basecost + node_weight. Here, node_weight = 1 + 50 * $2^{\min\{overflow,\ 20\}}$ basic_overflow_weight + current_history penalty. Moreover, h(n) is our heuristic cost, representing the Manhattan distance.
      Furthermore, I combine the A* algorithm along with the "Rip-up and Re-route" algorithm. During the Rip-up algorithm, we are able to record how clogged the current node is, if a GCell overflowed, then we add HISTORY_PUNISHMENT to history array every time.
2. My findings
   a. Our algorithm is extremely sensitive to the parameters we set, such as OVERFLOW_WEIGHT (30000), HISTORY_PUNISHMENT (50000) and the iterating times (50). Note that the numbers I give is the best one I found.