CPEN 513 – Assignment 1

## Introduction

This report describes an implementation of the Lee-Moore and A\* routing algorithms. The code is available at <https://github.com/lucylufei/CPEN513-assn1>

## General Implementation

The application begins by reading in the .infile and displaying the routing problem. The Lee-Moore routing algorithm is implemented in lee\_moore.py, and the A\* routing algorithm is implemented in astar.py. The program runs on one wire at a time, iterating through each of the sinks for the wire before moving to the next wire. Every step is displayed graphically, including the expansion numbers for Lee-Moore and the Manhattan distance from A\*. An alert is optionally displayed at the end, indicating the number of wires and number of pins successfully connected.

## Multiple Sinks

Multiple sinks are handled in the application in sequential order, one sink at a time. The application begins by randomly re-sorting the order of the pins after selecting the first pin as the source, then progressively attempts to route each sink to the source. Once a sink has been successfully connected to the source, the entire path, including the sink, can then be considered as part of the source.

In the Lee-Moore implementation, the expansion continues only until it reaches a cell on a previously established path between the source and another sink in the wire or the original source itself. In the A\* implementation, the Manhattan distance calculated at each step is the shortest distance to any cell that has already been connected to the source. This ensures that multiple sinks for the same wire are able to share their paths.

The application currently does not consider other sinks of the same wire as possible sources if the sink hasn’t been successfully connected to the source. For example, while routing the first sink, it may encounter another pin that has not been routed yet. In this situation, the application considers the location of the unrouted pin to be an obstacle even if it is part of the same wire. This implies that if a sink cannot be routed to its source, there will not be a cluster of sinks connected to each other.

## Optimizations

Two optimizations are made to improve the number of segments that can be successfully routed.

### Randomized Ordering

Firstly, the program begins by randomly sorting the wires to be routed. For the selected wire, the first pin is assumed to be the source and the remainder pins are re-ordered randomly. This heuristic allows the program to test various priorities for each wire and pin to maximize the number of successfully connected segments. This could potentially be improved by allowing the program to route pins from various wires out of order, without having to complete a wire before moving to the next.

### Tracking Difficult Wires

As suggested, the program tracks wires which are difficult to route. The routing runs multiple iterations in order to find a successful path. In each iteration, any wire that could not be successfully routed are pushed to the front of the priority list. In the next iteration, these wires will be routed before other wires. This process continues until either a successful path has been found or a maximum number of iterations has been met. The program also tracks wire lengths, which can be considered when choosing an optimal path.

## Results

Most circuits in the provided benchmark can be successfully routed with both the Lee-Moore method and the A\* method. The exceptions are **Oswald**, **Stdcell**, and **Temp**. There is a trade-off between routing speed and the wire length for the two methods. Although A\* is able to find a solution faster than Lee-Moore, it often results in a longer path between the source and the sink because it aims directly for the source without considering obstacles. This characteristic enables A\* to succeed in Oswald where Lee-Moore does not but causes A\* to fail in Stdcell where Lee-Moore routes successfully. Table 1 shows the benchmark results.

Table 1 - Benchmark Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Benchmark | Lee-Moore | | A\* | |
| Wires Routed | Segments Routed | Wires Routed | Segments Routed |
| Impossible | 1 of 3 | 6 of 8 | 1 of 3 | 6 of 8 |
| Impossible2 | 2 of 3 | 6 of 7 | 2 of 3 | 6 of 7 |
| Kuma | 3 of 4 | 9 of 10 | 3 of 4 | 9 of 10 |
| Misty | 4 of 4 | 9 of 9 | 4 of 4 | 9 of 9 |
| Oswald | 1 of 2 | 3 of 4 | **2 of 2** | **4 of 4** |
| Rusty | 3 of 3 | 7 of 7 | 3 of 3 | 7 of 7 |
| Stanley | 3 of 3 | 8 of 8 | 3 of 3 | 8 of 8 |
| Stdcell | **8 of 8** | **26 of 26** | 7 of 8 | 25 of 26 |
| Sydney | 3 of 3 | 6 of 6 | 3 of 3 | 6 of 6 |
| Wavy | 1 of 1 | 8 of 8 | 1 of 1 | 8 of 8 |
| Temp | 6 of 8 | 23 of 25 | 6 of 8 | 23 of 25 |

Images of the final routes for the Stdcell and Stanley benchmarks are attached in Appendix A – Final Route Diagrams.

## Application Guide

The program can be run by running the gui.py file. The graphical display is implemented with tkinter and the program also requires the numpy library.

When the program begins, the user is prompted to enter the name of the .infile to be read. Then, the routing problem is drawn, and the user can choose from several actions indicated in Figure 1 and listed below:

* **Init/Step**: This is a step-by-step implementation. Begin by choosing “Init” to initialize the necessary variables, then each click of “Step” will run a single step of the algorithm until the sink/source pair is routed or deemed impossible.
* **Connect 1 Pin**: This is a coarse-grained step-by-step implementation. Each click routes 1 sink/source pair.
* **Run Once**: Choose this option to attempt routing all of the wires for a single iteration.
* **Optimize**: This is the full implementation. By selecting “Optimize”, the program will run for *n* iterations (or until a successful route has been found), while reordering the wires in each iteration as described earlier.
* **Debug**: This is a debugging feature that dumps out the current state of the program.
* **Reset**: This resets the circuit and undoes any existing connections.



Figure 1 - GUI Buttons

The program can be configured using settings.py to adjust various parameters. For example, there is an option to use the closest cell connected to the appropriate source for the A\* algorithm rather than the actual source. The algorithm to be used (Lee-Moore or A\*) is also selected in the settings.py file.

Table 2 outlines the contents of each Python file.

Table 2 - File Descriptions

|  |  |
| --- | --- |
| File Name | Purpose |
| gui.py | Main file containing GUI elements and runs the application |
| infile\_parser.py | Parses the .infile into an appropriate format for the program |
| util.py | Contains useful functions shared between algorithms |
| lee\_moore.py | Implementation of the Lee-Moore algorithm |
| astar.py | Implementation of the A\* algorithm |
| settings.py | Contains settings for the program |

## Testing Procedure

Testing for this program was completed manually. The **Init/Step**, **Connect 1 Pin**, and **Run Once** options were designed to iteratively build up to the final program. Using the **Init/Step**, I can observe that the expansion is proceeding in the correct order. The final program also maintains a log file for each .infile that can be manually checked to ensure the program is proceeding correctly. Otherwise, assertions are included throughout the file and exceptions are raised for unexpected results.

A few additional circuits were also designed as part of the testing procedure.

## Appendix A – Final Route Diagrams

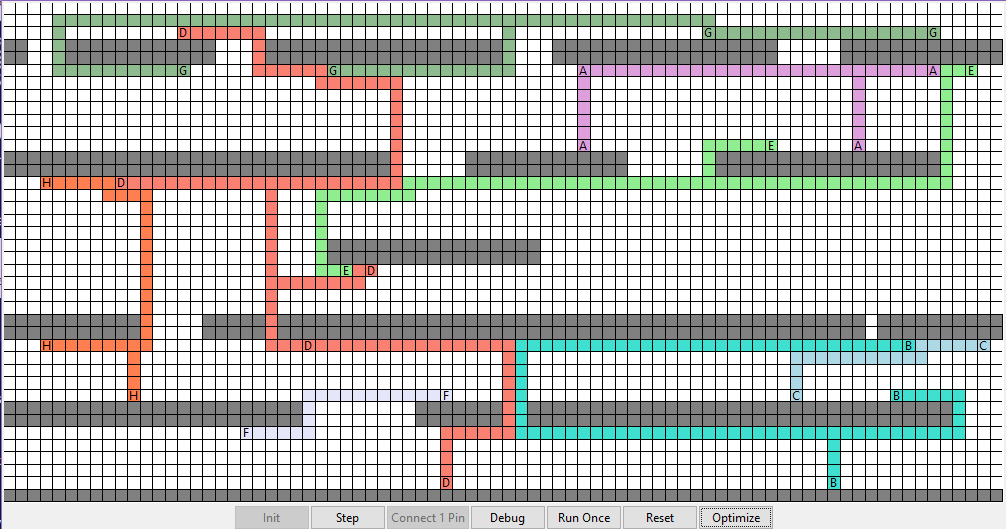


Figure 2 - Stdcell Final Route (Lee-Moore)

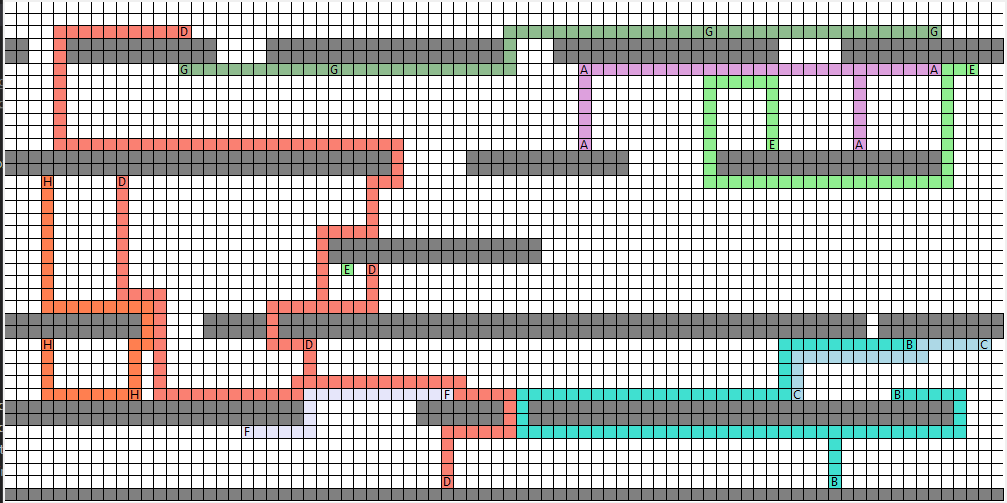


Figure 3 - Stdcell Final Route (A\*)

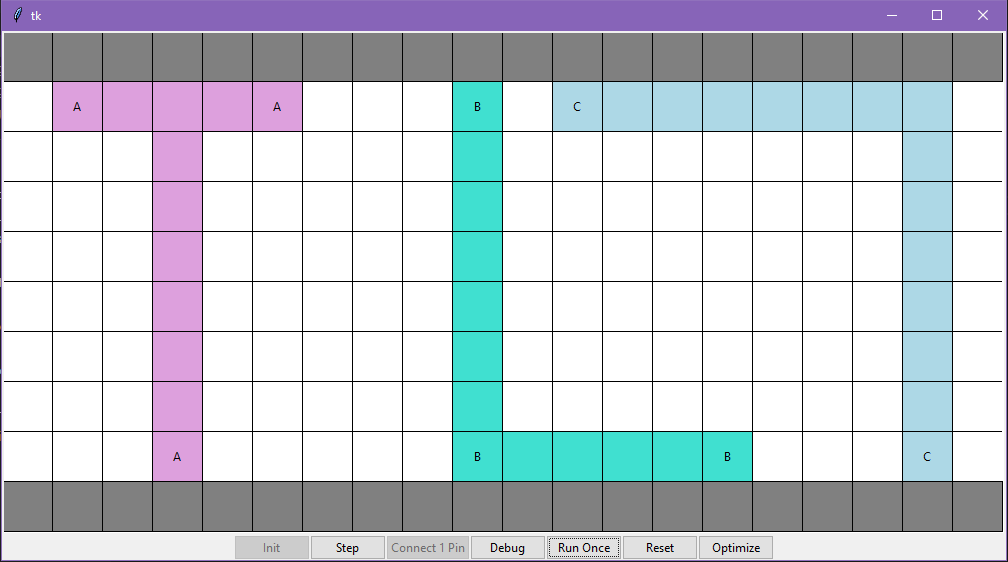


Figure 4 - Stanley Final Route (Lee-Moore)

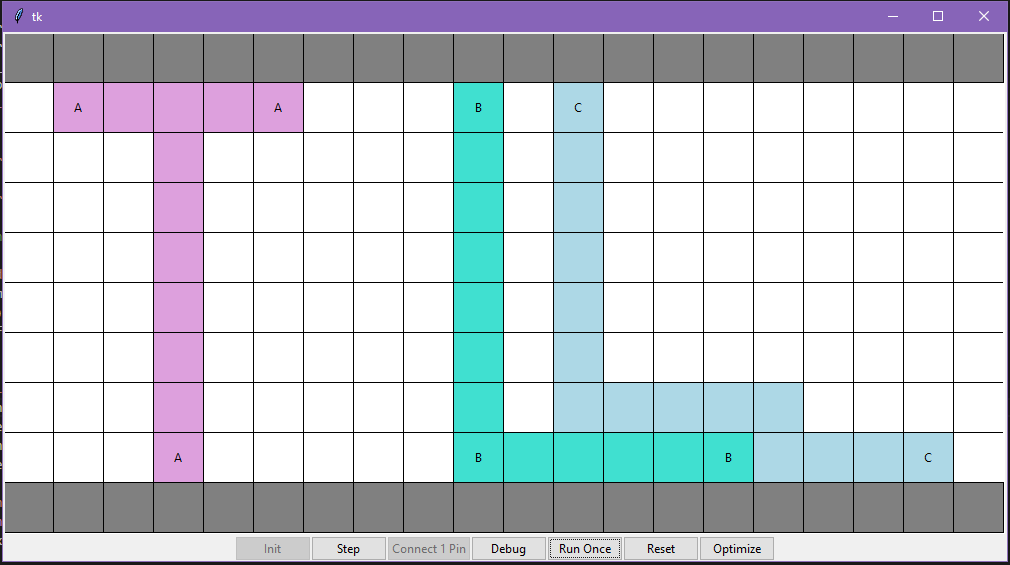


Figure 5 – Stanley Final Route (A\*)