The American University in Cairo

**School of Sciences and Engineering**

CSCE3304-01, CSCE3304-02

Spring 2018

Project 2 Proposal

**Ali Khaled, Karim Abdel Hamid, John Sourour**

**900140499, 900143463, 900140842**

**Under the supervision of Mohamed Shalan**

## **Division of Labor**

We divided our team to work on the parts equally. We still worked together as much as possible to aid and understand each others’ parts.

* John developed the Parser, with some help from Karim
* Ali focused on the research
* Karim wrote the router.

## **Running**

To view sample output, check output.log. To run the program, simply run ***make***, and then ***./router***

## **Router**

We built a very simple example of the router, using a 2D Grid. First, we add all start points, end points, and blockers. Next, we traverse the grid from the starting point. From each cell, we visit four neighboring cells to the top, bottom, left, and right. All newly added cells are added to a queue. At each iteration of a while loop, we pop an element of the queue and visit its four neighbors. This is done until a timeout elapses (to make sure it doesn’t count to forever, we may remove this in the future, but are using it to ensure we don’t have an endless loop for debugging purposes), we find the exit, or we are trapped. Then, we run another function that calculates the path back from the end to the start. It does this by checking the value of each node. It goes to the lowest adjacent node until it finds the start. It pushes each of these nodes to a queue, which is used for printing the final path. We also built a python router, which we used when considering adopting a python parser.

Here is some sample output. In this example, ss represents the start, ## represents a blocker (other wires, etc), and ee represents the end. the numbers represent the distance froms the start. Then, in the next example, \_ indicates a blank space, s indicates the start, # indicates a blocker, x indicates the path, and e indicates the end. This can be generated from making and running the executable.

3 2 1 2 3 4 5 6 7 8

2 1 ss 1 2 3 4 5 6 7

3 2 1 2 3 4 5 6 7 8

4 3 2 3 4 5 6 7 8 9

5 4 3 4 ## ## ## 8 9 10

6 5 4 5 ## \_\_ ee 9 10 \_\_

7 6 5 6 ## \_\_ \_\_ \_\_ \_\_ \_\_

8 7 6 7 8 9 \_\_ \_\_ \_\_ \_\_

Found End! Tacking...

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

\_ \_ s \_ \_ \_ \_ \_ \_ \_

\_ \_ x \_ \_ \_ \_ \_ \_ \_

\_ \_ x x x x x x \_ \_

\_ \_ \_ \_ # # # x \_ \_

\_ \_ \_ \_ # \_ e x \_ \_

\_ \_ \_ \_ # \_ \_ \_ \_ \_

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

## **Parsing**

We built two different parsers to extract the needed information for the routing, a LEF parser and the DEF parser. The LEF Parser is used to extract all the gates in the library, their widths and heights as well as the placement of their respective pins.

The DEF parser is composed of several classes. A DefComponent, a DefPin, a DefTracks, a Routed and a DefNet (including special nets) that contains the information needed for each field in the DEF file. Each class is responsible to extract its own info as well as its properties. A print function outputs the entire file’s equivalent content. This is simply used as an example, and we can access any data on the run.

For sample output, please check the attached file, output.log, as the output is too long to be placed here.

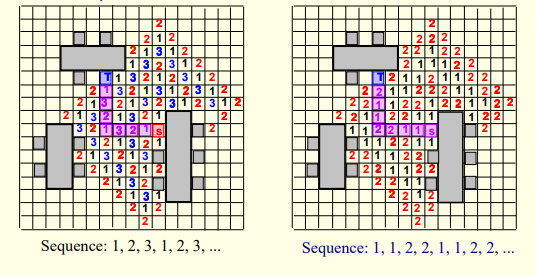
## **Research**

The lee’s maze algorithm is a robust algorithm used for routing digital circuits but usually requires significant amounts of memory and time , both of which can be scarce or expensive , but after doing some quick research it seems there are a couple of ways to reduce space and time complexity.

## Space Complexity

As it says in the Reference given for the project , we can create a small coding scheme for routing , we see that Akers developed this in 1967 where we constrict the grid labelling to 2-3 bits instead of an incrementing integer the farther we go from a certain pin . in Akers case he used the sequencing 0,0,1,1,0,0,1,1….etc to find the best path by filling the grid with this sequence and also backtracking with this sequence when finding the route. In our case we can use a simple 2-bit scheme to signify certain states of each square on the grid.

Our sequence would be the same as Aker’s but with the added 2 and 3 for labeling already present routes (blocked) and free squares (empty).



## Time Complexity

In order to reduce runtime we should expand lee’s algorithm to use BFS and DFS at the same time , which was proposed by Soukup’s algorithm (1978). This algorithm uses depth first search from the source to the target .once reached , we use BFS to maneuver around obstacles until it reached the source. The soukup algorithm is said to be 10-50 times faster than lee’s algorithm yet it almost uses the same memory.

Another method is using limited search areas and can be done with three methods

• Starting point selection: Choosing a point away from the center of the grid as the starting point.

• Double fan-out: create a circular area around both source and target cell in order to limit searching to these 2 circumferences

• Framing: Search inside a bounding box that contains both the source and target cells by 20% which should be able to limit searching while simultaneously giving the route some room for shortest path while avoiding any obstacles

