Heatmap Router: An encoder-decoder approach to PCB routing

by

Zachary Taylor Johnson

A Creative Component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Forrest Sheng Bao, Co-major Professor
Julie Dickerson, Co-major Professor

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this creative component. The Graduate College will ensure this creative component is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2022

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

ABSTRACT

Printed circuit board (PCB) routing is a complex problem which has only become more challenging in the face of increasingly complex circuits. With increasing numbers of pins/nets and different physical constraints for every board, routing PCBs is often an expensive and time-consuming task requiring significant manual design effort. A comprehensive and generalized PCB routing algorithm would save countless hours of engineering time. In this work, I propose an algorithm to solve a simplified version of the PCB routing problem using a heuristic-based search that utilizes an encoder-decoder generated heatmap. The algorithm can solve many examples of 5-net, 30x30 circuits, and could be extended to work with more nets, more layers, and larger layers. I also compare the performance of this algorithm with some popular open source routing algorithms that are currently available.

CHAPTER 1. INTRODUCTION

PCBs are ubiquitous in our everyday lives. Almost all computing devices, even those that we don't traditionally call "computers", contain one or more PCBs. Additionally, any given PCB may contain thousands of pins, multiple layers, and a series of unique design constraints, making the design process very expensive and time-consuming [Hui Kong and Wong (2009)]. Automating any part of this process has the potential for great reward. Circuit routing, or the design step in which nets are connected via the placement of wire, is a very important part of the design process. Unfortunately, circuit routing is also a very challenging problem: an NP-Hard one, in fact [Kramer and Van Leeuwen (1982)]. This, combined with the large sizes of modern circuits, forces us to come up with clever solutions.

Often, routing circuits involves designing a custom routing algorithm using domain knowledge or routing the circuit manually and using an algorithm to detect inefficiencies, solutions which are far from ideal considering the amount of engineering overhead. Additionally, many algorithms have a global routing phase, in which constraints are relaxed, followed by a detailed routing phase, where global routing results are used to place wires of the appropriate size in the correct positions [Charles J. Alpert and Sapatnekar (2008); Chen and Chang (2009); Andrew B. Kahng and Hu (2011)]. Unfortunately, these two phase solutions are not perfect; often the global routing results do not lend themselves well to the detailed routing algorithm [Zhang and Chu (2012)].

This report outlines an algorithm which leverages a deep learning heuristic to route small circuits. The primary element of the algorithm is a heuristic function based on the output of an encoder-decoder neural network. The network is trained on thousands of circuit heatmaps, generated by taking pre-solved circuits, generating hundreds of alternative solutions, and "adding" those solutions together, resulting in a heatmap of the most-traveled-through parts of

the circuit. The trained encoder-decoder network can then generate heatmaps for new, never-before-seen circuits, and use those heatmaps to guide an A* grid search of the circuit.

Chapter 2 outlines the problem formulation, including the way the circuit routing problem will be framed in this context as well as the constraints on the algorithm inputs. Chapter 3 describes the data generation process used to create the training dataset for the encoder-decoder network. Chapter 4 covers the details of the routing algorithm itself. Chapter 5 describes the metrics used to evaluate the encoder-decoder network and routing algorithm performance, and provides a comparison to other popular routing algorithms and benchmarks. Chapter 6 concludes the report and highlights possible future work.

CHAPTER 2. PROBLEM FORMULATION

This section outlines the constraints and formulation applied to the circuit routing problem for the purposes of this project.

2.1 Constraints

For the sake of simplicity and ease of implementation, constraints have been placed on the dimensions and net counts of the input circuits. Input circuits must conform to the following constraints:

- Circuits must have only a single layer.
- Circuit width and height must be 30x30.
- Circuits must have exactly 5 two-pin nets.

The constraints are quite strict in order to allow a fixed-size input to the convolutional encoder-decoder network. However, as will be discussed later, future work could allow for these constraints to be relaxed.

2.2 Formulation

The routing problem is formulated as a grid search, and therefore the circuit is formulated as a matrix. The input to the routing algorithm is a single 30x30 matrix in which an integer 1 represents an obstacle, an integer 0 represents an empty square, and any integer 2-6 represents a net pin, where the number corresponds to the net it belongs to plus one (i.e. the pins of net 1 are labeled with a 2). The goal of the algorithm is to connect the two pins of each net with a path without any of the paths intersecting one another. For the sake of simplicity, paths can only move in 4 directions: up, down, left, and right.

CHAPTER 3. DATA GENERATION

In order to train the encoder-decoder network, a large number of data points had to be generated. This section describes the format of the data used to train the model as well as the methods used to generate it.

3.1 Heatmap Generation

3.1.1 Generating Input Circuits

Test circuits were generated through a random circuit generation process. In this process, 5 two-pin nets were placed first, followed by 2-4 large block obstacles. Lastly, small (one grid square) obstacles were added to each grid square with a probability of 0.1.

3.1.2 Solving Input Circuits

Once circuits were generated, they needed to be solved. This was done using Lee's path connection algorithm. Of course, Lee's algorithm is not able to solve all circuits, so many failed [Zhang (2016), Huang et al. (2014)]. However, it was not necessary for all of the generated circuits to be solved. The number that were successfully solved was enough for a very large dataset. In total, this data generation process resulted in 21,446 circuits.

Note: the failure of Lee's algorithm to solve all of the circuits is a potential flaw of the data generation process. Because only solved circuits are used for training, this may result in a selection bias effect in which the encoder-decoder model can only produce accurate results for "easy" circuits. However, this does not appear to have had limiting effects on the results of the algorithm.

3.1.3 Generating Heatmaps

The last step in the data generation process is the computation of the heatmaps. These were generated by repetitively making slight modifications to the existing circuit solutions and updating the heatmap with each repetition. The process was as follows:

1. Initialize a 30x30 matrix of zeros.
2. Using breadth-first search (BFS), find a new shortest path for one of the nets without intersecting any of the other paths.
3. Repeat steps 2 for each net.
4. Repeat steps 2-3 200 times. Every iteration, add 1 to all grid squares which paths pass through in the empty matrix.
5. Normalize the values in the matrix to be in the range [0, 1].

Figure 3.1 shows an example of a solved circuit before and after a BFS update of each path. As you can see, it is possible to find alternative circuit solutions with minimal computation once we've computed the initial solution to the circuit. By computing a large number of alternative solutions, we can gain some statistical information about where paths are likely to pass through. In other words, at the end of the 200 iterations, the grid squares which are mostly likely to have a path pass through them in a solution will have the highest values. For example, if a path passed through a certain grid square on *every* iteration, the value at that index in the heatmap will be 1. If no paths ever passed through that index, the value will be 0. This information is valuable because it highlights which paths an algorithm should follow to get to its destination. Figure 3.2 shows the heatmap resulting from 200 iterations of the algorithm.

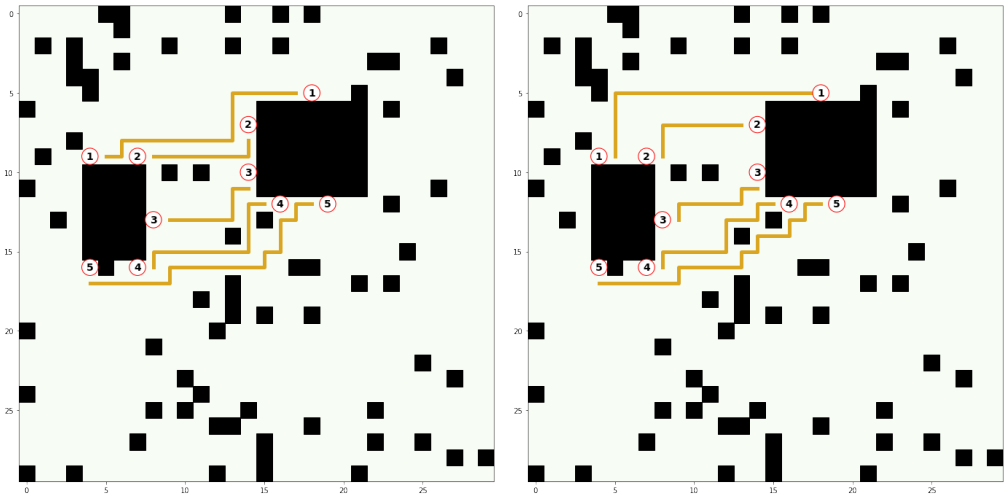

Figure 3.1 Initial solved circuit (left) and alternative circuit solution calculated with BFS (right).

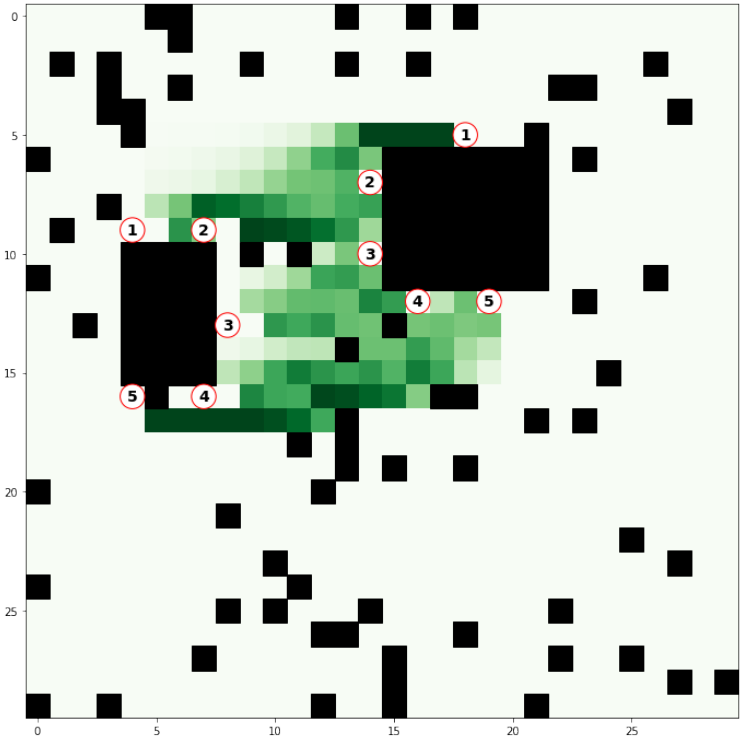

Figure 3.2 Heatmap resulting from adding the 200 circuit solutions together. Darker squares have had more paths pass through them, while lighter squares have had fewer.

3.2 Encoder-Decoder Input

The encoder-decoder input is a representation of the circuit matrix formatted in a way such that the convolutional encoder-decoder can effectively learn the spatial layout of the circuit and use it to generate the heatmap. It is formatted as follows:

- Each net is represented as a 30x30 matrix in which the integer 1 represents a pin and an integer 0 represents a non-pin.
- The obstacles are represented in a single 30x30 matrix in which an integer 1 represents an obstacle and an integer 0 represents a non-obstacle.
- The resulting matrices are concatenated into a single 30x30x6 tensor.

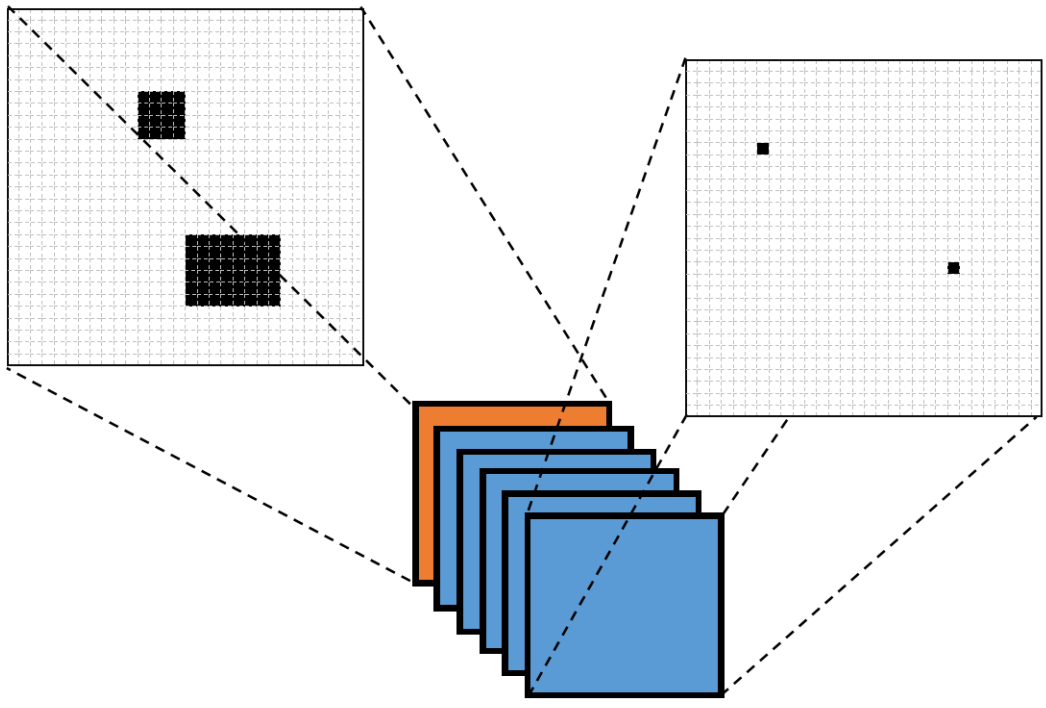

Figure 3.3 Encoder-decoder network input. Blue layers represent net channels while the orange layer represents the obstacle channel. Black squares in the feature matrices represent ones while white squares represent zeroes.

The resulting tensor is used as input to the encoder-decoder network, while the corresponding heatmap of the circuit is used as the expected output.

CHAPTER 4. ROUTING ALGORITHM OVERVIEW

This section outlines the routing algorithm itself, including the basic pathfinding algorithm, the encoder-decoder network, and the ways in which the heatmap is used as a heuristic.

4.1 A* Search

The basis of the algorithm is a simple A* search which solves all of the nets as part of a single search tree to allow backtracking to prevent blocking nets. In other words, we use A* to solve each net; when one net is complete, the next net is begun on the same search tree. This is necessary because if the algorithm reaches a point where no solution is possible, it may need to backtrack to a previous net to unblock the current net. This is an improvement upon sequential A*, which is often blocked due to its inability to backtrack to previous nets. The A* search also provides a timeout parameter which can be used to stop the algorithm after a certain amount of time, resulting in routing failure.

4.2 Heuristic Function

The A* search uses a combination of Manhattan distance and a special "heat" metric derived from the heatmap. The Manhattan distance includes the Manhattan distance from the head of the current path to its destination pin as well as the sum of the Manhattan distances of each remaining unrouted net (from pin to pin). The second part of the heuristic is what I will call "heat". The total heat is equal to the sum of the heatmap values at each index which a path has passed through so far. Since heat in this case should be incentivized, the total heat is subtracted from the Manhattan distance. The heuristic function also incorporates a parameter called "heat sensitivity" which scales the heuristic's reliance on the heat metric via multiplication. The resulting heuristic function is thus: $h(x) = manhattan\_dist - heat\_sensitivity * total\_heat$.

4.3 Thresholding

In addition to being used for the heuristic function, the heatmap can also be used to limit the search space. By blocking grid squares which have too low of a heat value (by placing obstacles), we can prevent the path search algorithm from searching large portions of the circuit. The selectiveness of the thresholding is determined by a user-provided parameter. Any grid squares with heat lower than the threshold parameter are blocked. Figure 4.1 shows an example of a board that has been thresholded to only include high-heat grid squares.

4.4 Parameter Grid Search

Using too extreme of a threshold, heat sensitivity, or timeout value can be detrimental to routing or even cause failure. Therefore, results can be improved by trying multiple parameter combinations. Heatmap router does this by performing a grid search over all possible parameter combinations. A user provides three lists: a list of threshold values, a list of heat sensitivities, and a list of timeouts. The algorithm attempts to route the circuit with every combination of parameters and terminates when the circuit is solved or when the final parameter set has been tested. For testing, the following parameter sets were used:

- Threshold: [0, 0.01, 0.1]
- Timeout: [5]
- Heat Sensitivity: [3, 5, 1, 6, 0.5, 8, 4, 7, 9, 10]

4.5 Convolutional Encoder-Decoder Network

This section describes the convolutional encoder-decoder network used to generate heatmaps, including the model architecture and training process

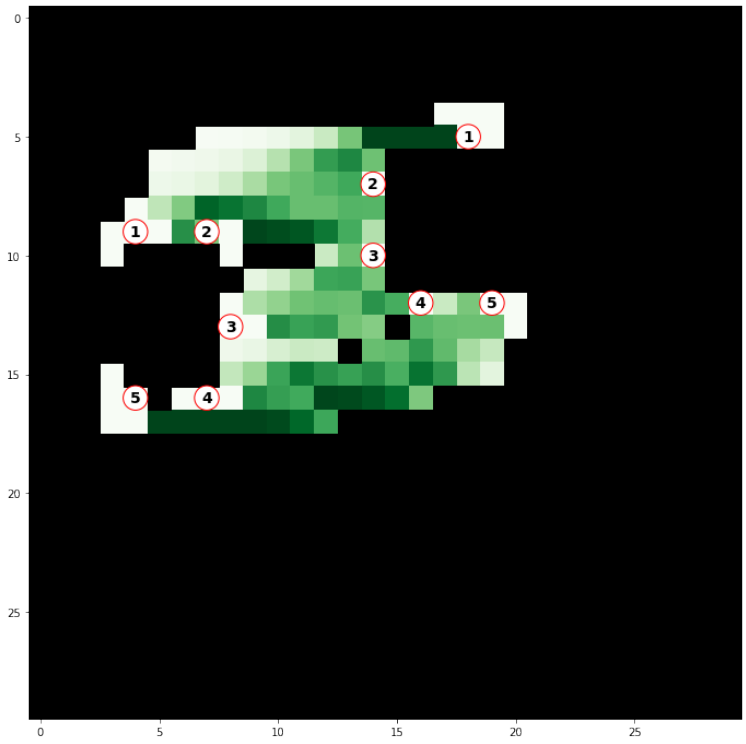

Figure 4.1 An example of a thresholded board. All grid squares with a heat value less than 0.1 have been covered by obstacles so that they cannot be searched.

4.5.1 Model Architecture

The heatmap model consists of an encoder and decoder, each consisting of several convolutional layers and a single fully-connected layer. Each layer is followed by batch normalization, and fully-connected layers have dropout with a probability of 0.3. The final layer of the decoder has a sigmoid activation function in order to output values between 0 and 1. The model architecture is described in Table 4.1 and Table 4.2.

Table 4.1 Encoder Network Architecture

Layer Type (Activation)	Input Channels	Output Channels	Kernel	Stride
Conv2d (ReLU)	6	64	5x5	1
Conv2d (ReLU)	64	128	3x3	2
Conv2d (ReLU)	128	128	3x3	1
Conv2d (ReLU)	128	256	5x5	2
Conv2d (ReLU)	256	512	3x3	1
Conv2d (ReLU)	512	512	3x3	1
Conv2d (ReLU)	1024	1024	3x3	2
Fully Connected (ReLU)	16384	256	N/A	N/A

Table 4.2 Decoder Network Architecture

Layer Type (Activation)	Input Channels	Output Channels	Kernel	Stride
Fully Connected (ReLU)	256	16384	N/A	N/A
Conv2d (ReLU)	1024	512	3x3	2
Conv2d (ReLU)	512	512	3x3	1
Conv2d (ReLU)	512	256	3x3	1
Conv2d (ReLU)	256	128	5x5	2
Conv2d (ReLU)	128	128	3x3	1
Conv2d (ReLU)	128	64	3x3	2
Conv2d (Sigmoid)	64	1	5x5	1

4.5.2 Training Process

The training dataset consisted of 21,446 samples, which were split into 15,012 training, 3,217 validation, and 3,217 test samples. The model was trained for 1,200 epochs with a batch size of 64, mean squared error (MSE) loss function, and an Adam optimizer.

CHAPTER 5. PERFORMANCE

This chapter outlines the resulting performance of the heatmap encoder-decoder model and routing algorithm. Results include performance metrics as well as comparison with other routing algorithms.

5.1 Heatmap Model

The encoder-decoder model was trained to minimize MSE. While this is not a very human-interpretable metric, it can at least help us to understand how well the model generalizes. This model resulted in a training MSE of 0.0549 and a test MSE of 0.0565. The fact that these two numbers are not drastically different illustrates that the model probably generalizes well to unseen inputs.

5.2 Routing Algorithm

5.2.1 Real Boards

To compare performance of the routing algorithm to an existing, popular solution, a test set of 8 real open source PCBs (in KiCAD format) was collected. From these PCBs, a dataset of routing inputs was created by separating the layers, choosing 5 two-pin nets, and reducing the size of the boards to 30x30 using max pooling. Through this series of transformations, a PCB can be converted into a form that fits the input constraints of the algorithm.

All of these modified boards were tested with Heatmap Router, FreeRouting [Wirtz (2004)], and DeepPCB [InstaDeep]. Of the 8 boards, all of them were able to be solved by all three algorithms. Heatmap Router and FreeRouting were both able to solve all of the circuits in a few seconds or less, while DeepPCB took 10 or more minutes to solve each of the circuits. Figure 5.1 shows two examples of real boards that were solved by Heatmap Router.

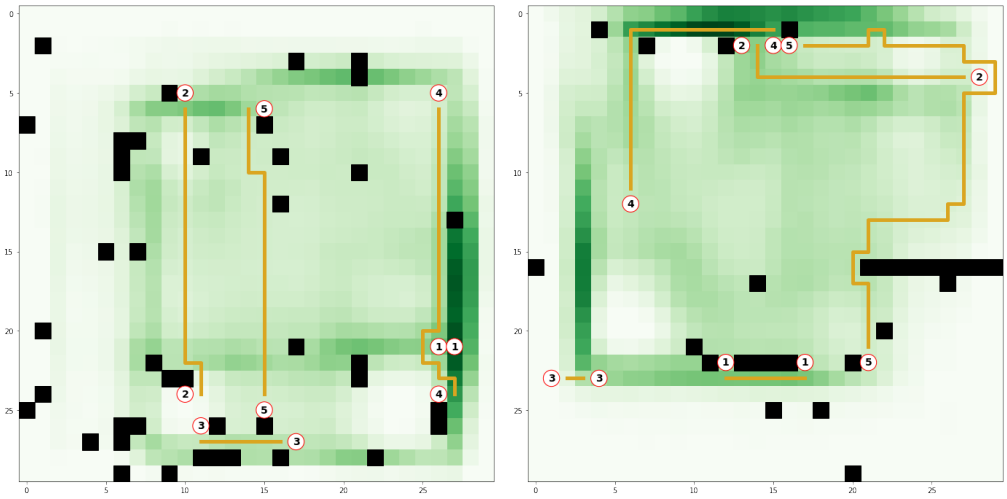

Figure 5.1 Two modified real-world PCBs solved by Heatmap Router.

5.2.2 Randomly Generated Boards

Additionally, a set of 100 random boards was generated using the process outlined in section 3.1.1. Heatmap Router and FreeRouting were both tested on these circuits. Heatmap Router was able to successfully solve 83% of the circuits, while FreeRouting was able to solve 100% of them. Figure 5.2 shows two examples of randomly-generated boards that were solved by Heatmap Router.

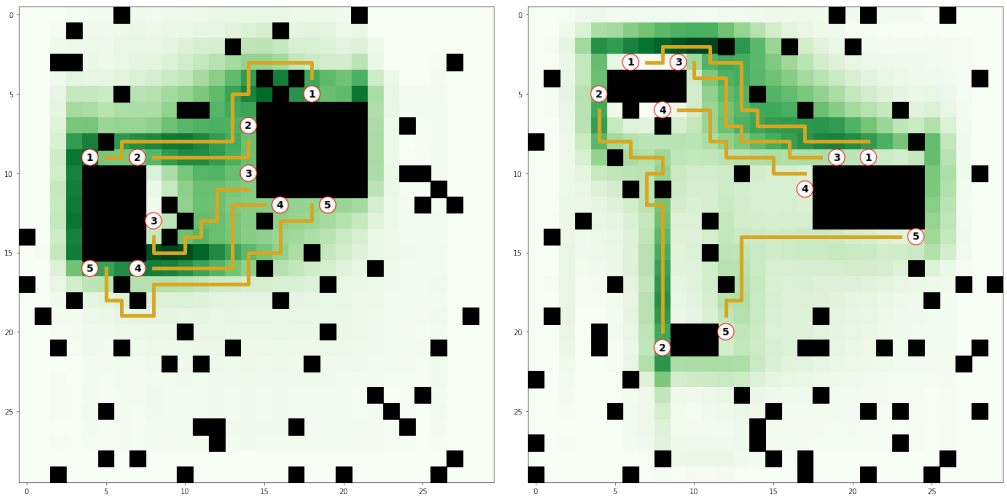

Figure 5.2 Two randomly-generated PCBs solved by Heatmap Router.

CHAPTER 6. CONCLUSION AND FUTURE WORK

The result of this work is a circuit routing algorithm which on its own is not incredibly effective, but which could be extended and repurposed to be very useful. The encoder-decoder heatmap concept could be applied to other circuit routing algorithms, especially those that utilize heuristic-based pathfinding. Additionally, thresholding could likely be applied as a preprocessing step to many different types of routing algorithms. In order to be useful for more types of circuits, however, the algorithm will have to be generalized to allow larger circuit dimensions and more nets. Methods such as recurrent neural networks could be applied to allow circuits with more than 5 nets. The A* search could be modified to allow for nets with more than two pins. Various minor improvements could be made to boost performance, such as randomizing net order and retrying the search.

The concepts applied in Heatmap Router certainly have a lot of room for improvement. Hopefully future researchers can utilize and improve these ideas to create even better routing algorithms.

BIBLIOGRAPHY

Andrew B. Kahng, Jens Lienig, I. L. M. and Hu, J. (2011). Vlsi physical design: from graph partitioning to timing closure. *Springer Science Business Media*.

Charles J. Alpert, D. P. M. and Sapatnekar, S. S. (2008). *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications.

Chen, H.-Y. and Chang, Y.-W. (2009). *Electronic Design Automation*.

Huang, T.-W., Wu, P.-C., and Wong, M. D. (2014). Ui-route: An ultra-fast incremental maze routing algorithm. In *2014 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*, pages 1–8.

Hui Kong, T. Y. and Wong, M. D. F. (2009). Automatic bus planner for dense pcbs. *2009 46th ACM/IEEE Design Automation Conference*, pages 326–331.

InstaDeep. Deeppcb. https://www.deeppcb.ai/.

Kramer, P. and Van Leeuwen, J. (1982). Wire routing is np-complete.

Wirtz, A. (2004). Freerouting. https://freerouting.org/.

Zhang, R. (2016). *A Study of Routing Algorithms for PCB Design*. PhD thesis, Waseda University.

Zhang, Y. and Chu, C. (2012). Interleaved global routing and detailed routing for ultimate routability. *In DAC Design Automation Conference*, pages 597–602.