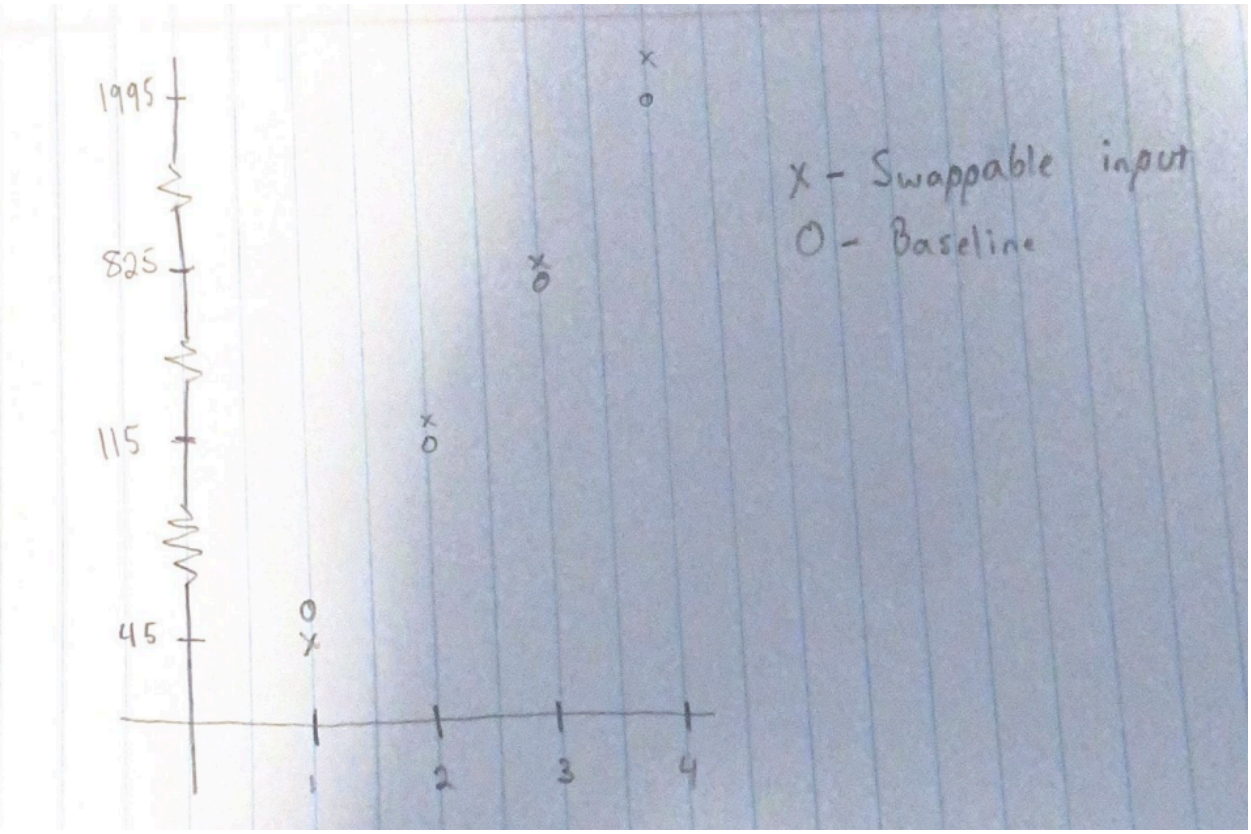
Theeban Kumaresan kumare15

2. A paper plot of the results from the four test files (using the value of W given in the test files) for the two routing styles (baseline and swappable inputs). The total number of routing segments used to route a design is a key metric in routing, as it is closely tied to performance and power consumption. Report the total number of routing segments used for each test file, for both routing styles.

Test #	BaseLine Segments	Swappable Segments
1	47	45
2	115	117
3	823	825
4	1994	1997



3. Report two items for each circuit, in table format, for both routing styles (baseline and swappable inputs): the smallest number of tracks/channel (W) that your program could successfully route each test circuit in; and, the total number of used routing wire segments (when W is minimum). That is, your program should be capable of varying the number of tracks per channel, and you are required to find the smallest number of tracks per channel that your program will successfully route the circuits in. Innovate to reduce W and minimize the number of routing segments used. Explain any optimizations you explored.

Test #	BaseLine minW	BaseLine # of Routing Segments	Swappable minW	Swappable # of Routing Segments
1	4	47	3	45
2	4	116	4	117
3	5	850	5	839
4	7	2011	7	2003

I had encountered one error where the router was exploring every single path multiple times, even those that it had already checked leading to an infinite loop in some cases or an extremely long run time for small test files. To circumvent this issue, I used a HashSet to prevent the router from "discovering" already discovered routes.

When implementing the Swappable input portion, I explored one way of swapping inputs such as always swapping the load pin so it would be as close as possible to the driver pin. While in theory this would result in the shortest path from the driver block to the load block, there were times when this would ultimately result in another connection having to take a much longer route or possibly not be routed at all. I ultimately decided on a technique where on the first connection of a block, I would calculate all the possible values for routing segments and find the lowest. From here, I would orient that set of pins for the rest of the routing in the test circuit. While this could have been optimized even further since there would be two possible orientations which might result in a lower total number of segments, the average run time would increase extremely and taking that into consideration, I decided against it.

4. In one paragraph, discuss the impact of swappable inputs on W and the number of used routing segments.

Generally, with swappable inputs, the router is able to take a shorter route as instead of going all the way around the block to get to the desired load pin, the router can take one segment less and only go one hop. Due to this, the number of used routing segments will decrease which will also decrease the needed W value in most cases as that extra track will be open for another connection.

5. Hand in a two-page description of the flow of your software, describing the major routines and data structures, and how they interact. Where you were faced with choices in your implementation of the algorithm, indicate what choices you made, and why.

BASELINE IMPLEMENTATION

The main flow of the software is as follows: First, the input file is read into the router.cpp file and placed inside the Grid class with i_connections storing all the connections, N the number of logic blocks and W storing the number of wires per track. From here, an adjacency matrix is created to store all the possible connections between every switch block. Afterwards, the main maze router function is invoked resulting in the program iterating over the i_connections vector and attempting to place every connection. If the connection is unsuccessful, the router removes the connection, places it at the start of the connection list and tries again, invoking the rip up and reroute segment. Once every connection is placed, the adjacency matrix is updated storing every connection between every switch block.

The main function attempting each connection is called processConnection. This function follows through the main maze routing algorithm as outlined in the lecture slides. Once this algorithm finds the load pin wire, it completes the traceback phase where it "takes" each wire segment it used to arrive there. This is done by a function labelled takeFirstTrackSegment which is responsible for labelling the track segment in the adjacency Matrix so no further connections can use that wire.

The major data structure being used in my implementation is a Graph data structure which represents the connections between each switch block. To utilize this graph data structure, an adjacency matrix, labelled adjacencyMatrix, is used to keep track of all the connections between each node, and is implemented using a 2d array of vectors of chars. Each row and col represent one switch block and a calculation is done to convert the X,Y coordinates into a singular integer value to represent a switch block. Each char in the vector of chars for each adjacency matrix cell is used to represent each wire connecting each switch block. When choosing a Graph as the key data structure, an array was considered to keep track of each switch block and its connections. However, if this had been chosen, it would have increased the level of complexity greatly when it came to the implementation since each cell in the array would have to keep track of the wire segment as well as the block it is attached to.

Another key data structure being used in this implementation is another adjacency matrix this time labelled costAdjacencyMatrix. This is set up exactly like the adjacencyMatrix, however instead of chars, ints are stored to keep track of the cost of getting to that segment. I elected to store W copies of each connection as since we are using a planar switch block architecture, it is impossible to switch between wire tracks and therefore, each wire segment will have its own respective cost.

SWITCHABLE INPUT IMPLEMENTATION

When implementing the switchable input implementation, I made a design choice to improve the efficiency of the router in terms of performance speed at a cost of increasing the wire segments used. When creating an initial connection from block (0,0) to block (1,1) for example, I decided to find the shortest cost based on each possible combination of inputs. Going forward, I kept that block at the same orientation unless there was a failed connection, in which case I reset it back to its original and tried the routing algorithm from the beginning.

This was accomplished by placing a modified processConnection function inside a wrapper function which would try making a connection for every possible input combination. From here, it would decide which was the lowest in terms of track segment then backtrack from the load to the driver, "taking" each wire segment as it went.

In order to keep track of the orientation, I first numbered each block from 0 to (N*N)-1. From here, I then created a vector which would store the orientation of the block with the 0th index representing the down direction or how the 1 connection is labelled on the a1 paper, 1st index representing the left direction or how the 2 connection is labelled on the a1 paper and finally, the 2nd index representing the up direction or how the 3 connection is labelled on the a1 paper.

It is important to note, that during each run of the modified processConnection function, between each invocation, all the cells of the adjacency matrix that were modified in the previous run are reset, in order for the function to work correctly.