Machine Learning Based Techniques for Routing Interconnects in Very Large Scale Integrated (VLSI) Circuits

by

Zhonghua Zhou

B.Sc., Shenzhen University, China, 2012

M.Sc., University of California, Riverside, USA 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering)

The University of British Columbia

(Vancouver)

April 2022

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Machine Learning Based Techniques for Routing Interconnects in Very Large Scale Integrated (VLSI) Circuits

submitted by **Zhonghua Zhou** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Electrical and Computer Engineering**.

Examining Committee:

André Ivanov, Professor, Department of Electrical and Computer Engineering, UBC
Supervisor

Guy Lemieux, Professor, Department of Electrical and Computer Engineering, UBC
Supervisory Committee Member

Tor Aamodt, Professor, Department of Electrical and Computer Engineering, UBC
University Examiner

Cristina Conati, Professor, Department of Computer Science, UBC
University Examiner

Abstract

Global routing is a significant challenge in Integrated Circuit (IC) designs due to circuits' increasing number of metal layers, transistors, and the resulting growth in design complexity. Congestion is a crucial factor contributing to routing completion because required interconnects of a design with no congestion can be easily routed. A circuit with congestion will have challenges during routing and may require a re-placement, which lengthens the time to complete the design and may delay time to market. Congestion also affects routing complexity, which may increase wire length and the number of vias and detours in a layout, affecting overall circuit performance. Furthermore, congested areas in a layout may cause manufacturing yield and reliability problems. Congested areas have a higher potential for creating shorts and opens which can eventually lead to unworkable chips. Traditional congestion estimation algorithms use simple, fixed models which do not change as the technology nodes scale to finer dimensions. To address this shortcoming, we investigate Machine Learning (ML) based congestion estimation approaches. By training from previously routed circuits, an ML-based estimator implicitly learns from the advanced design rules of a particular technology node as well as from the routing behaviours of routers.

In this investigation, three ML-based approaches for congestion estimation are explored. First, a regression model to estimate congestion is developed, which is in average 9$\times$ faster than traditional approaches while maintaining a similar quality of routing solution. Second, a Generative Adversarial Network (GAN) based model is developed to accurately predict congestion of fixed-size tiles of a circuit. Third, a customized Convolutional Neural Network (CNN) is designed for congestion estimation which uses a sliding-window approach to smooth tile-based

discontinuities. This CNN produces the best correlations with actual post-routing congestion compared with other state-of-the-art academic routers. Furthermore, an improved global routing heuristic is developed with which congestion estimators can be merged. Results show that my work achieves 14% reduction in runtimes on average compared with other routers and achieves significantly lower runtimes on difficult-to-route circuits. Overall, this work demonstrates the feasibility of using ML-based approaches to improve routing algorithms for complex IC implemented in nanometer technologies.

Lay Summary

Routing, an action that makes wire connections between circuit components, is one of the most computationally expensive processes in designing chips. Most of this runtime is spent on resolving congestion which occurs when too many connections attempt to pass through the same region. To reduce the runtime, we developed new ways of predicting where congestion will occur before the routing, and modified the routing algorithm to use this information and converge more quickly. Traditional congestion estimation methods becomes inaccurate when encountering new technologies due to simplifications in the models used. Instead Machine learning algorithms are developed here and they are able to capture complex and hidden relationships between large quantities of data in circuit design [49, 56]. Experimental results show this work can be computed quickly and are more accurate than prior techniques, and they also lead to a reduction in the global routing runtime.

Preface

The major contributions of this dissertation are described in either published or submitted conference proceedings and journal papers.

For the publications, Zhonghua Zhou is the first author, he proposed the topics and ideas, carried out the research and necessary coding, performed the experiments, analyzed the results. These manuscripts were written by Zhonghua Zhou under the supervision of Prof. André Ivanov (Supervisor) and Prof. Guy Lemieux (Co-Supervisor). Co-authors S. Chahal, Z. Zhu (supervisor: J. Chen), and Y. Ma (supervisor: B. Yu) are students from University of British Columbia (UBC), Fuzhou University and Chinese University of Hong Kong, respectively. They worked with Zhonghua Zhou under cooperative programs, wrote scripts and gathered experimental data. Zhonghua Zhou performed research based on tools and platforms provided by Prof. T. Ho, a visiting scholar to UBC from National Tsing-Hua University, and Dr. P. Hallschmid.

- Chapter 3: Z. Zhou, S. Chahal, T. Ho and A. Ivanov, "Supervised-Learning Congestion Predictor For Routability-Driven Global Routing," 2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 2019, pp. 1-4, doi: 10.1109/VLSI-DAT.2019.8742060.

- Chapter 4 and Chapter 5: Z. Zhou, Z. Zhu, J. Chen, Y. Ma, B. Yu, T. Ho, G. Lemieux, and A. Ivanov "Congestion-aware Global Routing using Deep Convolutional Generative Adversarial Networks," 2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD), Canmore, AB, Canada, 2019, pp. 1-6, doi: 10.1109/MLCAD48534.2019.9142082.

- Chapter 6: Z. Zhou, G. Lemieux and A. Ivanov, "MEDUSA: A Multi-

resolution Machine Learning Congestion Estimation Method for Global Routing," preparing to submit to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) in 2021.

- Appendix C: Z. Zhou, P. Hallschmid and A. Ivanov, "Local congestion and blockage aware routability analysis using adaptive flexible modeling," 2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS), Monte Carlo, 2016, pp. 438-439, doi: 10.1109/ICECS.2016.7841230.

Table of Contents

List of Tables

List of Figures

Glossary

ASIC Application-Specific Integrated Circuit

CNN Convolutional Neural Network

CV Computer Vision

DRC Design Rule Checking

DRV Design Rule Violation

EDA Electronic Design Automation

FCL Fully Connected Layer

FPGA Field-Programmable Gate Array

GAN Generative Adversarial Network

GCV Generalized Cross-Validation

GNN Graph Neural Network

GDSII Graphic Database System Stream

HDL Hardware Description Language

HPWL Half Perimeter Wirelength

IC Integrated Circuit

ML Machine Learning

MST Minimum Spanning Tree

MARS Multivariate Adaptive Regression Splines

NN Neural Network

NRMSE Normalized Root Mean Square Error

PCC Pearson Correlation Coefficient

RSMT Rectilinear Steiner Minimal Tree

SOC System on a Chip

STL Standard Template Library

SVM Support Vector Machine

Acknowledgments

I would like to take this opportunity to thank my supervisor, Prof. André Ivanov, for your help and support for both research and life. Thank you Prof. Guy Lemieux for becoming my co-supervisor. I would never have been able to complete this dissertation without the encouragement, patience and guidance from my supervisors. I am very thankful to Prof. Tsung-Yi Ho for sharing your ideas and research tools when I most needed them. I would also like to express my gratitude to Prof. Steve Wilton, Prof. Sudip Shekhar, and Prof. Shahriar Mirabbasi for being my committee members and helped me through the years of my study. I am very grateful to Dr. S. Arash Sheikholeslam for his friendship and help over the past few years. I would like to thank UBC, NSERC and Huawei; the work would not have been possible without the financial support from them. Finally, I would like to give my sincerest appreciation to my family for their unconditional love, support, encouragement, and patience.

I am forever indebted to my late grandfather for his moral support, and his memory will be with me always.

Chapter 1

Introduction

The routing of wires that interconnect components of Integrated Circuit (IC) is part of the so-called back-end physical design in Electronic Design Automation (EDA) [22, 57]. An IC is developed using a standard cell design procedure [54] and many contain over a billion logic gates. Standard cells are interconnected with wires that form the required power supply, clock, and signal nets. These nets share the same routing resource (metal layers); as a result, the competition for such resources between various nets is a major problem in routing [30]. The severity of this problem, in other words, the *routability* of a design, affects its performance. This dissertation focuses on improving the routability of designs by using Machine Learning (ML)-based algorithms and a novel global routing strategy based on the former.

1.1 EDA of IC Physical Design

The physical design of ICs, particularly placement and routing of components, was a manual process until automation tools arose. EDA tools have evolved since the placement and routing tools were first developed in the 1970s [31, 66, 82, 83]. With EDA, the limitation of manual design was reduced and the possibility of compiling a chip design to silicon from programming languages was advocated by C. Mead's *Introduction to VLSI Systems* [78] and other algorithm developments [9, 47, 80]. In other words, EDA development resulted from a market need [7] and became a

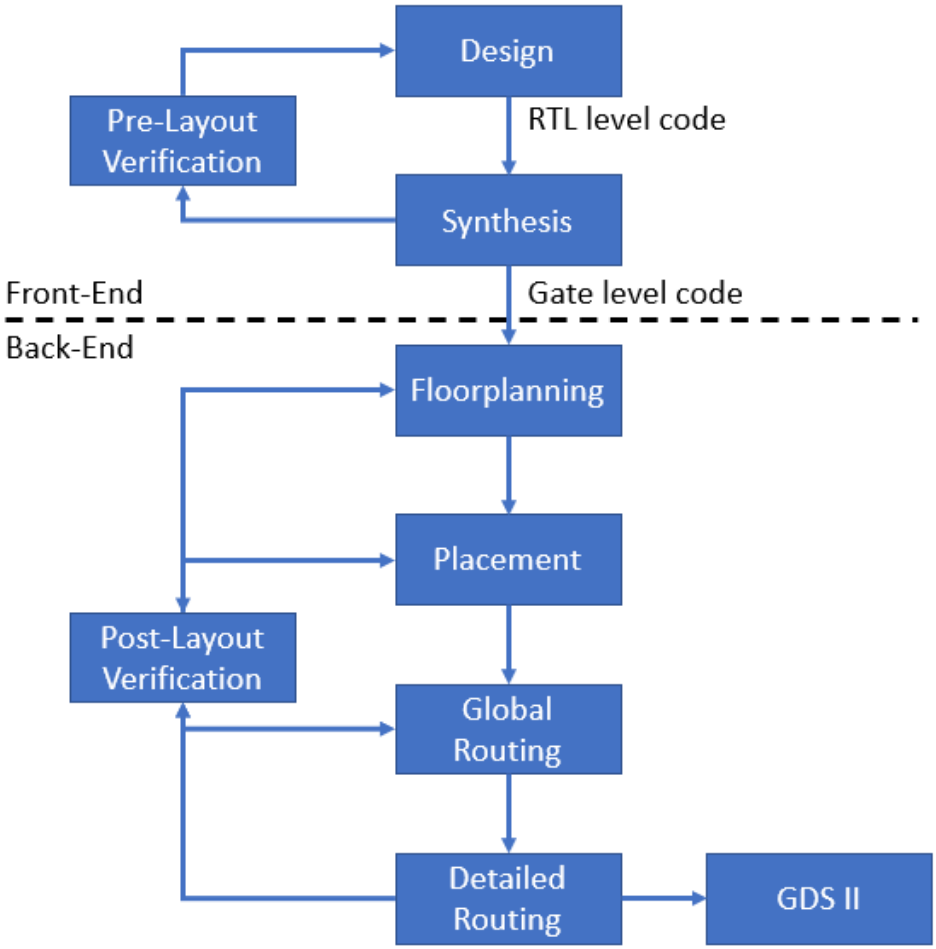


Figure 1.1: The classical EDA flow.

driver for complex IC, such as System on a Chip (SOC) designs [65].

The current classical EDA flow is modular [28], as shown in Figure 1.1. It has two principal stages: Front-End and Back-End. The Front-End stage of the flow mainly focuses on describing functions of a circuit by using textual programming languages. In front-end, a design is developed and described using a Hardware Description Language (HDL) code such as Verilog [5]. This code is a behavioral description of the functionality of a logic circuit. This description is fed to a synthesizer by which a gate-level file will be generated. This gate-level file, called a

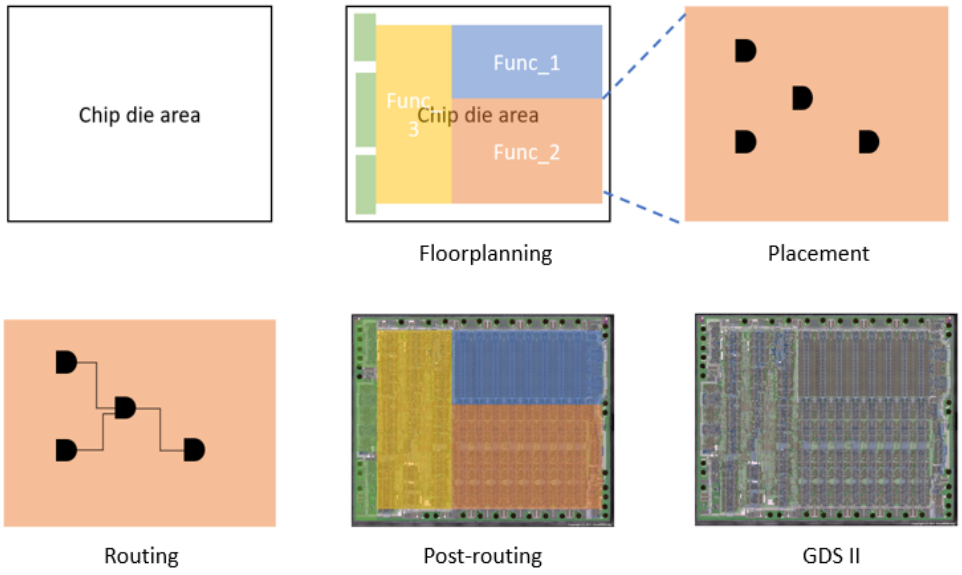


Figure 1.2: Visualization of Back-End physical design stages in EDA.

netlist, is a structural description of a circuit. The structure is based on a network of interconnected logic gates.

The netlist, along with other design information such as standard cell sizes, are passed down to the Back-End stage, as shown in Figure 1.2. The floorplanner [86], which is the first major step of back-end physical design, generates a tentative partitioning and relative location of the functioning blocks in the circuit, such as **Func_1**, **Func_2**, and **Func_3** in Figure 1.2. Once partitioning is determined, placement [35, 39, 42, 44, 103] is performed, it focuses on placing standard cells and gates within each functioning region of the circuit. When placement is finished, the design is passed down to the routing phase. This phase has two steps: *global routing* and *detailed routing*. Routing can be further categorized into two approaches: parallel [11, 17, 113, 114] or sequential [16, 20, 55, 59, 73, 81]. The global routing step determines whether the current placement has enough space to make all interconnects between standard cells, and provides a global routing solution. If a global routing solution is not found, meaning that there is at least one net that can not be routed, then a re-placement is necessary. Otherwise, the detailed routing step is performed and it defines how the actual wires between cells

are connected within the global routing solution. Once the design passes through all the physical design processes and necessary verifications, a binary output file is produced in the format of a Graphic Database System Stream (GDSII) [91], which is used for chip production by semiconductor foundries.

1.2 Motivation

The routing phase determines the wiring connections between the cells that have been placed in the previous stages. Deciding whether or not such an interconnect solution exists under certain constraints is known as an NP-Complete problem [106]. Moreover, the resources for routing signal nets are limited since these nets can only use spaces not taken by power or clock wires. The routing of all nets are sequential, following a certain order [33, 34, 60, 63]. Power supply nets are routed first, then, clocks, and then signals. This competition of resources during routing, called routability, needs attention and optimization in order to increase the likelihood of finding an acceptable routing solution with a feasible computational effort (time). This routability optimization is complex because not only does it need to consider already occupied and limited routing space, but also the increasing number of transistors and design rules due to the downsizing of technology nodes. Design rules introduced to the router must be satisfied to ensure connections are properly formed, such as rules that specify the physical width of wires and the distance between wires. A design is recognized as feasible with a routing solution only if all nets have been routed and rules are met. If the global router reports a design as unfeasible, meaning at least one net can not be routed or one rule not met, then re-placement and re-routing are inevitable. The placement and routing processes are known for their lengthy runtime [54] because the optimization version of placement and routing problems are typically NP-Hard [90]. Therefore, it is important to ensure routability prior to actual routing so as to avoid the lengthy re-iterations of such time-consuming tasks.

1.3 Approach and Contributions

The goal of this dissertation is to optimize the design routability at the global routing phase by using new ML predictive models and their application in a global

routing algorithm. As technology nodes get smaller [4], traditional methods become less effective in terms of both runtime and performance. ML has showed the ability to process large amount of data at new technology nodes and at multiple points of the EDA flow. This dissertation describes also algorithms based upon ML frameworks to improve design routability during global routing by estimating the severity of routing space competition, called congestion, of a circuit placement prior to routing. This dissertation contains chapters based on standalone articles that are either published or under review process.

In Chapter 3, a supervised-learning regression framework [36] is used to investigate the feasibility of congestion estimation using Multivariate Adaptive Regression Splines (MARS). Using MARS, a congestion prediction model is trained to improve the global routing runtime while maintaining the quality of the routing solution. To train this model, various physical layout features are studied and extracted, such as pin position and net density. These features are translated into a matrix format, called input *feature maps*. This mechanism of combining feature maps and a regression algorithm for congestion estimation shows an improved algorithm runtime and an initial routing solution compared with non-ML-based methods. This research shows the improvement of applying ML in EDA for routability optimization in global routing. However, the elements of input feature maps are not spatially coherent, for example, two adjacent elements in an input feature map cannot reflect information of a net if it lays in two adjacent layout regions. Therefore, the idea of developing an advanced feature extraction algorithms is formed to include missing spatial information and it is done in Chapter 4.

In Chapter 4, an advanced Generative Adversarial Network (GAN) framework for congestion estimation is developed. This framework consists of a customized Deep Convolutional Generative Adversarial Network, called c-DCGAN [125]. To apply the proposed c-DCGAN, the input (design features) and output (congestion map) must be converted to an image format. To address this conversion, a Data-Image Translator (DIT) is developed that produces image inputs to the proposed c-DCGAN directly from the circuit netlist. The produced design layout image from DIT is a reinterpretation of the circuit based on the data stored in the netlist. DIT supports both the state-of-the-art industrial "LEF/DEF" [99] and popular academic "bookshelf" formats [1]. The c-DCGAN congestion estimation shows good

correlations with actual post-routing congestion, and visual heatmaps also show good similarity between estimates and actual. However, the c-DCGAN structure is complex compared with other Neural Network (NN) and it requires longer time for training and tuning. Therefore, the structural complexity of the c-DCGAN should be simplified so that the model tuning and training time can be reduced.

Chapter 5 presents a "divide-and-conquer" routing strategy that decomposes the global routing process into four stages, each with their own optimization objectives. The proposed strategy is in contrast to conventional approaches based on a single global optimization objective driving the entire process. This global routing strategy is combined with the proposed c-DCGAN congestion estimation, showing routing quality improvements and runtime acceleration compared with traditional global routing algorithms.

In Chapter 6, two directions of improving ML in congestion estimation are explored: extracting more input features; and simplifying the NN architecture. In order to achieve the above two goals, a new framework, Multi-resolution Machine Learning Congestion Estimation Method for Global Routing (MEDUSA), is developed. MEDUSA consists of three parts: 1) an encoding algorithm which extracts multi-dimension features into multi-channel images, called "hyper-images"; 2) a customized Convolutional Neural Network (CNN) with fixed resolution which takes a tile of this "hyper-image" as input and makes congestion predictions for a sub-region of the tile; and 3) a sliding-window method for repeatedly applying this CNN on a layout. Congestion maps produced with MEDUSA show better accuracy than prior estimation techniques using both quantitative metrics and qualitative visual inspection.

1.4 Dissertation Organization

Chapter 2 presents the background of this dissertation, including the terminology and general metrics of routability-driven global routing, as well as a review of previous work. Chapter 3 illustrates the work using MARS, to explore the feasibility of applying a well-known ML regression framework for congestion estimation. Chapter 4 details the second work, routability optimization framework c-DCGAN, which consists of an advanced GAN, and a feature extraction algorithm. Chapter 5

describes a multi-stage global routing algorithm, which is a strategy that breaks the traditional global routing into four different sub-processes. Chapter 6 covers an optimized framework called MEDUSA, which consists of a modified NN, and a feature extraction algorithm that extends the number of input features for model training. Lastly, Chapter 7 concludes this dissertation.

Chapter 2

Background

This chapter provides an introduction to the routability optimization problems found in IC design process. Before the discussion of congestion estimation for routability optimization, the global routing in EDA is reviewed, and the basic terminologies and metrics are introduced. Then, the congestion estimation problem in IC design process is formulated. Finally, various congestion estimation methods are discussed and their performance are evaluated.

2.1 Global Routing Topology

To manage the complexity of routing in EDA, the routing process is organized in two sequential steps: *global routing* and *detailed routing*. In the global routing step, the global router assigns a channel or a series of channels for all the nets of the design with the objective of minimizing wirelength while at the same time avoiding congestion, the demand or over-consumption for wiring resources (metal tracks) in specific locations. Then, the detailed router assigns nets to specific wiring tracks within the global routing channels.

A global router determines the channels used to connect all pins for each net. This global routing process is performed on a graph which represents the physical layout of the circuit. This graph is created in two steps, as illustrated in Figure 2.1. First, the layout is tessellated into multiple sub-regions, called tiles. The tiles form the grid. Second, a graph $G = (V, E)$ is extracted from the grid, where V is the set

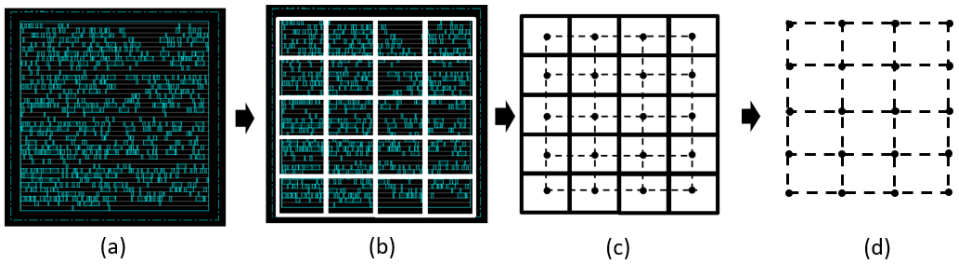


Figure 2.1: (a) A layout is divided into (b) rectangular tiles. (c) The dark dots (vertices) and dashed lines (edges) of the graph $G = (V, E)$ represent the tiles and the boundaries of the grid, respectively. (d) The final routing graph.

of vertices and E is the set of edges. Each vertex $v \in V$ represents a corresponding grid tile, and each edge $e \in E$ between vertices is the shared boundary of two tiles. Tiles are connected with vertical and horizontal edges (routing channels), where routes between pins are determined and wires can be created. The routing graph $G = (V, E)$ is a taxicab geometry in which the *distance* between two vertices is defined as the sum of the absolute differences of their Cartesian coordinates [104]. This metric is referred to as the rectilinear distance between vertices, also known as the manhattan distance.

2.1.1 Minimum Spanning Tree

A Minimum Spanning Tree (MST) [98] is a tree topology constructed from a weighted graph that connects vertices with minimum weight cost and without the creation of new vertices and cycles. In the case of global routing, the weight is defined as the distance between vertices. In the example shown in Figure 2.2, two versions of MSTs are constructed using the same set of 4 vertices, and each of the vertices is connected through an existing edge of the graph such that the MST connects all vertices acyclically. An MST $T = (V_T, E_T)$ is a subgraph of the graph $G = (V, E)$, such that $V_T \subseteq V$ and $E_T \subseteq E$. Given a total of n nodes, the best runtime complexity $O(nlog(n))$ can be achieved if binary heap [112] is used when constructing the MST. However, the Prim's algorithm [88], an implementation without the use of binary heap, is commonly used with time complexity of $O(n^2)$ [108].

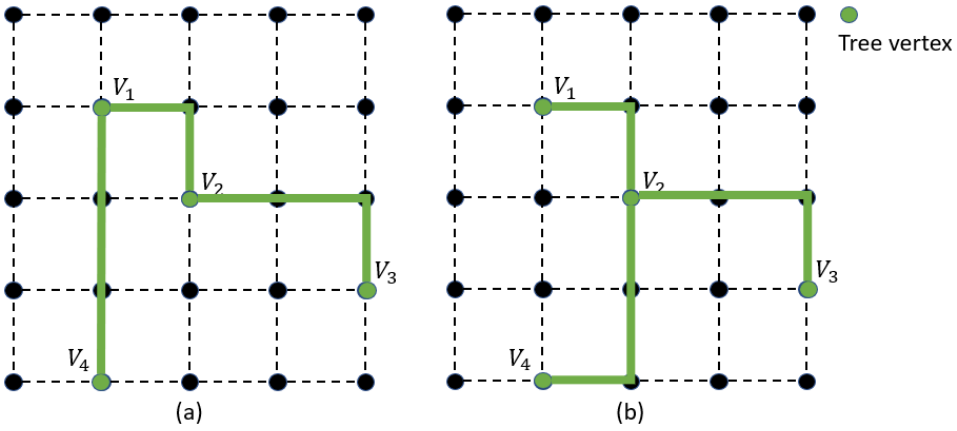


Figure 2.2: Two versions of MSTs using the same set of vertices.

2.1.2 Rectilinear Steiner Minimal Tree

The Rectilinear Steiner Minimal Tree (RSMT) [43, 51] is a graph topology which secures the shortest rectilinear distance between vertices of the tree by introducing the concept of finding "Steiner Points": Given a set P of p pins in a net, find another set S of s Steiner points such that an MST of $(P \cup S)$ has the minimum distance. The RSMT using 4 vertices is illustrated in Figure 2.3. Two extra Steiner points are created such that the overall total length of the tree is guaranteed the shortest while the connections between vertices and Steiner points are all rectilinear.

Finding an RSMT is an NP-Hard problem [38], many algorithms [13, 26, 41, 75, 122] using optimal or near-optimal heuristics have been developed for this task. However, the computational complexity of these algorithms is too expensive to use in practice, therefore a corresponding MST that uses the same set of vertices are often preferred.

2.1.3 RSMT vs. MST

The RSMT, by definition, provides the shortest rectilinear path between vertices. The MST, on the other hand, produces a length that can go up by $1.5\times$ compared to that of an RSMT given the same set of vertices [52]. As shown in Figure 2.4, the more vertices there are to form a tree, the longer length the MST could have when compared with a corresponding RSMT. As a result, there is a trade-off between

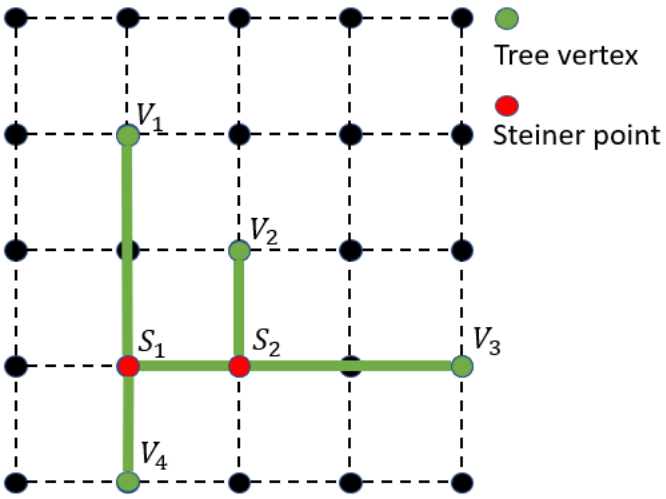


Figure 2.3: An example of an RSMT.

runtime and overall length when choosing between a RSMT or an MST; the RSMT is shorter in length but MST is faster and easier to construct.

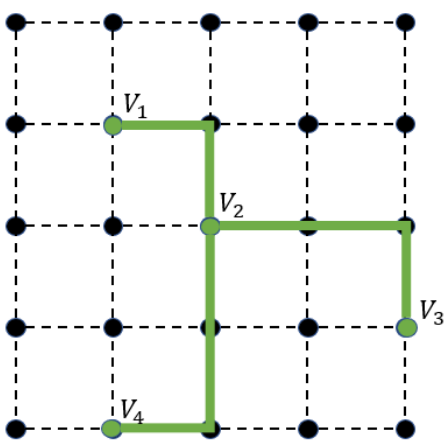
2.2 Global Routing Metrics

The following terminology and definitions are relevant to global routing and are commonly recognized and used by the EDA community [12].

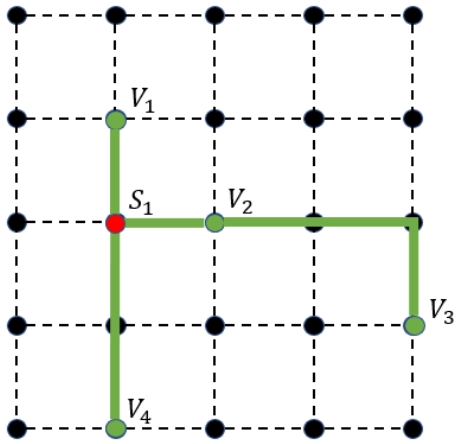
Capacity

The routing *capacity* of an edge $e \in E$, $C(e)$, between two vertices $u \in V$ and $v \in V$ is defined as the maximum available routing resource (metal layer space) between the two vertices. Only adjacent vertices are assigned an edge with a capacity greater than zero.

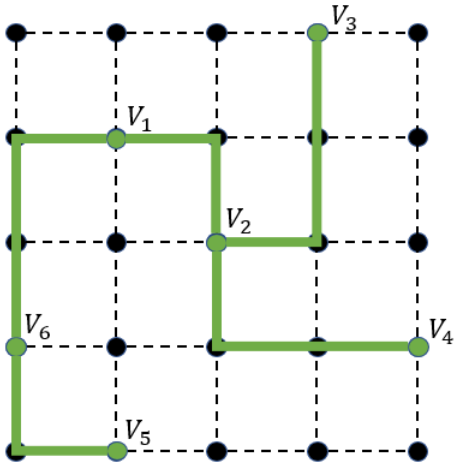
Routing capacity can be measured in different arbitrary ways. For example, in a simple capacity metric, the capacity of an edge e can be defined as the maximum number of routing tracks that can cross that edge. The calculation of the capacity can take into account other factors, such as deducting from the capacity any area blocked by an obstacle such as standard cell blockages. Design rule constraints that require routing resources can also be translated into a reduction of capacity



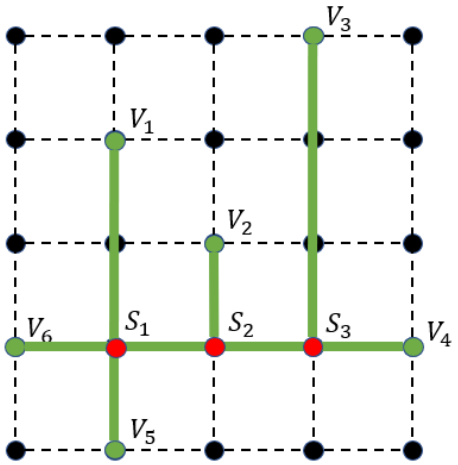
(a) An MST with 4 vertices, and the length is 8.



(b) An RSMT with 4 vertices, and the length is 7.



(c) An MST with 6 vertices, and the length is 13.



(d) An RSMT with 6 vertices, and the length is 11.

Figure 2.4: Examples of MSTs and RSMTs, where (b) is 1 unit length shorter than (a) and (d) is 2 than (c).

such as a minimum width requirement of routing tracks.

Utilization

The *utilization* for an edge $e \in E$, $U(e)$, is defined as the amount of routing resource occupied by signal wires, vias, and the minimum spacing between them. Similar to capacity, the utilization of edge e can be quantified by the number of existing routing tracks that cross the edge.

Routing Track

A routing *track* is a path available for signal net wiring in a certain direction (horizontal or vertical). A net usually uses a sequence of vertical and horizontal tracks. Track direction is normally fixed for each layer, and connecting tracks with different directions is done by adding inter-layer vias.

Via

A *via* is created when routing must change its metal layers, and this typically comes with a change of routing direction too, called a bend. A bend, by definition, means that the direction of a routing wire changed to the perpendicular direction from its original one. A via affects the utilization of an edge $e \in E$. It is considered an important objective to minimize during routing because a via is usually wider than wire. One via could take up spaces in multiple layers and block more than one wire from passing through those areas and therefore a via can consume more routing resources than a wire does.

Overflow

The $overflow$ of an edge $e \in E$, is the amount of excess utilization of that edge. If the utilization is larger than its capacity, in other words, the overflow of one edge is larger than 0, then this edge is considered "congested-to-be". Detailed routing will not be able to route all nets assigned to congested areas due to a lack of routing resources. The overflow of an edge $e \in G$ is expressed as:

$$OF(e) = \begin{cases} U(e) - C(e), & U(e) > C(e) \\ 0, & otherwise. \end{cases} \tag{2.1}$$

where $U(e)$ and $C(e)$ are the utilization and the capacity of the edge $e \in G$, respectively. The *total overflow* (TOF) of a circuit is defined as the summation of the individual $overflow$ of each edge $e \in G$, and it can be expressed as:

$$TOF = \sum_{e \in E} OF(e) \tag{2.2}$$

A global routing solution with $TOF > 0$ is considered invalid, and it requires a re-placement and re-routing to find a new solution with $TOF = 0$.

Wirelength

The *wirelength* of a route between two pins is defined as the number of edges this route has crossed under the condition that the routing is performed in a taxicab geometry. The wirelength becomes longer if a detour is needed to avoid congested edges.

Given a set P of p pins from a multi-pin net i with a set S of s Steiner points (if any), the union set ($P \cup S$) forms a tree topology T_i. The wirelength of this net, $L(T_i)$, is the length of the tree if no detour is made. The *total wirelength* (TWL) of a design is defined as the summation of wirelength of all nets:

$$TWL = \sum_{i=1}^{n} L(T_i) \tag{2.3}$$

where n is the number of nets to be routed in a design. The minimization of wirelength is an important metric for global routing. Decreased wirelength implies smaller power consumption and better chip performance. Minimizing wirelength is often at odds with minimizing congestion since detours (which increase wirelength) may be needed to avoid congested regions. Therefore, the trade-off between these features has to be carefully tuned based on the requirements of circuit designs.

2.3 Global Routing Optimization - Routability

One of the key objectives of global routing is to optimize routability of the design, other optimization objectives can be chip performance and power drop. A better

routability means that a routing solution can be completed and computed with less effort. Routability-driven global routing, a path-finding [45] variation problem, is one significant step in the back-end physical design phase.

2.3.1 Importance of Routability

A design without routability optimization may result in a slower or nonfunctioning circuit at the end of the physical design process. Three common issues raised by this factor are:

- Performance: The lack of routing resources could cause wires to make detours, which lead to longer wirelength and wire delay. If the delay is increased on a critical path, it will operate more slowly.
- Convergence: Uncertainty introduced by congestion and routing detours may affect convergence of the design flow. If one circuit component cannot be routed within its designated region, surrounding blocks may have to be redesigned for a new possible routable solution. Otherwise, the implementation cannot be completed
- Yield: The manufacturing yield can also be negatively affected by deficient routability optimization in two ways: (1) dense wiring regions have an increased number of vias, which may lower the yield of the design; and (2) congested wiring regions have a higher probability for the appearance of shorts and opens.

2.3.2 Routability Optimization - Congestion Minimization Formulation

The routability of a chip can be quantified by the severity of its congestion [94], and the goal of routability optimization is to minimize the congestion. As described in Section 2.1, a routing graph $G = (V, E)$ is constructed at the beginning of the global routing process. Let n be the number of nets to be routed in global routing. For an edge e in the graph G between two adjacent vertices $u \in V$ and $v \in V$, then the capacity $C(e)$ is calculated according to the design specifications and available

tracks between u and v. The congestion of an edge e, $\lambda(e)$, is the utilization of the edge e divided by its capacity, and it can be written as:

$$\lambda(e) = \frac{U(e)}{C(e)} \tag{2.4}$$

Under the following assumptions: 1) that the capacity of all edges $C(e), e \in E$ are the same; 2) the rectilinear distance is 1 between each and every two adjacent vertices; and 3) the utilization value of an edge e, $U(e) += 1$ when a routing track crosses that edge, then the summation of utilization over all edges of the graph is equal to the total wirelength of all nets in a design, this relationship can be expressed as:

$$\sum_{e \in E} U(e) = \sum_{i=1}^{n} L(T_i) \tag{2.5}$$

The summation of congestion over all edges of the graph can be derived from Equation 2.4 and Equation 2.5 as an equivalent to the TWL of a design:

$$\begin{aligned} \sum_{e \in E} \lambda(e) &= \sum_{e \in E} \frac{U(e)}{C(e)} = \frac{1}{C(e)} \sum_{e \in E} U(e) \\ \sum_{e \in E} \lambda(e) &= \frac{1}{C(e)} \sum_{e \in E} U(e) = \frac{1}{C(e)} \sum_{i=1}^{n} L(T_i) = \frac{TWL}{C(e)} \end{aligned} \tag{2.6}$$

Minimizing the total congestion is one aspect of routability optimization problem. The congestion minimization can be formulated as:

$$min \sum_{e \in E} \lambda(e) = min \frac{TWL}{C(e)} = min \sum_{e \in E} \frac{U(e)}{C(e)} \tag{2.7}$$

the above notation indicates that congestion can be minimized by reducing TWL if assumptions are met when Equation 2.5 is introduced. In practise, congestion should be minimized by decreasing $U(e)$ or increasing $C(e)$, in other words, reducing the overflow, $OF(e)$. All works presented in this dissertation evaluate the performance of congestion minimization using both metrics, but with a emphasis on the latter.

2.4 Congestion Minimization - Congestion Estimation

In order to minimize the congestion of circuit designs, a global router must evaluate the congestion of a given placement configuration at an earlier stage before actual routing is completed. The Equation 2.6 shows congestion depends on the tree topology of nets in a design, $T_i = (V_{T_i}, E_{T_i})$. If MST is chosen for the congestion prediction, the estimated wirelength and utilization could be much longer than the actual situation. On the other hand, RSMT is more similar to the topology generated during the actual routing in terms of wirelength but its generation is an NP-Hard problem [38]. Therefore, different congestion estimation algorithms have been proposed to approximate the utilization of edges instead, skipping construction of the RSMT.

Different estimation algorithms entail different trade-offs between the computational overhead and topology prediction accuracy [65]. They range from simple-but-fast proxies for congestion, such as wire density, to accurate-but-slow estimation approaches such as fast global routing. The simple-but-fast techniques are more often deployed in the early stages such as placement, whereas accurate-but-slow techniques are more suitable for later stages, such as global routing.

2.4.1 Simple Metric-Based Estimations

Simple metric-based estimation algorithms for congestion estimation generate a congestion map based on simple design parameters. For example, pin density of an edge, which is the total number of pins in the two vertices that are connected by the edge, divided by the area of those two grid tiles, is one such design parameter. Rent's rule [64] is one of the earliest works used to estimate congestion [105, 118, 119]. A relationship between Rent's rule and wirelength is discovered in [29, 32], making them similar metrics for congestion estimation and such an algorithm is reported in [23]. Moreover, the work in [14] proposed to use the total number of nets in the two vertices to approximate the congestion of the edge that is incident to the two vertices. The ratio of "local net count" and the areas of two grid tiles is used as congestion metric in [95].

The congestion estimates generated from simple parameters is of low quality because these algorithms have inputs with limited information. The simplicity

of these metrics cannot capture the influence of technology scaling on congestion because they do not model low-level details such as the increasing number of transistors and the complexity of design rules. As a result, congestion estimation using simple metrics has not been used in modern routers [19, 21].

2.4.2 Probabilistic-Based Estimation

Lou *et al.* [74] proposed a stochastic estimation approach that calculates the utilization, $U(e)$, of an edge $e \in G$ from all possible routes that could be taken by each net and assigns to them an equal probability. The work of Westra *et al.* [111] adapted and extended the concept of Lou *et al.* [74] such that congestion analysis is based on the routing of two-pin nets (multi-pin nets are broken up into a set of two-pin nets). This work also restricted the total set of possible routing shapes per net to eliminate configurations that were deemed impractical or rarely-used based on observations made from six different industrial chips. As shown in Figure 2.5, for a two-pin pair, there are generally five shapes of routing.

To further explain the $U(e)$ calculation using these shapes, the "Short shape" is demonstrated in Figure 2.6. When two pins in a pin-pair are both located in one tile, it is called a "short shape" because it has the shortest wirelength among all other types. Given the coordinates of two pins, the $U(e)$ of x and y directions can be obtained by calculating the ratio of absolute differences of their coordinates and tile width and length, respectively. The *utilization* in the x-direction is $U_x(e) = \frac{x}{w}$, and in the y-direction is $U_y(e) = \frac{y}{h}$ for a tile of width w and height h. The utilization of other routing shapes can be obtained in a similar way.

Other papers, such as [96, 97], consider more routing shapes than Westra *et al.* [111] do. As designs change in sizes and technologies, the accuracy of the probabilistic method is not guaranteed. In practice, the actual shapes of routes are numerous. Therefore this probabilistic based congestion estimation ceases to work as the size and the complexity of the chip grow because more undefined shapes could potentially be created by automated routers.

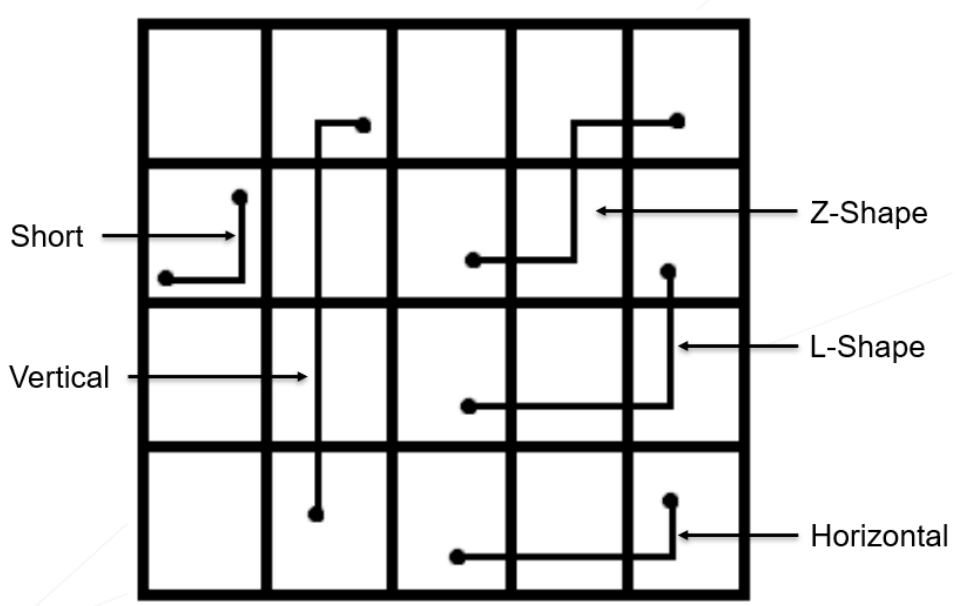


Figure 2.5: Illustration of different shapes of routing pin-pairs.

2.4.3 Global-routing-Based Estimation

The global-routing-based approaches [46, 48, 61] perform the actual routing on designs to estimate congestion. These approaches do not consider blockages, and do not try to optimize routability in order to reduce the runtime. An example is shown in Figure 2.7. Given a two-pin pair, the router connects the two pins and the resulting solution is used to calculate congestion. The congestion of an edge equals the average length of the wire in two incident tiles.

Mainstream global-router-based algorithms [46, 48, 61] have been proposed in recent years. Pattern routing [87] makes routes based on simple types similar to Figure 2.5. Saeedi *et al.* [93] present an estimation method which combines both the probabilistic and global-routing approaches. Global-routing based estimation approaches increase both the computational effort and run-time of calculating the congestion, because actual routing is carried out during the process.

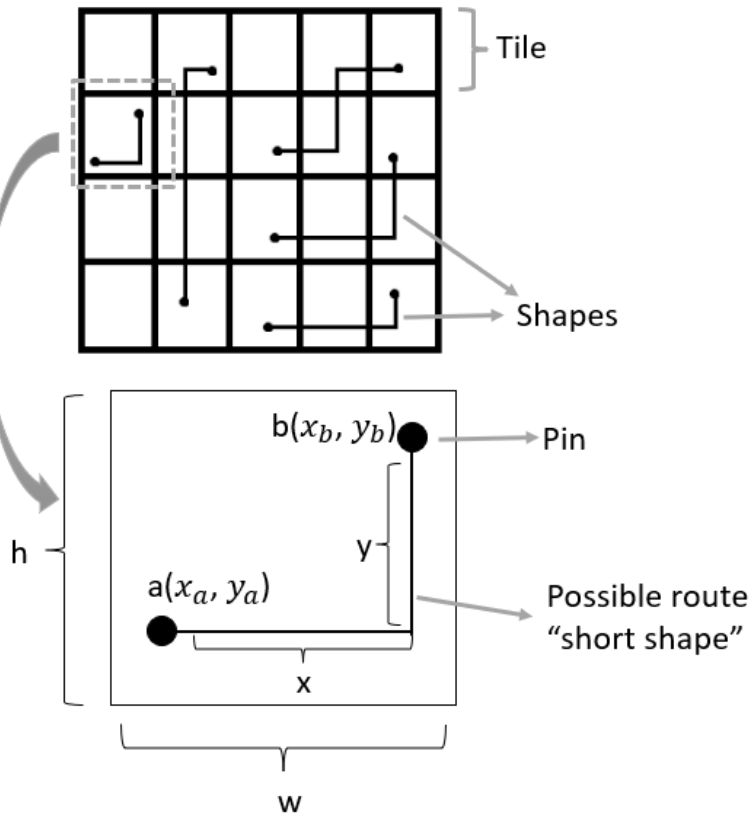


Figure 2.6: An example of a "short shape" two-pin pair.

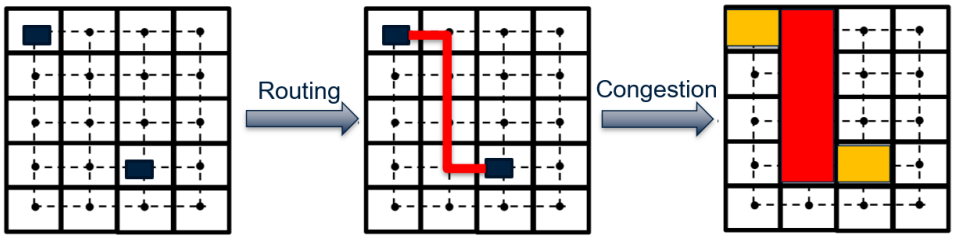


Figure 2.7: An example of the relationship between congestion and wire-length by using global-routing-based congestion estimation.

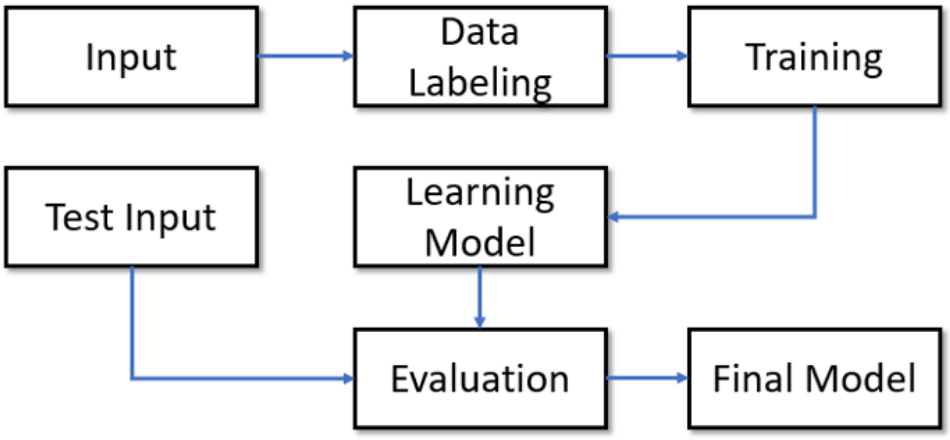


Figure 2.8: A generic ML flowchart.

2.4.4 Machine Learning-Based Estimation

Machine learning can develop congestion estimation models which are trained on ground-truth congestion data from actual routed circuits. A general ML algorithm is illustrated in Figure 2.8. For the use of congestion estimation, the training produces a model which attempts to predict the congestion using selected design characteristics, called features, that are known before routing. Features may include various physical information such as pin location and net density. Empirical experiments [85] show that hidden relationships between design factors, such as congestion, design constraints, or technology changes can be captured by machine learning networks. As discussed in [120], researchers have discovered the effectiveness of applying various machine learning techniques on EDA applications such as placement and routing.

Neural Network-Based Design Rule Checking (DRC)

Congestion can make pin-pairs unroutable, introducing DRC Violations in later verification stages. Therefore, predicting the congestion of an edge $e \in G$, $\lambda(e)$, becomes equivalent to predicting the DRC violations of two adjacent vertices that share the edge. This approach can be described as a DRC violation checker, as illustrated in Figure 2.9. The DRC violation prediction output is a binary classification:

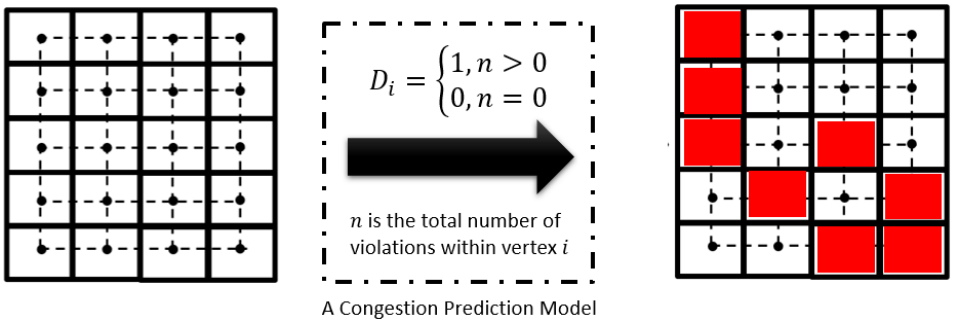


Figure 2.9: An example flowchart of a DRV checker.

a "1" implies the current vertex is predicted to have DRC violation(s); "0" implies the current vertex does not have DRC violation. The congestion of an edge $e \in G$ between two vertices $u \in V$ and $v \in V$ can be expressed as:

$$\lambda(e) = \begin{cases} 1, & \text{if } D_u \vee D_v = 1 \\ 0, & \text{if } D_u \vee D_v = 0 \end{cases} \tag{2.8}$$

where D_u and D_v are the Design Rule Violation (DRV) prediction output of vertices u and v, respectively. However, this transformation between the prediction of congestion and DRV causes a problem that the severity of congestion is not being considered as the DRC produces binary answers only (either "have" or "does not have" DRV). As a result, global routing algorithms that need to make decisions upon the severity of congestion do not obtain sufficient information from these binary classifiers.

Recent works [18, 107] proposed a DRV prediction model which determines whether a certain vertex has DRC violations. Xie *et al.* [115] proposed a DRC violation prediction algorithm using a fully convolutional network. Other [50, 68] proposed enhanced feature extractions and customized neural networks for DRV prediction. The work [89] takes the DRC violation prediction into consideration when performing routing, results show that DRC violation prediction does help to improve the detailed routing, but global routing is worsened with longer runtime.

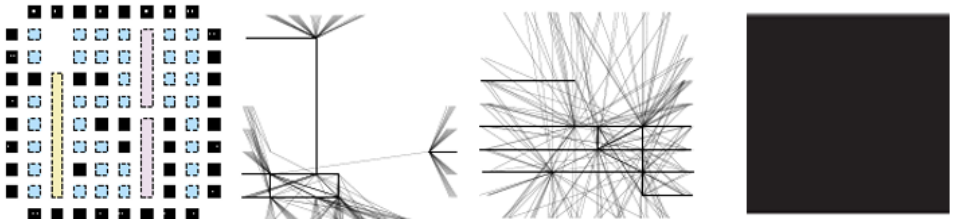


Figure 2.10: Connectivity image from Yu's [121], left is the circuit, the feature image changes as the complexity of the circuit grows and the image eventually turns into total black [10].

Neural Network-Based Congestion Estimation

Recent works such as CongestionNet [62] for Application-Specific Integrated Circuit (ASIC) and for Field-Programmable Gate Array (FPGA) [121] have been proposed to predict the severity of congestion instead of DRC violations. Different NNs are used such as Graph Neural Network (GNN) and GAN. Various features are extracted for predicting congestion. However, benchmarks used in these works are too small compared with public ones and the features extracted from those works can not scale with the design size and complexity. For example, Figure 2.10 shows a feature called connectivity from Yu's [121], it visualizes pin-pairs to be connected. As the design complexity and number of pin-pairs grow, this feature has been verified that it will eventually become a total black image [10]. Unscalable features cause an issue that can decrease the accuracy of the congestion estimation model when larger and more complex designs are used. Therefore, it is important to develop appropriate NNs and features that can scale with designs' size and complexity.

2.5 Summary of Prior Congestion Estimation Techniques

Simple-metric-based methods consider the least amount of factors when estimating congestion. On the other hand, probabilistic-based methods are alternative algorithms that take into account more parameters during congestion estimation. The congestion predicted on the assumption of fixed routing shapes runs two to three times faster than the global-routing based approaches [110]. Furthermore, if only one routing shape is allowed, comparing with global-routing based methods, the

probabilistic methods become faster and accurate for designs which are not congested [94]. Nonetheless, the probabilistic technique is known to be inaccurate in congested designs because it does not have the ability to consider various kinds of detour shapes. On the other hand, global-routing based approaches perform an actual routing, allowing the router itself to be used as the congestion estimator and therefore produce an congestion map that can be more accurate to the actual one after routing. This runtime-performance trade-off makes both probabilistic-based and global-routing-based algorithms have their markets and they are selected based on the needs of users and the complexity of designs.

The simple metric based, probabilistic based, and global-routing based congestion estimation are based on limited parameters or fixed routing shapes derived from empirical experiments. As discussed in [110], such methods cannot capture hidden relationships between design properties because it is difficult to manually create a model that can capture and apply all possibilities. For example, making a detour around an obstacle can be done in many possible ways, such as moving the route to another layer while maintaining the shape of the route, or expanding the routing region and re-routing. It is time consuming for a global-routing based algorithm to find such detours in congested circuits. However, finding complex patterns and relationships in quantitative data is empirically shown to be what ML does better than humans [56]. By using techniques such as supervised learning, congestion estimation models can be trained using data that comes from actual routing solutions that abide by the numerous and sometimes complex design and technology constraints.

Table 2.1: Relative accuracy and runtime ranking of different congestion estimation techniques

	Accuracy	**Runtime**
Simple Metric Based	3rd	1st
Probabilistic Based	2nd	3rd
Global-routing Based	2nd	4th
Machine Learning Based	1st	2nd

An accuracy and runtime ranking of different estimation techniques is shown in Table 2.1. The simple-metric-based method is the fastest when compared with others because the mathematical model of the simple-metric-based method is primitive. By primitive it means the model uses methods such as extrapolation to estimate congestion based on limited number of factors, such as number of pins. Therefore the prediction quality of simple-metric-based method ranks the worst among all other methods. The probabilistic and global-routing based method both have the runtime-accuracy trade-off. They have been deployed in commercial physical design tools for different market concerns [94]. They both produce congestion estimation with similar quality, however, the probabilistic based technique is usually faster than global-routing based ones because actual routing is not needed. The ML based approach is new in the field of congestion estimation and DRC prediction. It has the best accuracy among all methods from empirical data. The runtime range varies as there are many ML algorithms to choose from, such as MARS [36], Support Vector Machine (SVM) [27], and GAN [40]. Most of them are faster than probabilistic and global-routing based methods, but slower than the simple metric-based approach. ML-based techniques have been researched more and more in recent years [49].

Each category of the congestion estimation techniques suffers from different drawbacks, such as low accuracy for the simple-metric-based method and long runtime for the global-routing-based method. Among all these works, the Rectangular Uniform wire Density (RUDY) [101], a probabilistic-based algorithm, is the most commonly used estimation approach in academic global routers [19, 21, 59, 70]. It calculates congestion, $\lambda(e)$, of an edge e as the following:

$$\lambda(e) = \sum_{i=1}^{m} \frac{HPWL(i)}{Area(i)} \tag{2.9}$$

where the $HPWL(i)$ is the Half Perimeter Wirelength (HPWL) of a net i and the $Area(i)$ is the area of the bounding-box region that contains the entire net. Each net in this calculation must include e in its bounding-box, and m is the total number of nets of the design that include e. An example of a net and its bounding-box is shown in Figure 2.11; there is a multi-pin net consisting of 4 pins in the routing graph, and pins are represented by vertices. The locations of vertices form a

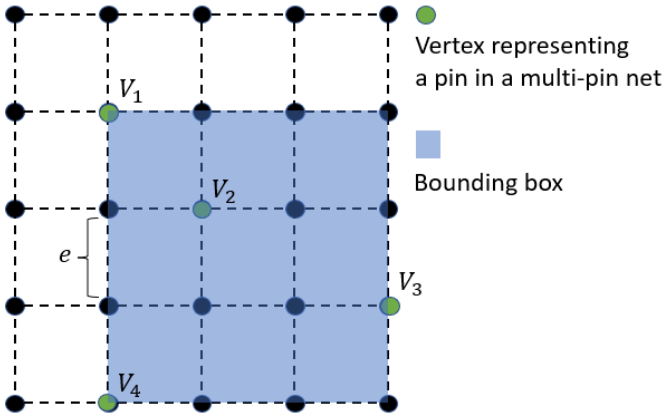


Figure 2.11: An example of a bounding box of a multi-pin net, and the net has 4 pins.

bounding box in such a way that all pins are either on the boundary of the box or within the area of the box. Assume the distance between two adjacent vertices is 1, therefore the HPWL is 6 and area of the bounding-box is 9. As a result, the congestion value of the edge e is $2/3$. RUDY has the fastest runtime compared with previously discussed probabilistic-based and global-routing based methods, and the accuracy is also the best among all of them [101].

This dissertation presents a set of ML-based congestion estimations that balance the trade-off between prediction accuracy and runtime. Moreover, novel feature extraction algorithms are also presented which address the scalability challenge that existing works have.

Chapter 3

Supervised-Learning Congestion Predictor For Routability-Driven Global Routing

In this Chapter, a machine-learning-based congestion prediction algorithm using Multivariate Adaptive Regression Splines (MARS) [36] is proposed and its implementation presented. This work shows the feasibility of applying ML for congestion estimation in routability optimization. The global routing results indicate that machine-learning-based methods can provide benefits to the back-end physical design.

3.1 Introduction

As discussed in Section 2.4, in the global routing phase, the congestion estimation algorithms found in routers tend to be either probabilistic-based or global-routing-based. Researchers have shown that these traditional estimation methods may not be sufficient to capture hidden relationships between design properties [110]. Using ML algorithms [92], however, a congestion estimation model can be trained to accurately estimate the congestion of new circuits based on example designs that contain actual routing solutions that abide by the numerous and complex design and technology constraints.

The MARS approach can construct a congestion estimation model which considers different information within each routing graph vertex, such as the number and locations of local pins and nets of each vertex.

3.2 Methodology

The MARS algorithm needs to be trained to create an accurate predictive model. For the purpose of training, certain features, served as inputs, are extracted from designs that have already undergone placement and placement only. Along with these input features, post-routing congestion information for each edge are also extracted to form a ground-truth output label. Based on input features and output label, one matrix and one vector are generated. The matrix contains the features and vector the congestion information. A trained MARS model is used to predict congestion and is implemented in the NTHU-Route 2.0 [19] platform.

3.2.1 Design Data Transformation

The key challenge in applying ML to EDA applications is how to map a circuit into a format readily "understandable" by ML algorithms chosen. In my case, I propose to transform the routing graph and its associated design properties into a "matrix-vector pair" format for the ML algorithm chosen, MARS.

The design data in the netlist are extracted and transformed in such a way that its routing graph is rearranged and represented by one matrix and one vector. The routing graph $G = (V, E)$ introduced in Section 2.1 is rearranged into the format shown in Figure 3.1. All edges $e \in G$ are queued up to form the rows for both the input matrix and output vector. The sequence of selecting the edges from G is decided by the order of graph traversal, which is determined by the c++ Standard Template Library (STL) default vector class setting [84]. The number of columns of the input matrix equals the number of features extracted, the number of rows corresponding to the number of edges in the design graph G, and the output vector corresponds to the congestion information of each edge. The values in the output vector are post-routing real congestion information of a design during the model training process. All elements of the vector are set to zero if the model is trained and used for congestion estimation, the estimated congestion for all edges $e \in$

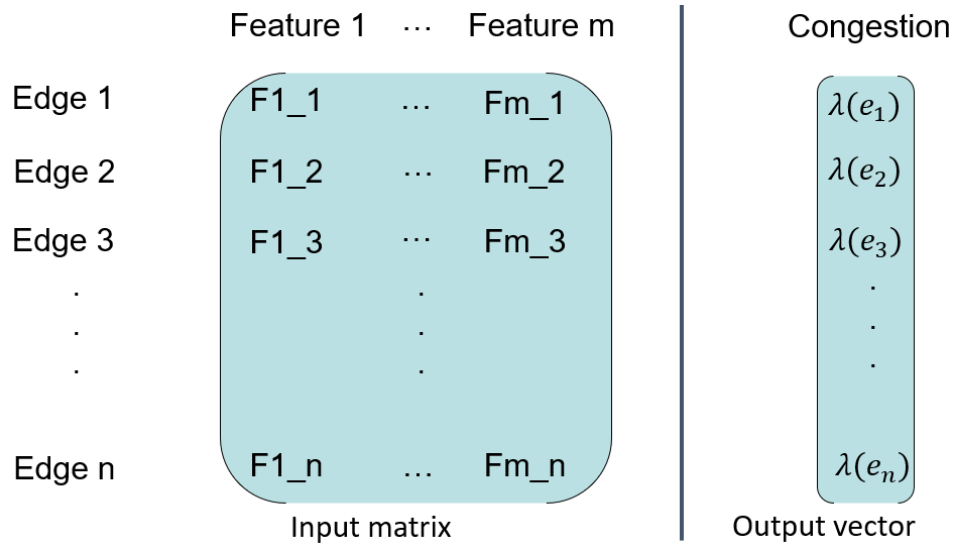


Figure 3.1: Design data transformation for MARS.

G will update the values of corresponding rows of the vector based on the input matrix.

3.2.2 Routing Feature Extraction

Four features considered relevant for causing congestion are extracted. The values of these features are stored in the input matrix discussed in Section 3.2.1. These features are the following:

- **Number of first-degree pin(s):** $P_{fd}(e)$ is the feature corresponding to the number of pins closest to an edge. To route a $first-degreepin$, at least one of the four edges surrounding that pin will be crossed by the wire. These pins are deemed to have a significant impact on the probability of the routing wire to cross-over the chosen edge, thus altering its congestion value. Figure 3.2(a) depicts a visual representation of this feature, in total of four pins are located in the two adjacent tiles that share the edge X, and three Y. Therefore, $P_{fd}(X) = 4$ and $P_{fd}(Y) = 3$. This feature provides a metric for the proximity or vicinity of pins with respect to an edge.

- **Number of first-degree net(s):** The *first-degree net(s)*, $N_{fd}(e)$, of an edge

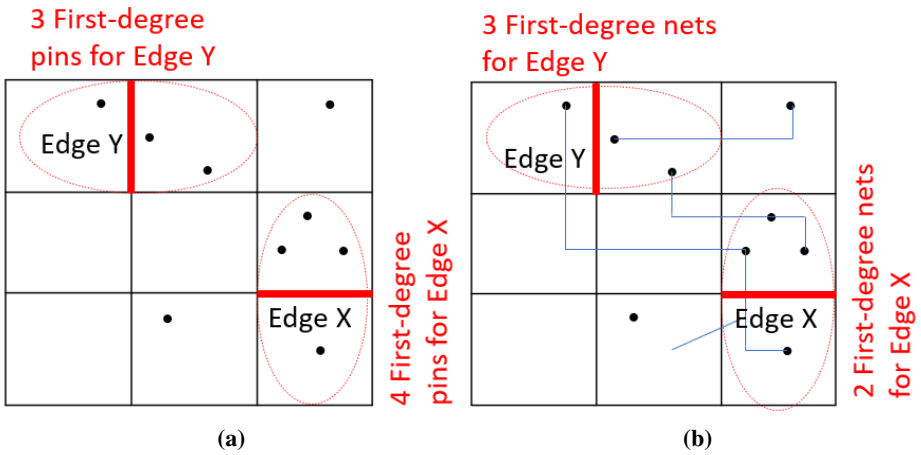


Figure 3.2: Examples of (a) first degree pins; (b) first degree nets.

are those nets that pass either of the two tiles adjacent to the edge. A tile can contain various pins which belong to different nets. Different pins in a tile can be a part of the same net or part of different nets. Figure 3.2(b) gives an example of this feature, two nets are drawn in the two adjacent tiles that share the edge X, three the edge Y. Therefore, $N_{fd}(X) = 2$ and $N_{fd}(Y) = 3$.

- **Pin density:** $P_d(e)$, gives density information regarding the number of pins of each edge. The $P_d(e)$ for a chosen edge e is defined by the number of *First-degree pins* of that edge, $P_{fd}(e)$, divided by the area summation of two tiles that this edge is a part of, for example, $P_d(e) = \frac{P_{fd}(e)}{Area(tile_1)+Area(tile_2)}$. If the areas of all tiles of a design are the same, then $P_d(e) \equiv P_{fd}(e)$. However, in practice, designs may have grids with different tile areas [2, 3], therefore this feature provides information of how crowded the pins are around an edge. The higher the pin density, the higher the probability that this edges will be frequently occupied by interconnects, in other words, a higher pin density implies that the edges has a higher chance of producing congestion.

- **RSMT-aware two-pin path Availability:** The *availability* of a path i, $A(i)$, represents how "good" this path is among all candidates when routing a two-pin pair during the global routing phase based on a probabilistic model. The algorithm for extracting this feature is a simplified version of a probabilis-

tic model [123] (the complete description of this model is presented in Appendix C). Here, "simplified" means that my extraction algorithm used for MARS has fewer routing types (only L-shapes used) compared to the full probabilistic model (5 routing shapes) in Appendix C. From Westra's [111] only 1.2% of the two-pin nets had more than two bends, therefore only L-shapes are considered as candidates and their availability features are calculated accordingly. An example is shown in Figure 3.3, given the location of a two-pin pair, various paths can be constructed from one end to the other. such as *path1* and *path4*. Subsequently, the availability for choosing a path i, $A(i)$, is calculated. It depends on the maximum capacity of each edge lying in the chosen path. More shapes can provide enhanced congestion estimation accuracy, but at the expense of a computational time penalty.

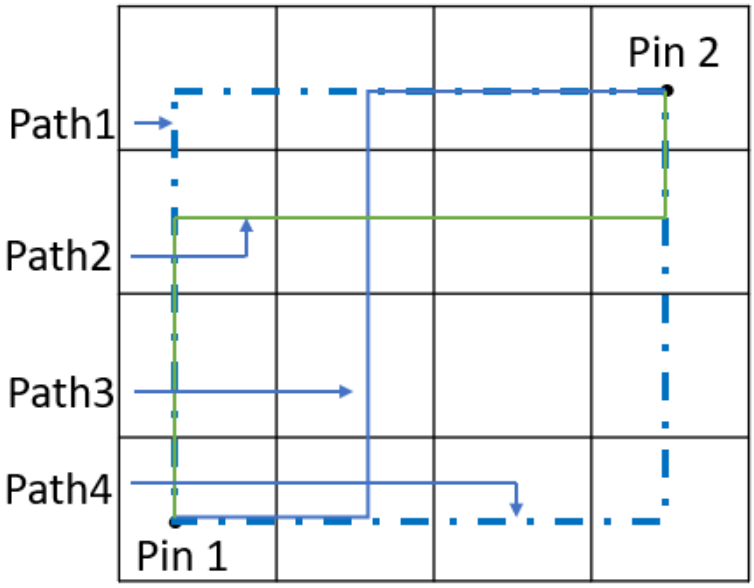


Figure 3.3: Four possible paths for a two-pin net.

Let e^i_j present one of the edges, j, that constitutes a path i between the two-pin pair in Figure 3.3, and let $C(e^i_j)$ be the maximum capacity for an edge

e^i_j, then the total availability, $A(i)$, for the path i, can be written as:

$$A(i) = \begin{cases} \sum_{j=1}^{n} C(e^i_j) & C(e^i_j) \neq 0, \forall C(e^i_j) \\ 0 & C(e^i_j) = 0, \exists C(e^i_j) \end{cases} \tag{3.1}$$

if any edge e^i_j in the path i has no capacity, the entire path i is eliminated from candidacy of being selected for routing, therefore the availability of path i, $A(i) = 0$. Otherwise, the availability of path i equals to the capacity summation of all n edges that constitute the path i.

An example of calculating the availability of candidate paths of a two-pin pair is shown in Figure 3.4. There are two possible paths, $Path\_1$ and $Path\_2$, for routing the pin-pair, where two pins are located at the lower left and upper right corners. The transformed routing graph shows the *capacity* of each edge those two paths cross, the *availability* of two paths can be obtained by adding capacities of edges that those two paths cross:

$$\begin{aligned} A(Path\_1) &= 20+20+10+18+18+18 = 104 \\ A(Path\_2) &= 20+10+20+20+20+18 = 108 \end{aligned} \tag{3.2}$$

Following the computation of availability, the probability of each path is calculated. Each edge e^i_j in path i is assigned the probability P_i which reflects its likelihood for being used in global routing. For example, if there are k candidate paths between a two-pin pair, the probability of a path i can be calculated as follows:

$$P_i = \frac{A(i)}{\sum_{n=1}^{k} A(n)} \tag{3.3}$$

Using the same example in Figure 3.4, after obtaining the *availability* of two paths, $A(Path\_1) = 104$ and $A(Path\_2) = 108$. The probability of choosing path $path\_1$ is $P_{Path\_1} = 104/(104+108) = 0.49$, and $P_{Path\_2} = 0.51$. These two probabilities show that $Path\_2$ has a slightly higher chance (2% higher) to be chosen to route the pin pair, but $Path\_1$ is still under consideration for being chosen by the router, because it is not 0%. In order to reflect this situation in the congestion estimation algorithm, the existing usage is transformed using the probability as a

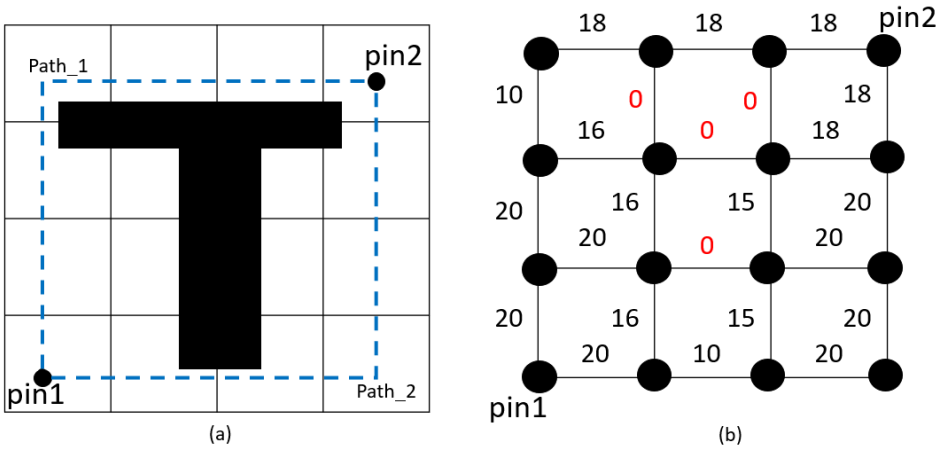


Figure 3.4: (a) A design grid with a T-shape blockage and two possible routing paths; (b) The design is transformed into a routing graph and the number close to each edge indicates the *capacity* of these edges.

weight, the new "weighted usage", U^*, is expressed as follow:

$$U^*(e^i_j) = U(e^i_j) \times P_i \tag{3.4}$$

3.2.3 Model Training

MARS is a flexible regression framework that fits equations to data by searching for interactions and non-linear relationships between input features and output labels. The reasons for the selection of MARS are as follows: 1) non-linearity between features can be captured by the MARS algorithm; 2) MARS has fewer parameters than most NN algorithms; and 3) the parameter tuning of the model is automatic. One of the core mechanisms from MARS is called a hinge functions. A hinge function $h(x-t)$, shown in Equation 3.5, is a function whose output value equals to its constant parameter when this constant parameter is greater than 0, and 0 everywhere else.

$$h(x-t) = \begin{cases} x-t, x > t \\ 0, x \leq t \end{cases} \tag{3.5}$$

where x is a input variable (in the case of this congestion estimation model, one

of the input features); t is the constant parameter which defines when the function becomes zero.

MARS creates a predictive model that is a linear combination of basis functions, each of which is a summation of one or more of the following: (1) a constant; (2) linear functions of input variables; and (3) hinge functions of input variables. An example basis function is expressed as follow:

$$y(x) = 5h(1-x) + \frac{1}{5}h(x-1) + 2 + 3x \tag{3.6}$$

where the basis function, $y(x)$, contains the hinge functions $5h(1-x)$ and $\frac{1}{5}h(x-1)$; a constant 2; and a linear function of $3x$. An predictive model that MARS creates can be written as:

$$M(X) = \sum_{\substack{i=0 \\ j=0}}^{n,m} y_i(x_j) \tag{3.7}$$

where $M(X)$ is the predictive model of input matrix X, which contains a total of m input variables. $M(X)$ is the linear combination of a total of n basis functions.

The MARS predictive model creation algorithm has two stages. First, MARS will produce an arbitrary number of basis functions for the input-output pair, with the objective of locally minimizing the squared error loss of the features and the congestion. The second step is called a pruning pass, which selects from the basis functions and finds those that produce a locally minimal Generalized Cross-Validation (GCV) score [76]. The GCV is a way of calculating an approximation of the prediction error.

To train the MARS model, one design is randomly chosen as the training set. The features mentioned in Section 3.2.2 and output label (congestion) of that design are extracted and transformed into the input matrix and output vector which are later fed into the MARS [36] algorithm. When the model is generated, it will be used to test designs that are not used for training. By doing this, it is ensured that the model remains unchanged when making congestion predictions for different designs. However, to predict the congestion for the design used as the training set, another model is trained using a design other than the one previous used.

A MARS model trained using design n6 in the benchmark suite [6], is shown

Table 3.1. There are in total of 6 terms in the trained model. The intercept is a constant and there is just one such term as intercept and the value is always 1 by definition. Feature variables $x0$ to $x3$ correspond to first-degree pin, first-degree net, pin density, and availability, respectively. The trained MARS model can be expressed as:

$$M(X) = 95.3977 + 0.497101x1 - 2.09381h(x3 - 215) - 0.412473h(215 - x3) + 2978.22h(x2 - 0.0444898) - 68.893h(0.0444898 - x2) \quad (3.8)$$

An interesting observation is that this model excluded the number of first-degree pin feature under the default settings, it implies that MARS considered $x0$ is not significant enough to be included in the model for predicting the congestion of design n6. The threshold of feature significance consideration can be manually changed, it is possible that a trained model can include all input features or fewer than the example given. Depending on the data used for training the model and the seed used for generating random initial terms, the coefficients of a model would not always be the same even if the training dataset and settings stay unchanged, therefore there could exist various different trained models when using design n6.

Table 3.1: The composition of a MARS model.

Basis Function	Coefficient
Intercept	95.3977
x1	0.497101
h(x3-215)	-2.09381
h(215-x3)	-0.412473
h(x2-0.0444898)	2978.22
h(0.0444898-x2)	-68.893

3.3 Experimental Setup

After the MARS model is generated, the predicted congestion from the model can be used in two stages: RSMT edge shifting and initial routing. In the following

content, these two stages are introduced and how MARS model plays a role in such stages is explained.

3.3.1 Edge Shifting

The edge shifting technique [87] is a tree topology optimization. It is done by shifting the edges of the RSMT [51]. The edge can be moved by adjusting the positions of both ends. The position of circuit pins cannot be replaced at any stages other than placement, therefore existing Steiner points of the RSMT will be adjusted by moving, deleting or inserting Steiner points to the topology. An edge shifting example is shown in Figure 3.5. For a given RSMT shown in Figure 3.5(a), there is only one Steiner point shown on the right side with a connection to three pins. In order to perform edge shifting, one end of the edge-to-shift must be connected to this Steiner point. The available range of moving the Steiner point in the vertical direction is shown in Figure 3.5(b). During the shifting, another Steiner point is created. Figure 3.5(c) is an example that the shifted edge has escaped the congested region while Figure 3.5(d) shows an edge that has not.

The edge shifting technique is applied after the RSMT construction algorithm [25], and the shifting uses congestion information from MARS model to optimize the RSMT before routing. The RSMT determines how pins in multi-pin nets are paired in two, based on which global routing is performed. It implies that the quality of the congestion map affects both the RSMT construction and the global routing. The congestion estimation information from the MARS model is inserted into the router before edge shifting takes place, therefore the RSMT quality can reflect the impact of the model. The evaluation of the quality of the routing topology is discussed in Section 3.4.2.

3.3.2 Initial global routing

The initial global routing stage is another stage that uses congestion estimation information in NTHU-Route. The router will generate a initial global routing solution from scratch based on the RSMT; the estimates act as a guidance that prevents the router from routing in potential congested areas.

Two initial global routing flowcharts with and without the use of MARS model

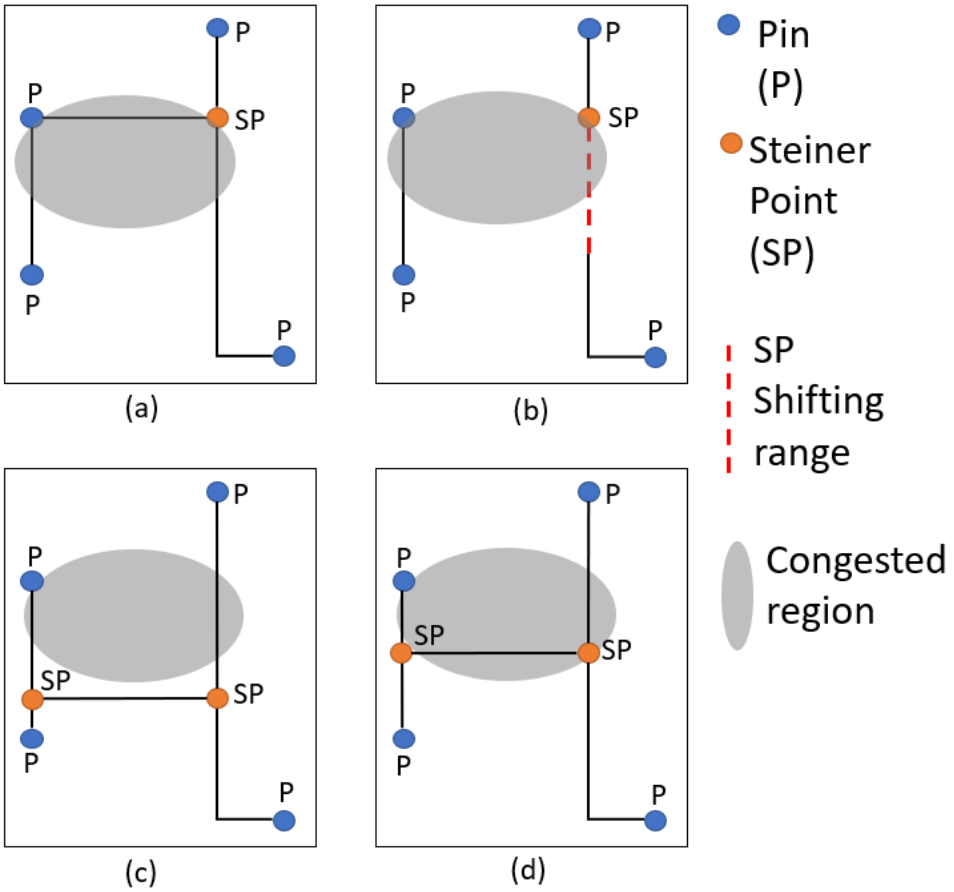


Figure 3.5: (a) The original Steiner tree to perform edge shifting; (b) The available range of moving the Steiner point in the vertical direction; (c) A new tree structure with a shifted edge that has avoided the congested region; (d) Another new tree structure where the edge has failed to escape the congested region.

are shown in Figure 3.6. The original congestion estimation is obtained by using pattern routing [87] after FLUTE [25]. The problem is that the original global-routing-based congestion estimation runs slower when Steiner points are introduced; the more Steiner points there are, the more two-pin pairs to route. For a total number of p two-pin pairs, the time complexity is $O(p)$ for accessing all pairs. However, the routing of those pairs take extra time and it's the dominant

part in the router's original congestion estimation. After the congestion estimation using pattern routing is finished, the edge shifting is performed. After RSMTs are updated the initial routing is invoked to generate the initial routing solution.

Unlike the original flow, the MARS flow congestion estimation is obtained by two steps. First, features mentioned in Section 3.2.2 are extracted based on RSMTs, then the MARS model is then called to predict congestion and to update the routing graph. During the feature extraction step, all features except availability are extracted by traversing edges of the graph once. The time complexity of extracting those features is $O(nm)$ regardless of the number of two-pin pairs, where n and m are the respective number of edges at the horizontal and vertical boundaries of the design graph. The availability calculation considers only L-shape routes for two-pin pairs and has the complexity of $O\{p*(n+m)\}$, and according to the benchmarks used, $p*(n+m) < nm$. However, even though complexity-wise the feature extraction $O(nm)$ is slower than traversal of two-pin pairs, $O(p)$, the overall MARS congestion estimation is still faster than performing a time-expensive routing for after traversing all two-pin pairs, this hypothesis is confirmed by experimental results in Section 3.4.1.

After the initial routing solution is generated, the global routing will enter an iterative process called the rip-up and re-route. This process aims at reducing the congestion caused by the initial routing; the estimated congestion will no longer be used after initial routing.

3.4 Experimental Results

For experimentation of the congestion estimation quality, the ISPD'08 benchmarks [6] were used. The experiments were run on a Linux workstation with 4-core 2.6GHz CPU and 8GB memory. The router originally uses L-shape as the only type in the probabilistic model to calculate the congestion map. Therefore the 2-pin path availability feature in the MARS is set to consider L-shapes only. The difference with and without MARS predictive model will be measured in terms of runtime, quality of the initial routing solutions, and quality of the final routing solutions. Quality of routing solutions is measured by wirelength and number of overflows.

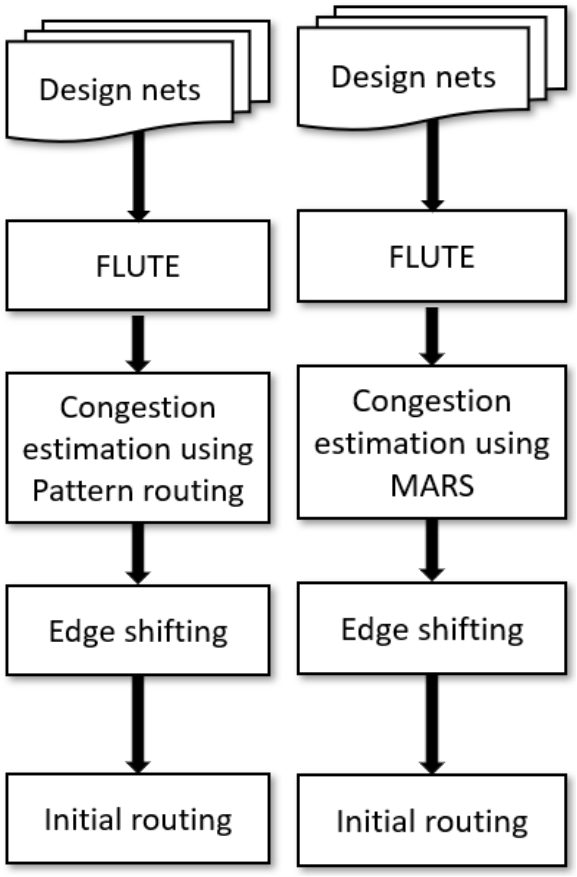


Figure 3.6: Flowcharts of (a) original and (b) revised initial routing algorithms. Both are implemented using NTHU-Route [19].

3.4.1 Runtime Comparison

Two runtime experiments were performed: using (1) the original congestion estimation used in NTHU-Route 2.0 [19]; and (2) the proposed MARS predictive model. The runtime comparison is shown in Table 3.2. The prediction runtime of NTHU-Route 2.0 shows how long the L-shape probabilistic model takes to estimate congestion for each benchmark. The feature extraction column shows the runtime the proposed MARS algorithm takes to extraction features. The prediction column of MARS provides the runtime of how long the predictive model takes

to generate congestion estimation for each benchmark. The normalization is calculated using the runtime summations of each benchmarks. The MARS achieve an average speedups of $100\% \times 1 - (1/17.45) = 94\%$, compared with NTHU-Route's congestion estimation method.

3.4.2 Congestion Estimation on Routability-Driven Edge Shifting

The edge shifting technique optimizes the RSMT based on a estimated congestion map. A better congestion prediction could result in a tree topology with fewer total overflow (TOF). Therefore, the quality of congestion estimations can be compared by evaluating the initial global routing solutions with different optimized RSMTs. Both the original congestion estimation and the MARS are used to optimize the RSMTs of all benchmarks in the experiments.

The experimental results in Table 3.3 shows the quality impact of RSMT edge-shifting optimization using different congestion estimations on the initial routing solutions. Routing algorithms are the same for both NTHU-Route 2.0 [19] and MARS [124]. The columns are the number of wirelength (WL), max overflow (Max_OF), and total overflow (TOF) of the initial routing solution. The normalization in the last row is calculated using the summation of respective metrics of each design. Max_OF is the largest overflow of one edge over the entire design; this metric provides the information about the worst-case (most congested region). Compared with the MARS model, the RSMT optimized with the router's original algorithm ended up having 18% more Max_OF and 11% more TOF. The reduction of MAX_OF using MARS shows that the most congested regions of these RSMTs have been improved, and the reduction of TOF using MARS shows that the overall congestion has improved as well. The WL has not changed a lot ($< 1\%$), which implies that the congestion are reduced not by making detours, but by more efficient routing due to the MARS congestion estimation.

3.4.3 Global Routing Performance

The impacts of using MARS [124] predictive model in two situations (on edge-shifting and initial routing) have been shown and discussed in the previous subsections. In NTHU-Route [19], the global routing process consists of two parts: rip-

Table 3.2: Congestion estimation runtime comparison

Benchmark	NTHU-Route 2.0 [19]	MARS		
	Prediction (s)	Feature Extraction (s)	Prediction (s)	Total (s)
a1	6.49	0.42	0.11	0.53
a2	5.81	0.35	0.13	0.48
a3	20.36	0.63	0.5	1.13
a4	20.62	0.58	0.51	1.09
a5	19.08	0.89	0.22	1.11
b1	4.93	0.31	0.05	0.36
n1	4.74	0.39	0.2	0.59
n2	9.85	0.47	0.20	0.67
n5	34.5	1.18	0.35	1.53
n6	24.06	0.96	0.17	1.13
Norm.	17.45	n/a	n/a	**1.00**

Table 3.3: The quality of initial global routing solutions using RSMT topologies optimized by different congestion estimations

Benchmark	NTHU-Route 2.0 [19]			MARS [124]		
	WL $\times 10^5$	**Max_OF**	**TOF** $\times 10^4$	**WL** $\times 10^5$	**Max_OF**	**TOF** $\times 10^4$
a1	33.91	112	24.55	33.91	77	20.33
a2	32.07	93	15.23	32.07	92	13.27
a3	93.46	80	59.31	93.46	75	49.48
a4	88.91	75	22.02	88.92	63	13.49
a5	95.05	132	60.36	98.05	116	55.91
b1	34.33	76	21.87	34.33	66	21.70
n1	23.25	52	5.25	23.26	43	5.28
n2	46.04	89	7.73	46.04	67	5.45
n5	142.31	155	62.77	142.30	123	55.66
n6	97.56	103	47.56	97.56	106	54.49
Norm.	**1.00**	1.18	1.11	**1.00**	**1.00**	**1.00**

Table 3.4: Result of final global routing performance before post-processing

Design	NTHU-Route 2.0 [19]					MARS [124]					
	#Iteration	Max_OF	TOF	WL$\times 10^5$	Time (s)	#Iteration	Max_OF	TOF	WL$\times 10^5$	Time (s)	Time %Change
a1	11	2	170	35.96	651	11	2	166	35.94	647	-1%
a2	12	2	197	33.07	224	12	2	175	33.06	223	0%
a3	8	2	177	94.72	608	8	2	128	96.70	561	-8%
a4	4	4	116	89.70	119	4	4	115	89.67	77.4	-36%
a5	14	2	173	103.08	1844	14	2	143	103.07	1774	-4%
b1	16	2	143	37.19	1407	15	2	198	37.17	1473	4%
n1	23	2	191	23.99	957	23	2	184	24.03	1051	9%
n2	3	4	197	46.43	52.9	4	4	112	46.43	43.5	-12%
n5	16	2	182	147.02	1924	16	2	174	147.01	1925	0%
n6	17	2	172	102.71	3229	17	2	172	102.71	3224	0%
Norm.	1.00	1.00	1.10	1.00	1.00	1.00	1.00	**1.00**	1.00	1.00	**-4.80%**

up & reroute phase and post-processing phase. The post-processing is an action to erase the *history* cost, $h(e)$, of all edges from the rip-up & reroute phase. Each time an edge is visited to construct a possible route, $h(e) = he(e) + 1$ regardless of the edge is eventually used or not. The repetitive visiting but not selecting of an edge would cause that edge to have a high $h(e)$ but low utilization, $U(e)$. The resetting of $h(e)$ after rip-up & reroute makes such edges valid routing resources again. The post-processing phase does not use any estimates because congestion at this stage is based on real wire overflows. Therefore, the outputs of rip-up & reroute before post-processing is compared to illustrate the difference of final global routing between using original congestion estimation and MARS predictive model in NTHU-Route. The TOF threshold is set to 200, which means the router will proceed to post-processing when TOF is below 200 during main-routing iterations. The results

are shown in Table 3.4. The "#Iteration" column shows the total number of rip-up & re-route iterations the router takes to finish routing the benchmark. The next three columns are the number of wirelength (WL), max overflow (Max_OF), and total overflow (TOF) before post-processing. The time column shows the runtime the router takes to finish the routing before post-processing. The "time %change" column shows the normalized runtime difference between global routing of each design with or without MARS congestion estimation. The normalization in the last row is calculated using the summation of respective metrics of each design. The data evidence from Table 3.4 demonstrates that the TOF of all designs using the original congestion estimation are on average 19% larger than the corresponding ones using the MARS congestion prediction. Moreover, an average 4.8% shortening of required runtime is achieved.

3.5 Conclusion

The global routing optimization problem using supervised-learning based algorithm was explored. A regression algorithm, MARS, was used. Four input features were designed and extracted for training a congestion prediction model. The model was embedded into RSMT edge shifting and the global routing solution using this optimized topology is improved in terms of having fewer Max_OF and TOF. The experiments showed that the predictive model is on average 94% faster than conventional congestion estimation method used in NTHU-Route 2.0[19] when applied on the ISPD'08 benchmark suite [6].

The model was also implemented in a global router to evaluate the effectiveness of the proposed algorithm.The routing quality is improved by having 10% less TOF under the same level of routing effort when compared with the routing using its original congestion estimation. Fewer TOF in the rip-up & reroute phase implies that the router spends less time on finding a routing solution with no TOF, hence shortening the overall routing runtime. The router achieved an overall 4.8% runtime speedup in global routing using MARS predictive model. Current industry designs are large and can take weeks to finish the entire design cycle, even a small percentage of runtime reduction such as 4.8% can provide benefits significantly.

Chapter 4

Congestion-Aware Global Routing using Customized Deep Convolutional Generative Adversarial Networks (c-DCGAN)

The input features and output labels of MARS model presented in the previous chapter are stored in matrices and vectors. The structure of these matrices and vectors do not relate to the layout structure of the design, and therefore the spatial information of the chip layout is missing when using MARS to make congestion estimation. This problem is addressed in this chapter. A novel congestion estimation algorithm which uses an advanced neural network called Customized Deep Convolutional Generative Adversarial Network (c-DCGAN) is introduced; an Data-Image Translator (DIT) is designed to resolve the data compatibility issue between the circuit design and c-DCGAN. In this chapter, the c-DCGAN structure and the DIT is presented and the experimental setup is explained. Lastly, the results are provided and discussed.

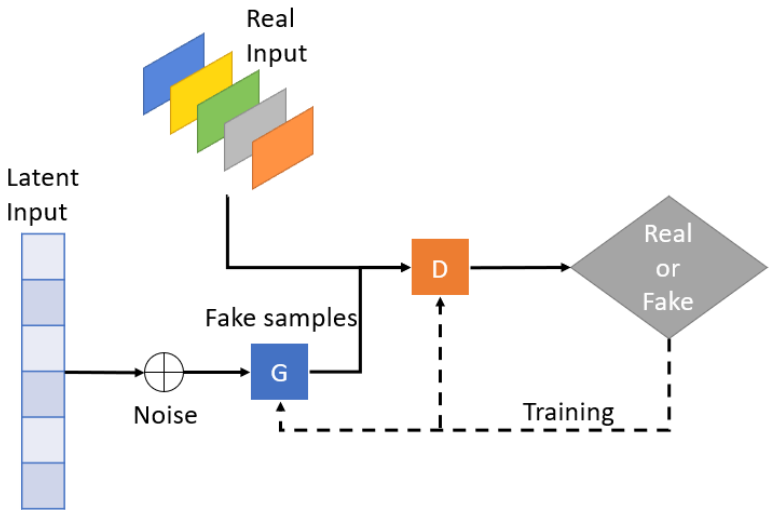


Figure 4.1: General structure of GAN.

4.1 Introduction

GANs [40] have produced good results in many fields that require data creation and predictions. My work here constitutes an exploration of the potential for GANs to alleviate the SoC global routing process. A GAN, as demonstrated in Figure 4.1, trains two networks simultaneously: a **generator** model G and a **discriminator** model D. These two models compete with each other in such a way that G tries to generate fake samples which are as close to the real ones as possible, to fool D. Meanwhile, D determines if a sample is from the real training dataset rather than a fake one produced by G. The fake sample is generated from hidden information that the network has no control of, called "Latent Input". This latent input is partially combined with random values, called "Noise", to form the final "Fake" sample. After training has converged, the **generator** is expected to produce outputs with a feature distribution of high similarity to the real dataset. Eventually, the **generator** will be used as the predictor for congestion estimation.

This chapter provides technical details on: 1) a novel congestion estimation methodology, called Customized Deep Convolutional GAN (c-DCGAN), is described, which learns from physical properties of actual routed designs and is scalable with the size of designs. The c-DCGAN predicts congestion such that place-

ments can be identified as routable or not before actual routing is performed; this chapter also describes an image encoding solution, which interprets routing data as image patterns. A "bridge" algorithm is designed to generate the image-based data used by c-DCGAN from the placement-oriented and netlist-oriented data created by the EDA tools.

4.2 Approach

Design benchmarks needs to be transformed into an image-like format so that it can be fed into the c-DCGAN, either to train the model or to make predictions using a trained model. The data transformation and the c-DCGAN structure are described next.

4.2.1 Data-Image Translator (DIT)

In the EDA field, there exists different tools and data formats, some of which are proprietary or incompatible with each other. Different routing tools use different design file formats such as "bookshelf" [1] or "LEF/DEF" [99] for sharing design information. To allow the integration of the c-DCGAN with standard tool frameworks, a Data-Image Translator (DIT) was developed to transform the input design data as well as output congestion information between image and design file formats. This allows bidimensional exchange of relevant data with the congestion estimation c-DCGAN. The DIT provides a solution for using the proposed c-DCGAN for global routing with the variety of routers available.

The DIT is described using a simple example design in Figure 4.2. In this example, the design's routing grid consists of 4 tiles, along with which are 12 edges (8 outer boundary edges and 4 inner edges shared by 4 tiles) and 9 intersections of horizontal and vertical grid lines. To encode such grid information, each tile will be replaced with a 2×2 pixel thumbnail, where each pixel has 3 channels (red, green, and blue). The top and right most edges have an extra column and row of pixels to hold boundary edge and intersection information. Each pixel holds feature information corresponding to the tile, to edges shared with other tiles, or to the intersection of tile edges. Finally the 2×2 routing grid becomes a 3×3 image after aligning pixels in rows and columns as shown in Figure 4.3.

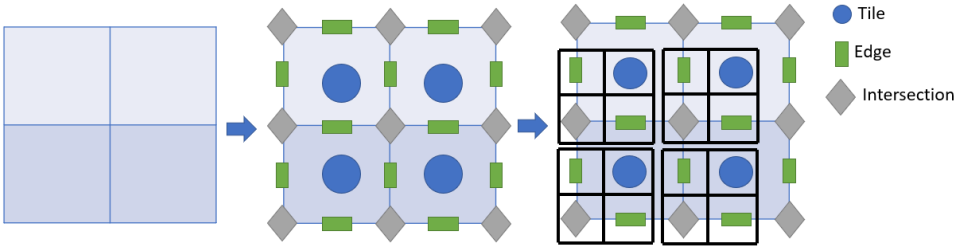


Figure 4.2: An example of feature encoding using a 2×2 routing grid.

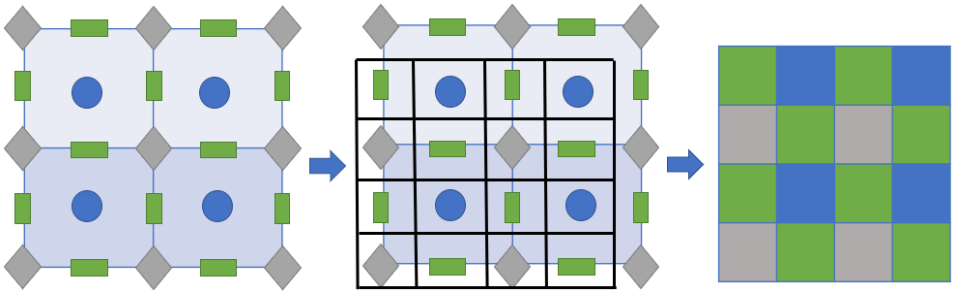


Figure 4.3: An example translation from a 2×2 routing grid with encoded features (left) to an image (right).

4.2.2 Data Encoding and Labeling

The congestion information of benchmarks after the final global routing is selected as the training target, or output label. For input features, the c-DCGAN only needs a minimum amount of netlist information: (1) *pin density*, (2) routing channel *capacities*, and (3) *net density*. The *pin density* is calculated in the same way as described in Section 3.2.2. The routing channel *capacities* reflect edge *capacities* affected by the placement of macros instead of existing interconnect wires. Macros are blocks of intellectual properties that are pre-existing on the chip layout. *Net density* is computed in the same way as Rectangular Uniform wire DensitY (RUDY) [101]. Visualizations of input features and output target are shown in Figure 4.4.

The availability feature used in Chapter 3 is excluded from c-DCGAN for two reasons: (1) the availability of one edge was manually calculated in such a way that it considers routing information of surrounding edges to compensate the draw-

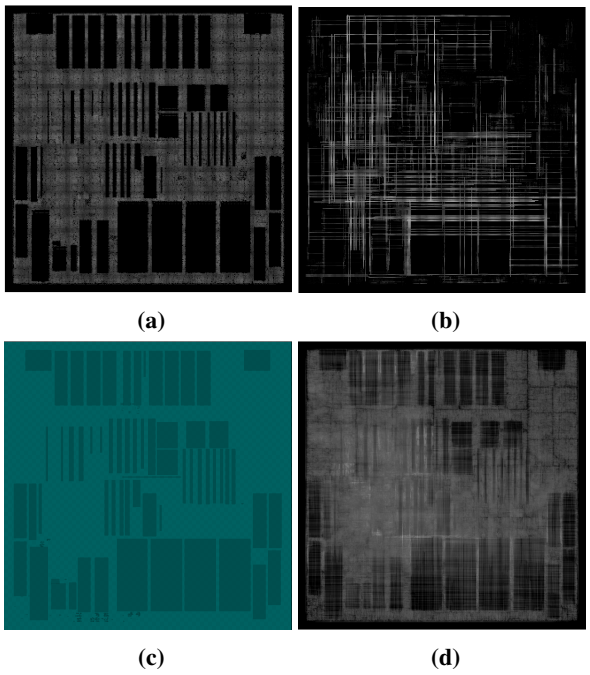


Figure 4.4: Design data in image format: (a) Pin density; (b) Net density; (c) Routing channel capacity; and (d) Congestion heatmap.

back of MARS, because MARS cannot directly take into account spatial information. c-DCGAN has no such problem because the input features are organized as images, and (2) the extraction of availability feature has the highest computational complexity compared with other features, therefore it is removed to accelerate the overall feature extraction process.

Each image pixel in Figure 4.3 has circuit information encoded into its RGB channels as follows. First, the red channel of edge pixels is to hold the ground-truth congestion values after global routing; the red channel will always be zero in all other pixels. Moreover, this red channel is forced to zero in all input images during the prediction phase, while the estimated congestion appears in this channel in the output of the c-DCGAN. In the 2×2 block of pixels, the green and blue channels are used to encode information about pin density, netlist density, and routing channel capacities as follows:

$image[2x][2y].g = image[2x][2y].b = 0$

$image[2x][2y+1].g = Vcapacity[x][y] + Vnetdensity[x][y]$
$image[2x][2y+1].b = Vcapacity[x][y]$
$image[2x+1][2y].g = Hcapacity[x][y] + Hnetdensity[x][y]$
$image[2x+1][2y].b = Hcapacity[x][y]$
$image[2x+1][2y+1].g = Hnetdensity[x][y] + Vnetdensity[x][y]$
$image[2x+1][2y+1].b = pindensity[x][y]$
where x and y represent the horizontal and vertical coordinates of the tile, respectively, and g and b are the green and blue channels of the image, respectively.

This colorization is designed so that no feature can directly interfere with the target congestion in the red channel. This allows to distinguish and extract the congestion map from the final output image, and this image can be translated back to the corresponding design data format and export back into the router.

4.2.3 Customized Deep Convolutional GAN

In the c-DCGAN structure, **Discriminator** D is only used during the model training process, and the predictive model used for congestion estimation is the **Generator** G. Using the DIT encoding presented in Section 4.2.1 and Section 4.2.2, design images can be generated regardless of file format used by the routing tool. These images contain enough information for G to perform congestion prediction by generating red-channel information from blue and green-channel data. The prediction outputs of the network are also images that can be translated back into a data format and export back to the router.

4.2.4 Generator Design

The **Generator** G learns the feature distribution of a dataset, which can be generally described using a mapping function [40]:

$$G(z) : p_z \rightarrow p_g \tag{4.1}$$

where p_z is the feature distribution of an input data set z, which in this application, are input features extracted from design data. Here, p_g is the feature distribution of a real dataset, which is the actual congestion after global routing. In other words, in c-DCGAN, G here, takes design input discussed in Section 4.2.2 and maps them

to the congestion of the same design. The generator's task is to maximize the log-likelihood [40], $\mathbb{E}_Z$, that the discriminator flags fake samples as real, for example, G works such that:

$$\max \mathbb{E}_Z[\log(D(G(z)))] \tag{4.2}$$

where $D(G(z))$ means the output of G is fed into the discriminator D. The mathematical expression of D is described in Section 4.2.5.

Inspired by [77, 117], the generator G in c-DCGAN is developed such that it comprises a stack of convolutional neural net layers, known as an encoder, and a stack of transpose layers, known as a decoder, as shown in Figure 4.5. The encoder transforms the input into a latent space. The decoder interprets and reconstructs an output from the latent space. In this specific congestion estimation application, the encoder extracts information from input image data using convolution, and these information are used by the decoder to perform the congestion estimation. The structural description of the G is shown in Table 4.1, where both encoder and decoder have 5 layers. The encoder has five convolution layers with the same activation function but different number of filters. The stride is set to 2 to downscale the image so that filters in each convolution layer can learn features from different resolution levels. A convolution with a kernel size of 3×3 for the 5th layer is too large comparing with its input of 4×4, therefore the kernel of the 5th layer is changed to 1×1. The decoder (layer 6 to 10) restores the output of the encoder back to the original image resolution using 5 transpose layers. All transpose layers have the same stride of 1 and activation functions.

In this c-DCGAN model, compared with conventional CNNs, there is no pooling layer (i.e., a layer whose purpose is to select one candidate from a data pool). The output of a pooling layer is considered as image resolution compression. However, it is designed in such a way that the image is already compressed in each layer by a stride 2 convolution, therefore a further resolution compression is unnecessary.

4.2.5 Discriminator Design

The **Discriminator** D is constructed as a CNN with multiple convolution layers and one fully connected layer. The task of training D is to maximize a value func-

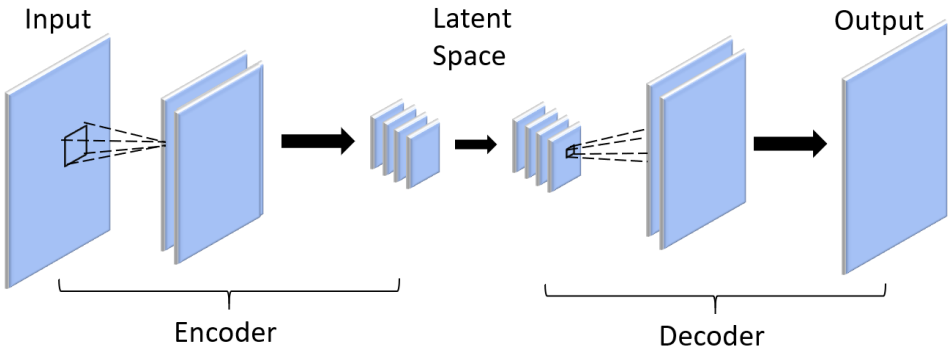


Figure 4.5: Proposed generator model G.

Table 4.1: The generator structure

	Layer #	Layer name	Stride	Activation Function	Input size	Output size
Encoder	1	Conv 3x3, 32	2	LeakyReLU	64x64, 3	32x32, 32
	2	Conv 3x3, 64	2	LeakyReLU	32x32, 32	16x16, 64
	3	Conv 3x3, 128	2	LeakyReLU	16x16, 64	8x8, 128
	4	Conv 3x3, 256	2	LeakyReLU	8x8, 128	4x4, 256
	5	Conv 1x1, 512	2	LeakyReLU	4x4, 256	2x2, 512
Decoder	6	Transpose 3x3, 256	1	relu	2x2, 512	4x4, 256
	7	Transpose 3x3, 128	1	relu	4x4, 256	8x8, 128
	8	Transpose 3x3, 64	1	relu	8x8, 128	16x16, 64
	9	Transpose 3x3, 32	1	relu	16x16, 64	32x32, 32
	10	Transpose 3x3, 3	1	relu	32x32, 32	64x64, 3

tion $V(D)$ [40], i.e.:

$$\max_{D} V(D) = \mathbb{E}_x[logD(x)] + \mathbb{E}_z[\log(1 - D(G(z)))] \tag{4.3}$$

where x is the real dataset, and z is the superposition of noise and latent input for G. The objective of D is to maximize the value of $V(D)$, which is to maximize the expectation of $\log D(x)$ and $D(G(z))$, called $\mathbb{E}_x$ and $\mathbb{E}_z$.

A congestion heatmap generated using DIT is the image format of the congestion information of a design. The purpose of D is to classify whether the congestion heatmap is the real output from the router or a fake one generated from G. The structure of D is illustrated in Figure 4.6 and the description is shown in Table 4.2.

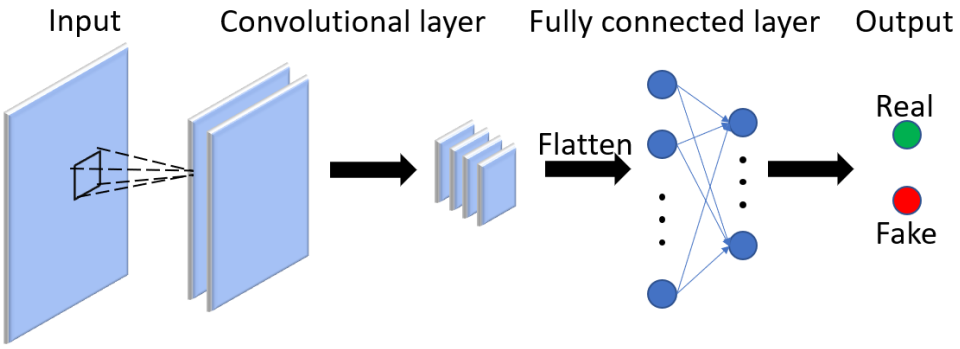


Figure 4.6: The structure of discriminator model *D* in c-DCGAN.

The first 4 layers are similar to that of the encoder in *G*, the stride of 5th layer is tuned to 1. Batch normalization [53] and dropout [102] techniques are applied to each layer for convergence purposes and to prevent model over-fitting. Lastly, data is fed to a flattened layer. The flattened layer converts the data image into a one-dimensional vector by concatenating the image data pixel-by-pixel, from the first pixel to the last one. Finally, the vector generated by the flattened layer goes through a dense layer, also called Fully Connected Layer (FCL), and generates a single value which ranges from 0 to 1, based on which a binary output will be produced. 1 represents the range of [0.5, 1], and 0 the range of [0, 0.5).

Table 4.2: The structure parameters of the discriminator model D in c-DCGAN

	Layer #	Layer name	Stride	Activation Function	Input size	Output size
Discriminator	1	Conv 3x3, 32	2	LeakyReLU	64x64, 3	32x32, 32
	2	Conv 3x3, 64	2	LeakyReLU	32x32, 32	16x16, 64
	3	Conv 3x3, 128	2	LeakyReLU	16x16, 64	8x8, 128
	4	Conv 3x3, 256	2	LeakyReLU	8x8, 128	4x4, 256
	5	Conv 1x1, 512	1	LeakyReLU	4x4, 256	4x4, 512
	6	Flattened	n/a	n/a	4x4, 512	8192
	7	Dense	n/a	sigmoid	8192	1

The objective function of the c-DCGAN can be expressed as:

$$\min_G \max_D V(D,G) = \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))] \tag{4.4}$$

the purpose of training D is to maximize the probability of correctly labeling samples, while that of G training is to minimize the log-likelihood of $1 - D(G(z))$. Eventually, a generator becomes capable of generating outputs that the discriminator cannot distinguish from actual/real data; i.e, $D(G(z)) \rightarrow 1$. Such a generator is considered to have "converged" and is saved as a trained predictive model.

4.2.6 Data Pre-Processing

The benchmarks [6] used with c-DCGAN contain up to 400×400 tiles. For training, fully routed designs are used, and each design is transformed into images with various features held in blue and green-channel and the label (congestion) in red-channel.

After the images are generated, they are segmented into a set of smaller partitions. The partition size directly affects the model size, memory requirements and, ultimately, prediction quality. Generally speaking, the larger the input size, the better the model accuracy, because larger input implies that the congestion output can be learnt from more parameters. Also, as the total number of designs is limited, a larger input size reduces sample numbers, resulting in smaller training dataset, which could cause the training to fail to converge. From empirical experiments, partition sizes of 64×64 pixels (32×32 tiles) yielded the best results. The output of G can be stitched back into the original circuit grid size by abutment.

4.3 Experimental Results

The results of using the described c-DCGAN are compared with other available academic routers, which are NTHU-Route 2.0, NTUgr2 [21], and NCTU-gr 2.0 [71]. The c-DCGAN is implemented using *Keras* [24] with *TensorFlow* [8] backend. The dataset used here has 4619 images with a resolution of 64×64 and 3 colour channels. The model is trained on a single Nvidia 1080 Ti GPU. The router algorithm is implemented in *C++* and tested on a server with Intel 2.2GHz CPUs. The model is trained for 300 epochs. There are 15 benchmarks and a total of 15 predictive models were generated, where each model was trained using 14 benchmarks as the dataset to predict the excluded one. The dataset was randomly split into two groups, 80% for training and 20% for evaluation. After the training, the

model was saved onto disk and loaded into the router to make predictions for a set of experiments.

Table 4.3: Prediction runtime

Design	Prediction (s)
a1	1.502
a2	1.596
a3	1.561
a4	1.487
a5	1.517
b1	1.506
b2	1.499
b3	1.538
b4	1.499
n1	1.479
n2	1.568
n3	1.649
n4	1.552
n5	1.715
n6	1.626

The prediction runtime of c-DCGAN is shown in Table 4.3. Although c-DCGAN is slower than MARS in Chapter 3 due to a more complex network structure, it is faster than traditional global-routing-based congestion estimation methods in Table 3.2.

The c-DCGAN is integrated into the NTHU-Route 2.0 [19], and it is compared with the original NTHU-Route 2.0. The normalized performance results are shown in Figure 4.7. In Figure 4.7 Time means the total runtime of routing all benchmarks. WL is total *wirelength* of global routing solutions of all benchmarks. TOF is the $total\ overflow$ of global routing solutions of all benchmarks. There is a small improvement in total WL and TOF, with a 19% improvement in total

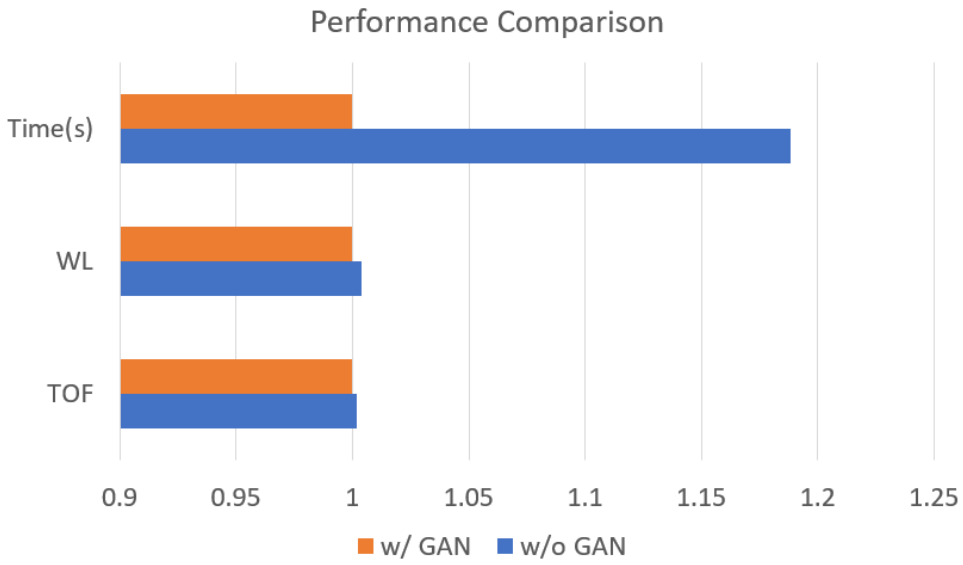


Figure 4.7: Global routing performance with and without c-DCGAN estimator using NTHU-Route 2.0 [19].

routing runtime.

Visual comparisons of congestion heatmaps of three designs are shown in Figure 4.8. The brightness indicates the congestion intensity. The brighter the pixel, the worse the congestion. It can be observed from the images on the left-hand side that regions predicted to have high congestion are also congested on the right-hand side, which is the ground-truth congestion produced by the router. Boundary artifacts exist in images produced by c-DCGAN, it was generated when smaller output images were stitched back together. This evaluation was strictly qualitative, but it provides reasonable confidence that the predictive model could generate correct congestion information based on solely the input features provided. To further improve the confidence, the c-DCGAN was also trained and tested with a human-face dataset, with results shown in Figure 4.9. The human-face experiment shows that the output (third row), has restored three-channel information (including previously removed red-channel) in a reasonable way, such as background, skin and hair. Some facial characteristics have changed such as the their appearances, and features such as glasses (left most image) and hands (second from the right) were not properly restored, it can be argued that it is because the c-DCGAN is not deep

enough to learn the complex features from human-face dataset, however, it is sufficient for learning circuit layout features. Moreover, if a pixel has a red colour only and if this red is removed, the model cannot distinguish whether this pixel was originally black or red. As a result such pixel could be restored in either black or red. In this example the model chose to restore the red shirt as black (second from the left). However, this issue would not affect the application of congestion estimation, as no pixel can have only the red colour, because this would imply that the corresponding edge is congested without any nets crossing it, and this situation is impossible in routing.

For a more quantitative comparison, the Pearson Correlation Coefficient (PCC) was used. The PCC ranges from -1 to 1. A PCC = -1 implies a negative perfect of correlation; 0.0 implies an absence of correlation; and 1 implies a positive perfect correlation. This value is computed by comparing each pixel of congestion heatmaps (estimated by c-DCGAN vs. ground-truth by NTHU-Route 2.0 [19]).

The quantitative comparison of congestion prediction is shown in Table 4.4. Two metrics are used: PCC and Normalized Root Mean Square Error (NRMSE). Note that "n3" has an unusually low PCC score because n3 is a synthetic design purposely created to challenge routers, as is shown in Figure 4.10. A further discussion of this specific design is presented in Section 7.1. The PCC of congestion heatmaps produced by the proposed c-DCGAN is greater than 0.821 (excluding design n3), indicating a strong positive correlation between the estimation congestion and the ground-truth. The NRMSE ranges from 0.104 to 0.295. This can be further improved through a method presented in Chapter 6.

4.4 Conclusion

A novel neural network, called c-DCGAN, was developed, as well as an associated data-image translator, DIT. The c-DCGAN is an advanced ML algorithm which can capture the spatial information from benchmarks. Spatial information is important in congestion estimation because the routing is performed on a grid, and resources of the grid related to congestion, such as edge *capacity* and *utilization*, are affected by how interconnects of two-pin pairs occupy spaces of the grid.

Two quantitative metrics, PCC and NRMSE were used to evaluate the quality of

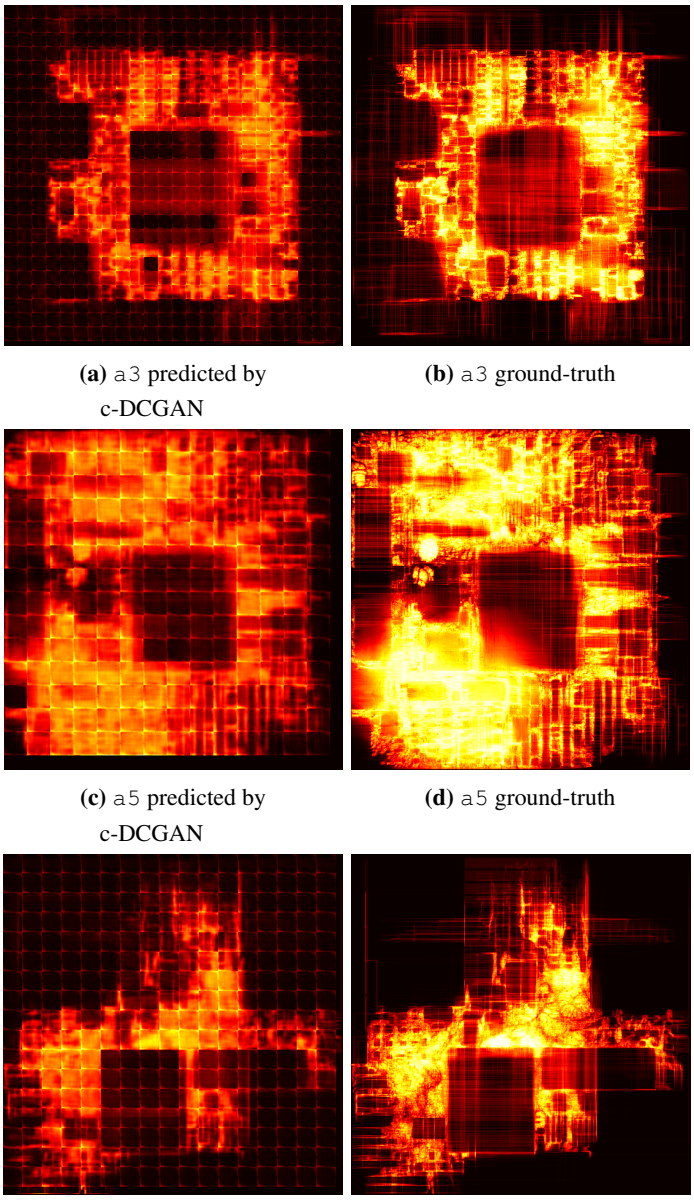
Table 4.4: Congestion estimation quality metrics

Design	PCC	NRMSE
a1	0.885	0.186
a2	0.822	0.143
a3	0.852	0.143
a4	0.767	0.150
a5	0.835	0.224
b1	0.866	0.295
b2	0.873	0.189
b3	0.828	0.118
b4	0.881	0.157
n1	0.897	0.131
n2	0.821	0.104
n3	0.269	0.074
n4	0.842	0.196
n5	0.839	0.211
n6	0.923	0.163
Avg.	0.813	0.166

the congestion estimation of c-DCGAN. The results show that the c-DCGAN provides accurate congestion estimation using the image data generated by the DIT. The PCC of congestion heatmaps produced by c-DCGAN is greater than 0.767 (excluding design `n3`). The NRMSE is less than 0.295 over the ISPD'08 [6] benchmark suite. The described DIT and c-DCGAN in this chapter can be applied to other routing tools that are different from the NTHU-Route 2.0 [19] platform. Also it appears as though the use of c-DCGAN and DIT can be extended to other back-end physical design stages, such as floorplanning and placement, for routability optimization.

Knowing the congestion prior to routing is important for routability optimiza-

tion, specially for early stages such as placement and initial global routing. The global routing algorithm itself plays an important role as well because the actual congestion (overflow) from the initial routing solution are depended on the router to eliminate. In the next chapter, a new global routing algorithm is proposed using a multi-stage strategy with different optimization objectives to solve the actual congestion problem.



(a) a3 predicted by c-DCGAN

(b) a3 ground-truth

(c) a5 predicted by c-DCGAN

(d) a5 ground-truth

(e) b3 predicted by c-DCGAN

(f) b3 ground-truth

Figure 4.8: Congestion heatmap visual comparison of three designs: congestion heatmaps predicted by c-DCGAN are on the left and ground-truth congestion heatmaps produced by NTHU-Route 2.0 [19] are on the right.

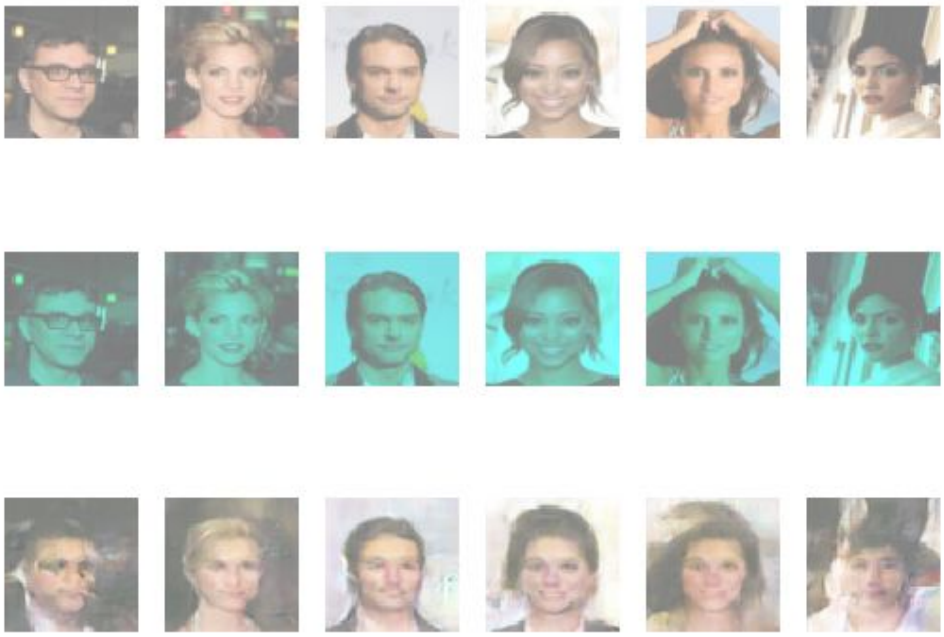
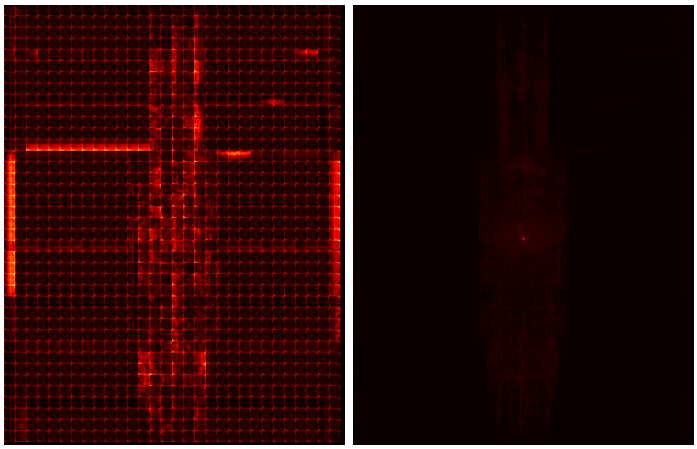


Figure 4.9: The c-DCGAN performance using human-face dataset, top row is the ground-truth, middle row is the input without red-channel, bottom row is the restored output image with network generated information in all three channels.



(a) `n3` predicted by c-DCGAN

(b) `n3` ground-truth by NTHU-Route 2.0

Figure 4.10: Congestion heatmaps visualizations: congestion heatmap predicted by c-DCGAN (left) and ground-truth congestion heatmap produced by NTHU-Route 2.0 (right).

Chapter 5

Multi-Stage Routability-Driven Global Routing Algorithm

The congestion prediction models presented in Chapter 3 and Chapter 4 do not show the impact on EDA routability-driven routing comprehensively. The predicted congestion information should be adapted to the router so that one can observe from the output whether the estimation is making an positive effort to the global routing process. In this chapter, a novel multi-stage global routing algorithm is described. It accelerates global routing runtime and improves routing quality. This algorithm consists of four optimization phases. Each phase aims at optimizing one objective, such as wirelength and overflow. In addition, experiments has been done using the modified multi-stage global router with both the c-DCGAN presented in Chapter 4 and the original congestion estimation of NTHU-Route [19] as the congestion estimator. Results of using the proposed multi-stage global routing with different congestion estimators on ISPD'08 benchmarks [6] are provided and discussed.

5.1 Introduction

Traditional congestion estimation methods [16, 20, 55, 59, 81] typically apply one or two routing stages during global routing, and multiple objectives are optimized in the same stage. This mechanism causes the problem that the iterative routing

progress may converge to a local optimum. In order to address this problem, a multi-stage global routing algorithm was developed here, which focuses on separately optimizing multiple routability objectives, such as reducing the congestion, wirelength, and total overflow. The three-dimensional global routing space is projected onto a two-dimensional plane, and based on this plane a two-dimensional global routing solution is generated by the multi-stage global routing algorithm. The two-dimensional solution then goes through the layer assignment [67] to restore a three-dimensional routing solution. Combined with the c-DCGAN congestion estimation algorithm presented in Chapter 4, the multi-stage global routing algorithm achieves the best global routing solution amongst three other available academic global routers [19, 21, 59], in terms of fastest runtime, shortest wirelength, and lowest total overflow.

5.2 Methodology

The flowchart of the global routing algorithm is shown in Figure 5.1. The 2D netlist is updated with 2D RSMTs constructions, and then read by the router. The congestion estimation is performed before the pattern routing [58] from which an initial routing solution is generated. Note that the congestion estimation method can be replaced with the c-DCGAN model described in Chapter 4.

The multi-stage global routing algorithm is such that the traditional rip-up & reroute is decomposed into three stages followed by a post-process stage, each with associated cost functions, routing orders, and routing patterns with the aim of reducing the overflow and accelerating the overall routing runtime.

1st Stage

The first stage of rip-up & reroute can be viewed as a pre-processing stage. Due to the fact that the pattern routing [58] searches only a very limited space, a monotonic routing [87] is applied to all routes that occupies edges with *overflow*. This routing process is restricted within the bounding box of the net. As a result, the *overflow* can be reduced without an increase in *wirelength*. the cost function for crossing an

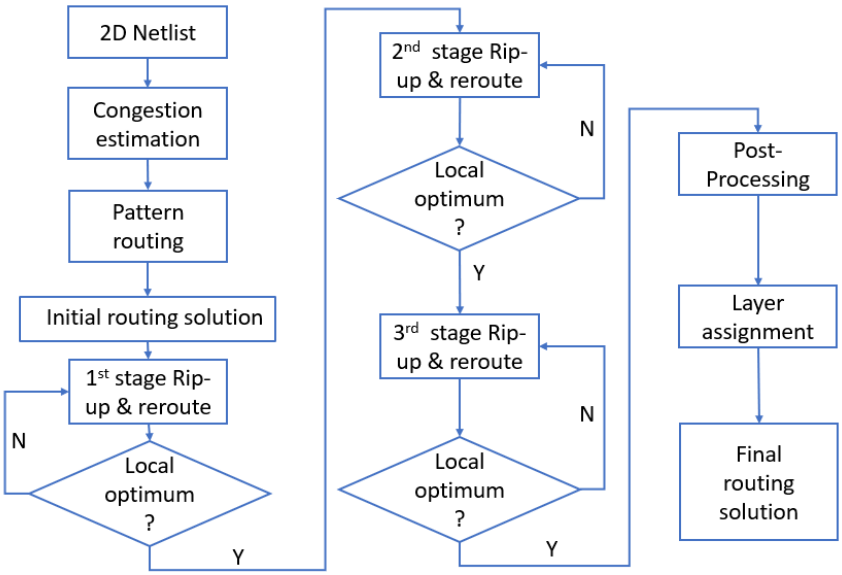


Figure 5.1: Proposed multi-objective routing flow.

edge e in the 1st stage is defined as:

$$cost_1(e) = 1 + \frac{\rho}{1 + e^{-\varepsilon * OF(e)}} + c_{via}, \tag{5.1}$$

where ρ and ε are user-defined parameters; they decide the slope of the cost with respect to congestion, and are empirically set to 0.8 and 2 in this work, respectively. Also, $OF(e)$ is the *overflow* of the edge e, and c_{via} is the cost of via, defined as:

$$c_{via} = \begin{cases} (1+\rho), & \text{route direction changes;} \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

The criterion for the algorithm to decide that a local optimum has been reached after an iteration is an overflow reduction of less than 2%. Once the criterion is met, the 2nd stage will be proceeded.

2nd Stage

At the 2nd stage, a new net sorting rule is used, one that is based on a *penalty score* that considers both the area of the net's bounding box and the net's wirelength. The higher the score, the higher the congestion of the net, because it occupies more resources and has less freedom to move. The penalty score is expressed as follows:

$$penalty\ score = (1-\omega)\frac{n_{area}}{w \times h} + \omega\frac{n_{length}}{w+h}, \tag{5.3}$$

where n_{area} and n_{length} denote the area of the net's bounding box and the net's current wirelength, respectively; w and h correspond to the respective width and length of tiles of the routing grid, and ω is a user-defined parameter to determine the weight of penalty assigned to long net's wirelength versus promoting a small bounding box area. The ω is set to 1.1 in experiments here, so that longer n_{length} contributes to the penalty score while larger n_{area} reduces the score. All nets are sorted according to the score in descending order, which means nets with longer wirelength and smaller area will be rerouted first.

After nets are sorted and the order of routing these nets is determined, the 2nd stage routing begins. In this stage, the *wirelength* constraint is relaxed to allow detours. Adaptive multi-source multi-sink maze routing (AMMMR) [37] is used. A new cost function in the 2nd stage is defined as:

$$cost_2(e) = c_{base} + c_{history} \times c_{penalty} + c_{via}, \tag{5.4}$$

where c_{via} is defined in Equation 5.2, and c_{base} is the cost for using an edge to route the net. It is expressed as follows:

$$c_{base} = 1 - e^{-\beta e^{-\gamma i}}. \tag{5.5}$$

where β and γ are user-defined parameters, and are set to 5 and 0.1 respectively, and i is the current iteration number of the rip-up & reroute process. According to Equation 5.5, $c_{base} \in [0,1]$, and it decreases as the iteration number increases. It implies that during the iterative routing in the 2nd stage, more edges will be used and therefore the *wirelength* is relaxed to allow detours for the optimization of the *overflow* objective. Also, $c_{history}$ has an initial value of 1 and for iteration $i+1$, it

is updated as:

$$c_{history}^{i+1} = \begin{cases} c_{history}^{i} + 1, & \text{current edge has overflow;} \\ c_{history}^{i}, & \text{current edge has no overflow.} \end{cases} \tag{5.6}$$

The penalty $c_{penalty}$ is defined as:

$$c_{penalty} = (\frac{c_{usage} + 1}{c_{capacity}})^{k}, \tag{5.7}$$

where k is a user-defined parameter ($k = 2$ in experiments here), and c_{usage} is the number of nets that have crossed the current edge. The cost function in the 2nd stage discourages the reuse of congested edges. As a result, the router will make most of its effort in finding detours.

3rd Stage

In the 3rd stage, congested areas are to be emphasized, called "forbidden regions" [21]. To find such a region, first the most congested edge in the design is identified, then two tiles that this edge belongs to is located, lastly the region is formed using the congested edge with the other six edges that surround the tile. This region has an initial bounding box of the size of two tiles, and this bounding box will expand iteratively from the most congested boundary, until the average congestion value of all edges of four boundaries of the bounding box go below a threshold $\lambda = 1.4$. Multiple forbidden regions can be formed at the same time if more than one edge has the same maximum congestion value and these regions can overlap with each other. The router is discouraged from using "forbidden regions" as these areas are too congested to solve, therefore detours should be made around such areas. The cost function in the 3rd stage is:

$$cost_3(e) = c_{base} + c_{via} + c_f, \tag{5.8}$$

where c_{base} is the same as Equation 5.5 and c_{via} as Equation 5.2. Here, c_f is related to the "forbidden regions" and is defined as:

$$c_f = \begin{cases} C_f \times (c_{usage}/c_{capacity}), & \text{in "forbidden regions";} \\ c_{history} \times c_{penalty} & \text{otherwise.} \end{cases} \tag{5.9}$$

where C_f is a large constant that serves as the penalty score and it discourages the use of forbidden regions, its value is set to 500 in experiments here. The $c_{history}$ is the same as Equation 5.6, and the $c_{penalty}$ is defined as:

$$c_{penalty} = \begin{cases} 1/(c_{capacity} - c_{demand}), & c_{capacity} > c_{demand}; \\ \xi + c_{demand}/c_{capacity}, & c_{capacity} < c_{demand}; \\ \xi, & c_{capacity} = c_{demand}, \end{cases} \tag{5.10}$$

where ξ is a user-defined parameter, and is set to 3 in experiments here. AMMMR [37] is still used here as the rip-up and reroute algorithm.

Post-Processing

The same cost function as the one in Equation 5.4 is used in this stage. After three stages of routing, some edges become less congested and can be released for reuse after multiple iterations of optimization. However, the $c_{history}$ term had been increasing in the previous stages and become dominant in the cost function in Equation 5.4. As a result, those edges that are not congested but having large $c_{history}$ values are blocked from being considered as usable routing resources. In order to solve this issue, the $c_{history}$ term is removed from the said cost function so that the router can re-evaluate those special edges.

5.3 Experimental Results

5.3.1 Routing Quality Comparison

The multi-stage global routing was developed based on NTHU-Route [19]. Two versions of modified NTHU-Route were developed: (1) multi-stage global routing

with NTHU-Route's original congestion estimation referred to as "half modified router", $R_{1/2}$, and (2) multi-stage global routing with c-DCGAN from Chapter 4 referred to as "fully modified router", R_F. The routing performance of all designs excluding n3 is shown in Table 5.1. TOF is the *total overflow* number, WL is the *wirelength* of the final routing solution with a scale of $\times 10^{-6}$, and T is the runtime for the complete global routing process (in seconds).

Compared with NTHU-Route 2.0, R_F successfully routed three more circuits (a2, n5, and b3), and the TOF for the remaining hard-to-route circuits is also reduced. The R_F produces the lowest TOF results. The TOF is reduced by $100\% \times (1-1/2.22) = 55\%$ compared with NCTU-gr 2.0; $100\% \times (1-1/1.12) = 11\%$ lower than NTUgr2; and $100\% \times (1-1/2.20) = 55\%$ lower than NTHU-Route 2.0. The total wirelength is reduced by $100\% \times (1-1/1.03) = 3\%$ and $100\% \times (1-1/1.01) = 1\%$ compared to NTUgr2 and NCTU-gr 2.0, respectively. More importantly, R_F is $100\% \times (1-1/10.75) = 91\%$ faster than NTUgr2 and $100\% \times (1-1/1.82) = 45\%$ faster than NTHU-Route 2.0. However, R_F is slower than NCTU-gr in terms of the overall runtime, because NCTU-gr uses an aggressive searching algorithm which ended up producing less optimal solutions with longer WL, compared with NTHU-Route 2.0 and the proposed multi-stage global routing. In terms of n3, R_F runs $100\% \times (1-1/8.08) = 88\%$ faster and produces $100\% \times (1-1/1.20) = 17\%$ less TOF, compared with NCTU-gr.

$R_{1/2}$, compared with NTHU-Route 2.0, successfully routed two more circuit (a2, n5) and has overall fewer TOF and faster runtime. The difference of WL of the three routers (NTHU-Route 2.0, $R_{1/2}$, and R_F) is less than 1%, this implies that the congestion reduction is achieved not by making detours, but by avoiding potential congested regions prior to routing or during routing. Results show that multi-stage global routing algorithm can improve the routing quality specially for hard-to-route designs, and the quality can be further improved when an ML-based congestion estimation is applied to replace the traditional global-routing-based method. Also noticeable is the runtime reduction when c-DCGAN is used. A better congestion prediction provided by c-DCGAN prevents the router from making routes at congested regions in early stages, reduces the routing complexity and makes hard-to-route designs become relatively easier.

As is mentioned in Section 4.3, the n3 benchmark is synthetically generated to

challenge routers. For this reason, n3 is excluded from results in Table 5.1, and is instead shown in Table 5.2. Academic global routers NTUgr2 [21] and NCTU-gr [59] have a runtime of several hours, and in contrast, both the R_F and $R_{1/2}$ finish in less than an hour. While all routers completed the trouting, NCTU-gr has a higher total TOF and the NTUgr2 has a higher total WL than R_F.

Table 5.1: Performance comparison of global routing with different state-of-the-art global Routers

Design	NTUgr2			NCTU-gr			NTHU-Route 2.0			Mult. Stg route w/ original ($R_{1/2}$)			Mult. Stg route w/ c-DCGAN (R_F)		
	TOF	**WL** $\times 10^6$	**T (s)**	**TOF**	**WL** $\times 10^6$	**T (s)**	**TOF**	**WL** $\times 10^6$	**T (s)**	**TOF**	**WL** $\times 10^6$	**T (s)**	**TOF**	**WL** $\times 10^6$	**T (s)**
Easy to route ciruits															
a1	0	5.60	177	0	5.44	102	0	5.36	207	0	5.33	253	0	5.38	207
a3	0	13.41	157	0	13.11	154	0	13.15	225	0	13.10	301	0	13.10	263
a4	0	12.29	59	0	12.19	63	0	12.17	56	0	12.15	74	0	12.23	77
a5	0	16.03	520	0	15.95	381	0	15.53	549	0	15.55	791	0	15.64	611
b1	0	5.85	428	0	5.97	204	0	5.57	406	0	5.55	574	0	5.60	283
n2	0	7.66	27	0	7.59	35	0	7.59	30	0	7.57	33	0	7.59	30
n6	0	18.55	487	0	18.27	238	0	17.69	968	0	17.62	724	0	17.67	439
Hard to route circuits															
a2	0	5.36	39	0	5.27	36	2	5.23	93	0	5.22	55	0	5.24	51
n5	0	23.90	1,220	0	23.46	281	18	23.14	721	0	23.07	445	0	23.21	399
b3	0	13.47	206	0	13.17	99	32	13.07	307	16	13.06	296	0	13.10	126
b2	2	9.42	6,617	4	9.10	171	84	9.00	400	62	8.96	485	8	9.01	189
n1	38	4.87	14,339	108	4.70	120	144	4.60	483	196	4.59	535	18	4.63	140
n4	148	13.55	16,327	172	13.00	158	242	12.88	1,033	224	12.87	1203	160	12.90	449
b4	212	23.96	4,479	512	23.17	277	266	22.78	2,146	268	22.73	1054	172	22.74	930
Norm.	1.12	1.03	10.75	2.22	1.01	0.55	2.20	**1.00**	1.82	2.14	**1.00**	1.63	**1.00**	**1.00**	**1.00**

Table 5.2: Performance comparison of global routing of `n3` with different State-of-the-art Global Routers

Design	NTUgr2			NCTU-gr			NTHU-Route 2.0			Mult. Stg route w/ original ($R_{1/2}$)			Mult. Stg route w/ c-DCGAN (R_F)		
	TOF	**WL** $\times 10^6$	**T (s)**	**TOF**	**WL** $\times 10^6$	**T (s)**	**TOF**	**WL** $\times 10^6$	**T (s)**	**TOF**	**WL** $\times 10^6$	**T (s)**	**TOF**	**WL** $\times 10^6$	**T (s)**
n3	31,136	17.96	36,326	37,182	10.80	21,053	31300	10.90	1.02E+06	31,082	10.67	2,790	31,050	10.70	2,604
Norm.	1.00	1.68	13.95	1.20	1.01	8.08	1.01	1.02	391.71	1.00	**1.00**	1.07	**1.00**	1.00	**1.00**

5.4 Conclusion

A multi-stage global routing algorithm which uses a "divide-and-conquer" approach was developed. It breaks the traditional two-phase global routing into four different routing stages, each associated with a unique cost function. The results show that, excluding the synthetic circuit n3, the router with c-DCGAN is able to reduce TOF up to 55% and *wirelength* up to 3%, compared with NTUgr2 [21], NCTU-gr [59], and NTHU-Route [19]. The results also show that by applying ML-based congestion estimation, c-DCGAN from Chapter 4, the final routing solutions are further improved. There is a trend shown in the results that, R_F can produce better global routing solutions if designs are hard to route. As nowadays circuit designs are becoming increasingly complex than ever, the hypothesis is that the described multi-stage global routing algorithm could bring more improvements to the back-end physical design.

Chapter 6

MEDUSA: A Multi-resolution Machine Learning Congestion Estimation Method for Global Routing

Chapter 5 showed global routing improvement by using a multi-stage routing algorithm with the c-DCGAN model versus the original congestion estimation of NTHU-Route [19]. c-DCGAN has unwanted artifact boundaries in the predicted congestion images. Also, its network structure is too complex to train and to tune compared with normal CNN. The congestion estimation can be further improved if the these issues of c-DCGAN can be resolved. In this chapter, a multi-resolution machine learning congestion estimation method for global routing, called MEDUSA, is reported. The MEDUSA contributes to three aspects: (1) simplifying the network structure complexity; (2) developing a new feature extraction algorithm, defining a new feature format as "hyper-image"; and (3) processing data after the prediction stages, respectively, to filter out the "artifact boundary" caused by image cropping and stitching.

6.1 MEDUSA Network Design

In Chapter 4, a customized Deep Convolutional Generative Adversarial Network (c-DCGAN) is presented, which uses an Auto-Encoder (AE) as a generator and a Fully Convolutional Network (FCN) as a discriminator. However, one of the drawbacks of c-DCGAN is that the network is generally difficult to train, because two sub-networks of c-DCGAN have to converge at the same time.

The network structure of MEDUSA replaces the complex c-DCGAN structure with a simplified CNN. The structure of the MEDUSA network is shown in Figure 6.1. The dimensions of the input images and the number of convolutional layers can be chosen arbitrarily. Obviously, the larger the input and the greater the number of layers, the more information is available for the model to learn. However, additional learning information comes at the expense of more memory requirements and longer training and prediction runtime. MEDUSA has four convolution layers (Conv); each convolution layer has a max-pooling layer attached.

Another modification made to meet the simplicity need is to not have the fully connected layer (FCL) in the MEDUSA network, as the total number of learning parameters introduced by the FCL is larger than that of the rest of the neural network combined. The learning parameter is a weight variable that is learnt during the training. All the parameters combined form the predictive model. More parameters do not guarantee a better learning ability to the model; in fact, it could cause issues such as over-fitting. The feature dimensions of the congestion estimation task is smaller than most Computer Vision (CV) tasks such as human face recognition. Therefore the FCL is removed and the MEDUSA model has less than half of the total learning parameters comparing with a traditional CNN with FCLs.

6.1.1 Network Training

The MEDUSA model training process is illustrated in Figure 6.2. The topologies of netlists of 15 benchmarks are generated at the beginning of the work flow. Designs are then routed using the multi-stage global routing algorithm described in Chapter 5. The actual congestion results form the output targets, which are used as labels during training. Output target image has a size of H$\times$W$\times$2, W and H correspond to the width and height of respective design grid sizes, and the 2 channels

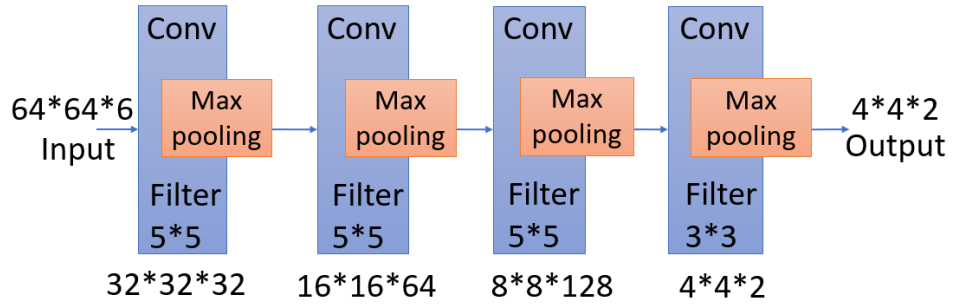


Figure 6.1: The proposed MEDUSA network structure, with functional layers and dimensions of those layers shown.

present the congestion in respective horizontal and vertical directions. On the other hand, 6 input features (discussed in Section 6.2) are extracted, and later merged into a hyper-image of size H×W×6. These "hyper-images" and output target images will be cut into smaller 64×64 and 4×4 pieces, respectively. Subsequent to the cropping, small pieces are randomly chosen in groups, called batch, and fed into the model for training batch by batch. The batch size in our experiments is set to 128. After the training process is finished, the model is considered the congestion predictor. This predictive model is used to make congestion predictions for routing with a revised routing algorithm discussed in Section 6.4.

The MEDUSA network is trained for 200,000 iterations here. Figure 6.3 illustrates how MEDUSA model converges through the plot of the MSE-loss vs iterations. The value of MSE-loss for each iteration in the plot is the average of MSE-loss summation of all training events. Experiments revealed that the model converged around the 75,000th iteration.

6.2 Feature Extraction

The method by which input features are extracted plays a critical role in machine learning algorithms. Traditional CNNs normally take visual images as input. These images, in turn, are generally represented as three layers of matrices, with matrix elements corresponding to image pixel values, and each layer corresponding to one of Red, Green or Blue (RGB) image channels. In MEDUSA, a set of circuit

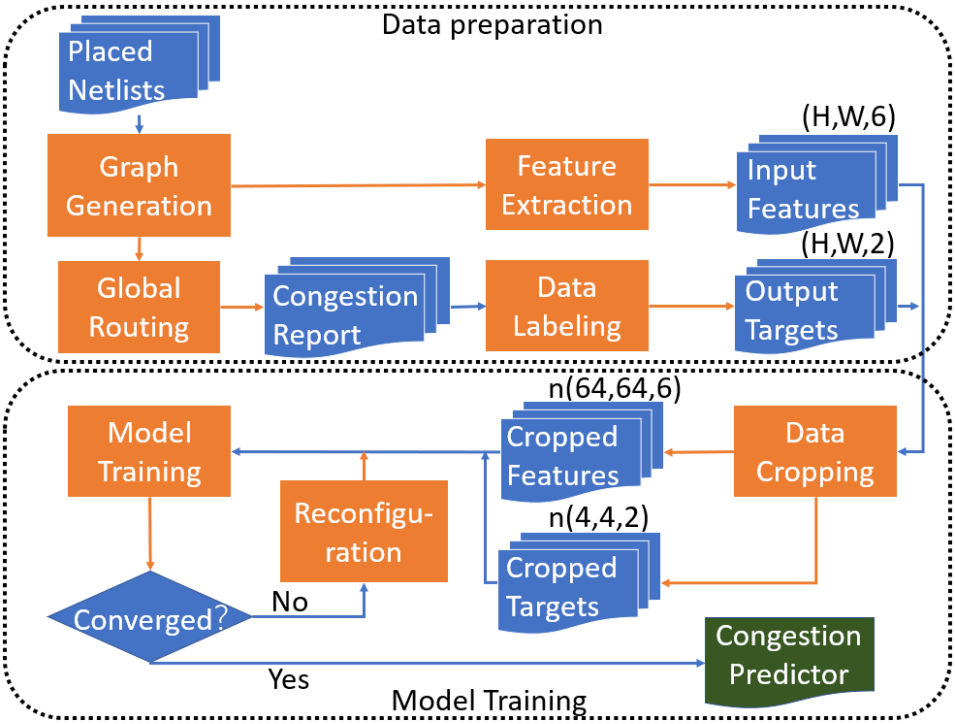


Figure 6.2: MEDUSA model training flowchart.

layout features that exceeds three is being considered. In fact, six features in total are extracted. So, in effect, a "hyper-image" constituted by six stacked layers of matrices is needed. Our proposed feature extraction is a graph to image encoding algorithm which produces hyper-images from circuit layout features.

When visiting each vertex in the routing graph, three sets of information can be obtained, including six input features:

- Vertex related: *pin density*, and *levelone pin density*
- Vertex-east-edge related: *net density*, *capacity*
- Vertex-north-edge related: *net density*, *capacity*

and two output targets:

- Vertex-east-edge *capacity*
- Vertex-north-edge *capacity*

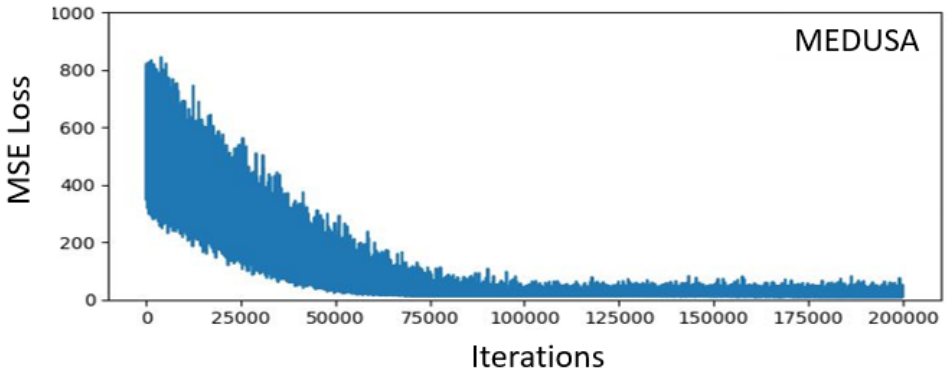


Figure 6.3: The MEDUSA model training convergence.

6.2.1 Feature Representation

The image-based data encoding algorithm used in Chapter 4 restricts all design information to be captured through three RGB channels. With such encoding, if the number of design layout features of interest or under consideration is to exceed three, then a sort of interference or aliasing will be inevitable. In particular, more than two features would need to be combined and saved in one channel, making it hard to distinguish them from one another. Indeed, in Chapter 4, several features were encoded into just 2 channels, which was potentially confusing to the 2D convolutions.

To overcome this encoding limitation, an encoding scheme that enables the creation of "hyper-images" through the use of a number of channels exceeding three was developed. This allows hyper-images to be fed into most function layers in neural networks, such as convolution layer. The features in hyper-images are described below:

Pin Density

This feature is the same as described in Section 3.2.2.

Level-One Pin Density

The *levelone pin density* of a vertex v is defined as the sum of the *pin density* all 8 adjacent vertices of v. Larger neighborhoods are not chosen as these would not have a significant impact on congestion estimation, as per [79].

Net Density (east & north)

Given a two-pin net bounding box, two L-shape edge paths are possible along the boundaries of that area. The total wirelength of these two routes divided by the half perimeter of the bounding box reflects the *net density* over a given layout region. This feature is extracted separately for east and north edges, corresponding to horizontal and vertical directions, respectively. This feature mimics how pattern routing [58] performs congestion estimation. It is the global-routing-based method that most state-of-art routers use.

Capacity (east & north)

The design netlist sets up the maximum capacity of the edges in the routing graph. However, design rules and specifications may reduce capacity in some areas of the design. The capacity feature of edges is calculated after the adjustment when all such blockages are taken into account. This feature too, has two metrics of east and north, and they represent the horizontal and vertical directions respectively as well.

A comparative study [79] shows the importance of ranking various layout features by using the correlation coefficient (CC) metric. The CC is a measurement which reflects the correlation between two variables. Here, the value of CC corresponds to the correlation of each layout feature and congestion. A simplified chart interpreted from [79] is shown in Figure 6.4, where wirelength and number of pins are the two most useful features. Neighboring features do contribute to the congestion estimation, but it is limited. Furthermore, the level-two feature provides equal or less information to the congestion estimation, but it is more expensive to compute than that of a level-one feature. Therefore, only the level-one neighboring feature is included to improve congestion estimation.

Apart from the six input features described above, the *congestion* values, de-

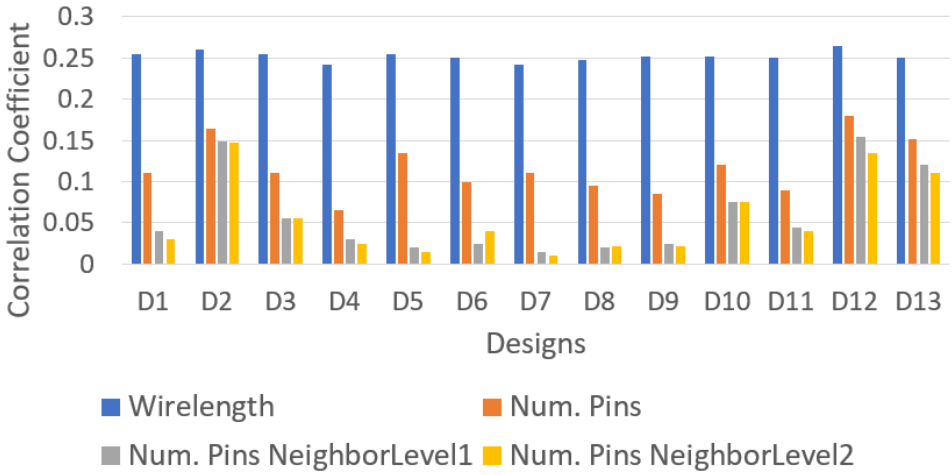


Figure 6.4: Comparison the importance of four layout features for congestion estimation using correlation coefficient[79].

fined as the exact number of routes crossing each edge in horizontal and vertical directions, are labeled and served as output targets during the model training. In the prediction phase, the model will be provided with six input features, based on which it generates output targets; in other words, it estimates congestion.

The feature extraction algorithm is listed in Figure 6.5. Feature variables are initialized at the beginning (line 1). All nets in netlist are traversed once (line 2 to line 9), to produce the *pin density* and *net density* features. The *level-one pin density* value for each vertex is assigned in the following graph traversal loop (line 10 to 17). After the first six input features are extracted, full routing is performed (line 18). Finally the actual *congestion* information is extracted by traversing the routing graph again (line 19 to line 22), and used as the target output for model training.

6.3 Congestion Prediction

The MEDUSA structure requires an input hyper-image with a size of 64×64. As a result, the output hyper-image size becomes 4×4 through the dimension manipulation after four convolution layers. Figure 6.6 illustrates how the congestion prediction inputs and outputs for each tile are made with the consideration of the

Algorithm 1 Input Feature and Output Target Extraction

Require: Benchmark netlists.

```
pin_den, level1_pin_den, net_den, capacity, congestion = Init_array();
while net ← nets.next() do
    while pin ← net.nextPin() do
        pin_den[pin.position()] += 1;
    end while
    for pos in range net.area() do
        net_den[pos] = calNetDen(net);
    end for
end while
while vertex ← graph.traverse() do
    e = vertex.edge(east/north);
    capacity[e] = e.max_cap() - macro[e];
    for i in range 8 do
        nei = vertex.neighbour(i);
        level1_pin_den[vertex.pos()] += pin_den[nei];
    end for
end while
fullRouting();
while vertex ← graph.traverse() do
    e = vertex.edge(east/north);
    congestion[e] = e.current_congestion();
end while
```

Figure 6.5: The input feature and output target extraction algorithm.

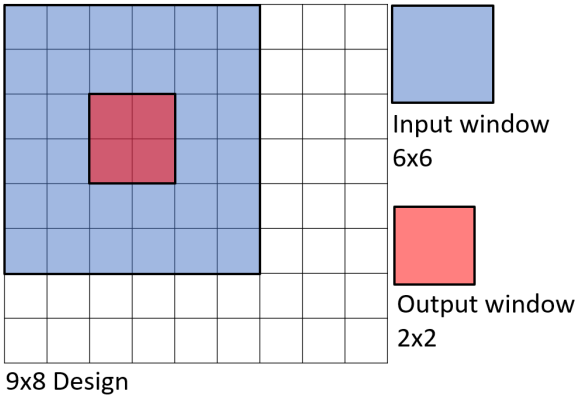


Figure 6.6: Example: The size and the location of one input to MEDUSA and its corresponding output.

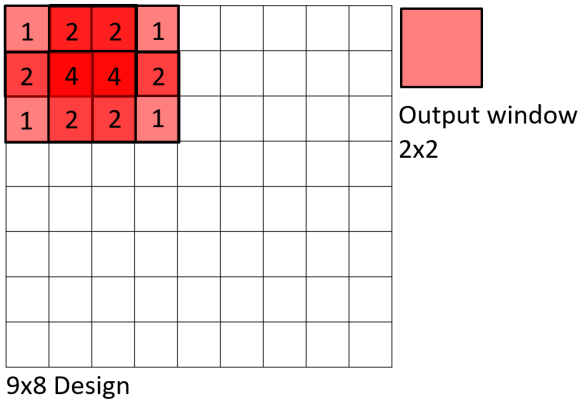


Figure 6.7: Example: Output windows, and counters of each tile for monitoring the overlapping.

region neighboring a particular tile. Here, one sample input (blue) is exemplified with a size of 6×6, on the top of the top-left corner of a design of size 9×8. Here, a 2×2 tile output (red) will be produced from the MEDUSA network corresponding to the input. Zero-padding is applied to complete the input values when predicting the congestion for the boundary tiles of a design.

6.3.1 Congestion Prediction Data Post-processing

The number of steps it takes for an input window to move to the next one is called the "stride". The smaller the stride, the more output windows there will be. A post-processing algorithm is proposed which introduces a counter for each tile, and this counter is used by the algorithm to monitor how many times one tile has been visited by output windows. An example of post-processing monitoring is illustrated in Figure 6.7. A total of six output windows are shown on the top-left corner, and the number on each tile indicates how many times a tile has been visited by an output window. Note that each tile in this example is a pixel in the hyper-image, and this pixel represents an unique edge in the routing graph. Each window passing a tile leaves a congestion value and this value is added with the existing ones. The final congestion value of one tile, t, is to take the summation of all estimated congestion values of that tile and divide it by the value of the counter of that tile, it is written as:

$$\lambda(t) = \frac{\sum \lambda_e(t)}{c} \tag{6.1}$$

where $\lambda_e(t)$ is the estimated congestion value of tile t, and c is the value of the counter which indicates how many times this tile has been visited by an output window. For the best accuracy, the stride can be set to 1 during the prediction process. As a result, the entire design will be covered with overlapping regions, as there will be multiple output windows.

6.4 Revised Routability-driven Routing flowchart

A comparative routing flowchart is shown in Figure 6.8. The main difference of the flowcharts is to make use of the congestion map produced by MEDUSA. The F_O uses pattern routing [58] as the congestion map for succeeding processes. On the other hand, F_M uses MEDUSA to provide the initial congestion map. Note that MEDUSA can predict congestion with parallel computing (GPU enabled `Keras` [24]) while the conventional approach uses a sequential algorithm.

The Flute [25] step constructs a Rectilinear Steiner Minimum Tree (RSMT), and an edge-shifting technique [116] (discussed in Section 3.3.1) is used in the router to optimize the RSMT topology based on the congestion map provided.

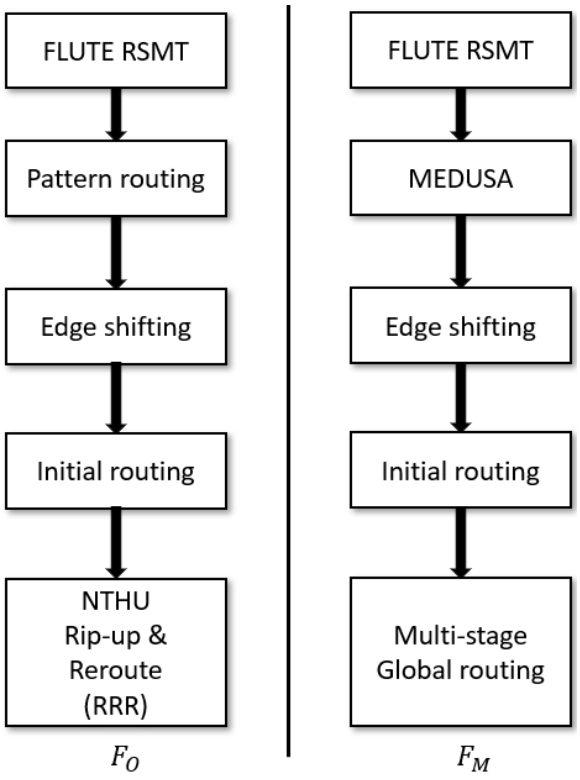


Figure 6.8: Routing flowcharts of F_O (pattern routing) and F_M (MEDUSA).

Multi-pin nets are broken into sets of two-pin pairs based on the new RSMT. The router enters the Rip-up & Reroute (RRR) stage, in which the sole objective is to reduce the congestion from initial routing with maximum effort. In MEDUSA flowchart, the multi-stage global routing algorithm proposed in Chapter 5 is used in RRR stage. The quality of a congestion map directly affects the initial routing quality which can reduce the workload of RRR. This hypothesis is confirmed by experiments in Section 6.5.2.

6.5 Experimental Results

The training of the MEDUSA model was done on a single Nvidia 1080 Ti GPU. The router algorithm was implemented in C++ and tested on a server with Intel 2.2GHz CPUs. For assessing the methodology, the training of the model was per-

Table 6.1: Congestion estimation quality comparison

Design	PCC (perfect = 1.0)				NRMSE(perfect = 0.0)			
	RUDY [101]	**c-DCGAN [125]**	**MEDUSA (stride 4)**	**MEDUSA (stride 1)**	**RUDY [101]**	**c-DCGAN [125]**	**MEDUSA (stride 4)**	**MEDUSA (stride 1)**
a1	0.899	0.885	0.933	0.965	0.219	0.186	0.124	0.094
a2	0.888	0.822	0.916	0.947	0.164	0.143	0.100	0.077
a3	0.904	0.852	0.920	0.955	0.169	0.143	0.097	0.072
a4	0.882	0.767	0.896	0.933	0.165	0.150	0.092	0.075
a5	0.899	0.835	0.927	0.967	0.231	0.224	0.138	0.083
b1	0.895	0.866	0.945	0.979	0.281	0.295	0.168	0.097
b2	0.911	0.873	0.929	0.954	0.199	0.189	0.118	0.092
b3	0.906	0.828	0.911	0.958	0.135	0.118	0.084	0.058
b4	0.922	0.881	0.932	0.967	0.186	0.157	0.101	0.072
n1	0.930	0.897	0.936	0.958	0.124	0.131	0.100	0.072
n2	0.915	0.821	0.913	0.955	0.074	0.104	0.082	0.070
n3	0.844	0.269	0.837	0.939	0.027	0.074	0.032	0.060
n4	0.909	0.842	0.936	0.964	0.220	0.196	0.119	0.082
n5	0.902	0.839	0.932	0.962	0.242	0.211	0.143	0.093
n6	0.929	0.923	0.953	0.979	0.196	0.163	0.134	0.082
Avg.	0.902	0.813	0.921	**0.959**	0.175	0.166	0.109	**0.079**

formed in such a way that the targeted design for congestion estimation would not be used to train the model. The model was trained for 140 epochs. After the training, the model was saved onto disk and loaded into the router to make predictions. The performances of different congestion estimation including c-DCGAN and MEDUSA were compared. Then, the same routing algorithm from Chapter 5 was used but with the proposed MEDUSA, to establish how the quality of the congestion map affects the global routing, with two different stride settings, i.e., a stride of 1 (s1) and a stride of 4 (s4). The final routing quality was compared with [19, 21, 71], using the metrics of Runtime, Total Overflow (TOF), and wirelength (WL).

6.5.1 Congestion Estimation Quality

The evaluation algorithm RUDY [101] is re-implemented to compare against c-DCGAN and MEDUSA. The input features are encoded into an image using algorithm discussed earlier, all pixel values normalized in the range of [0,1]. Based on these congestion estimation and ground-truth data, the PCC and the NRMSE are calculated.

Results are shown in Table 6.1. The PCC obtained in c-DCGAN [125] with a value of 0.813 is improved to 0.921 in MEDUSA when stride is set to 4, this value is further improved to 0.959 when stride is 1. NRMSE is introduced in FPGA congestion predictor [10]; their work is overall 15% better than the conventional RUDY [101] on their benchmark suite. The results here show that MEDUSA is 50.7% better than RUDY on our benchmark suite. This indirect comparison implies that MEDUSA is more accurate than the FPGA congestion predictor.

The runtime of MEDUSA's congestion estimation only using two stride settings is shown in Table 6.2. The number of inputs of the entire benchmark suite are enlarged using the windowing technique discussed in Section 6.3; the smaller the stride, the more inputs that are made available (indicated by #Inputs). The runtime for inputs increases linearly, as shown in Figure 6.9. Runtime can be reduced by an GPU upgrade without any change to the code implementation. For instance, Nvidia 3080 Ti has 10,240 cuda cores which is $4X$ more powerful than the GPU used in obtaining the experimental data of MEDUSA. Moreover, the runtime can be further reduced by stacking GPUs. On the other hand, we have not seen parallel implementations of other congestion estimation algorithms.

6.5.2 Impact of Congestion Estimation on Routing

A more accurate congestion map generally enables the router to find a better initial routing solution, which in turn, can accelerate the following RRR phase. Three metrics, *Initial TOF*, *Final TOF*, and *# Routing Iterations* are shown in Table 6.3, Table 6.4, and Table 6.5, respectively. MEDUSA has a better accuracy than c-DCGAN, and this improvement has transformed into a better initial routing solution with a 50% reduction in $initial\ TOF$. This reduction implies that the router should need fewer optimization iterations. As expected, the number of routing

Table 6.2: Runtime of MEDUSA's congestion estimation using different strides

Design	Stride 4		Stride 1	
	#Inputs	Runtime (s)	#Inputs	Runtime (s)
a1	6561	1.580	103041	6.651
a2	11236	2.192	177241	13.122
a3	37829	3.192	598296	37.141
a4	37829	3.255	598296	38.505
a5	13689	2.010	214830	13.982
b1	3248	1.617	50176	4.020
b2	13806	2.099	217620	12.442
b3	19459	2.424	305808	18.915
b4	10301	2.044	160800	10.560
n1	9999	1.989	156816	10.117
n2	16239	2.331	254840	17.091
n3	76616	5.170	1215410	70.856
n4	13109	2.088	205660	13.027
n5	25600	2.504	403858	26.775
n6	13456	2.023	212060	13.979

iterations is 5% less when using MEDUSA, in contrast to c-DCGAN [125].

These results confirmed a hypothesis that the time-consuming RRR process can be accelerated through improved congestion estimation. Moreover, as mentioned in Section 6.3.1, the result comparisons of using stride of 1 and 4 are also shown in Table 6.3, Table 6.4, and Table 6.5, respectively. When stride is set to 4 for an output window size of 4×4, there is no overlap between output windows. As a result, both initial TOF and # Routing Iterations are worse than using a stride of 1. This indicates a useful trade-off with MEDUSA. Given a fixed output window size, one can choose from no overlapping congestion estimation (fast but inaccurate) to having overlap (slower but accurate).

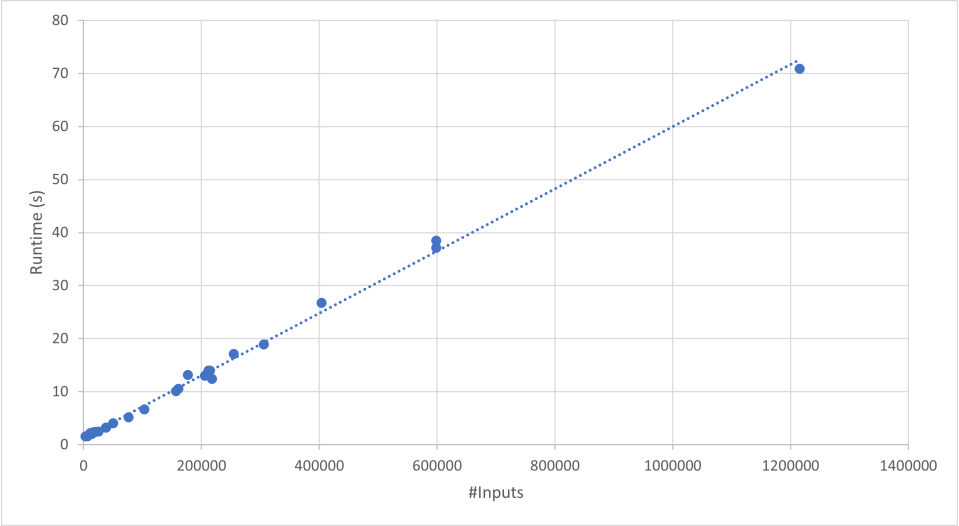


Figure 6.9: The scatter chart of relationship between Runtime and #Inputs, dotted line is the linear trend line of the two sets of data.

6.5.3 Visualization

In addition to the quantitative metrics *Initial TOF*, *Final TOF*, and Numer of Routing Iterations (*# Routing Iterations*), visualizations of congestion data for all benchmarks are presented in Appendix B. In this section, for brevity, two circuits which represent typical results were randomly selected. Figure 6.10 and Figure 6.11 show visual results for b1 and a1, respectively. Compared to the ground truth, RUDY underestimates the extent of congestion, while c-DCGAN shows artifacts at tile boundaries. In contrast, MEDUSA shows the most accurate congestion prediction and does not suffer from tile boundary noise issues. These trends are also apparent in Table 6.1, where MEDUSA has a higher PCC and lower NRMSE values compared to those of RUDY or c-DCGAN.

In addition, Figure 6.12 shows the problems encountered with `n3`, a synthetic circuit which is notoriously difficult to route. This circuit is completely unroutable (by any router) due to insufficient routing resources. There is a highly congested point in the central area of the circuit in the ground truth image which can be observed by magnifying it. c-DCGAN has difficulty with this design as there are many artifacts in the image and as a result it has a very low PCC score of 0.269.

RUDY and MEDUSA both predicted a correct pattern of congested regions, with MEDUSA predicting even more congestion than RUDY (MEDUSA is brighter than RUDY). Note that MEDUSA is able to predict the central congested point while RUDY can not.

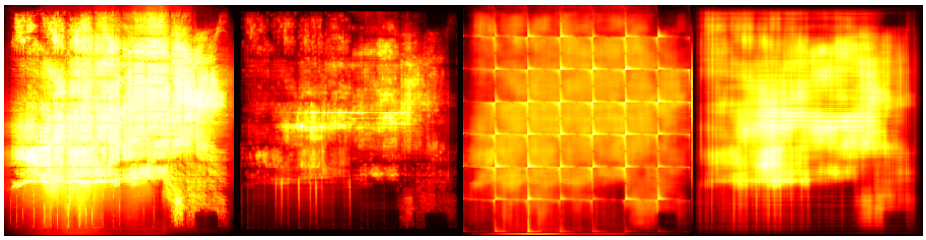
6.5.4 Global Routing Performance

The multi-stage global routing from Chapter 5 using c-DCGAN from Chapter 4 and MEDUSA were used to route the benchmark set along with three academic routers [19, 21, 71]. The routing quality (Runtime, TOF, and WL) results are shown in Table 6.6, Table 6.7, and Table 6.8, respectively. The "n3" is a special case which is purposely designed in such a way that it will give the router extra work finding routable detours in the central layout area of the design. Detailed discussion can be found in Section 7.1. Due to the special nature of n3, two sets of comparisons are presented here, one with the consideration of n3, and one without.

Comparing MEDUSA to other routers, MEDUSA has an equally good WL in comparison to NTHU-Route 2.0 [19], and other two routers are 1% and 7% worse. For final TOF, MEDUSA achieves superior performance over all the other three routers. When excluding n3, MEDUSA runs slower than NCTU [71], however, the extra runtime does yield a significant improvement in the TOF reduction. The TOF left in the global routing stage will be carried over to the next phase, detailed routing. The workload for a detailed router resolving TOF can grow exponentially. Therefore, this runtime overhead is acceptable due to the significant TOF amelioration. Lastly, if n3 is taken into account, MEDUSA becomes the fastest and also delivers best routing solutions in terms of TOF and WL, amongst all routers.

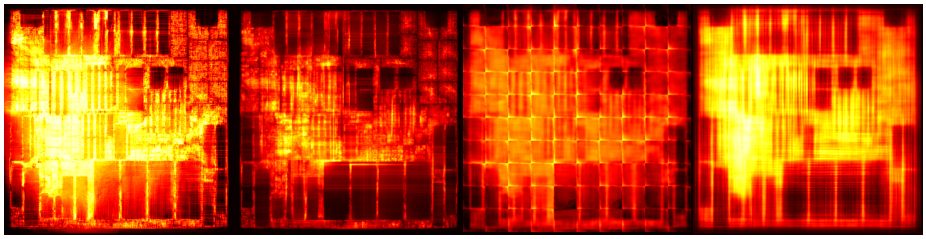
When comparing MEDUSA with c-DCGAN using the same multi-stage global routing algorithm from Chapter 5 (let R_M represents the routing algorithm using MEDUSA, and R_c using c-DCGAN), R_M has similar WL and TOF in final routing solution despite that R_M has a better PCC and NRMSE than R_c. Both R_M and R_c have a significant runtime reduction on n6 when compared with NTHU-Route 2.0, however R_c is better than R_M. The R_c also has an large runtime reduction on b1 while R_M does not, but R_M runs faster on the special case n3. Both R_c and R_M have longer runtime than NTHU-Route 2.0 on a5. This phenomenon shows that the

routing algorithm plays an important role in producing the final routing solution. The multi-stage global routing has optimized all benchmarks using c-DCGAN in terms of TOF and runtime, therefore a further routing improvement is small even though a better congestion estimation from MEDUSA is used. Designs that still have TOF may have already reached the lower bound, because comparing with R_c, the overall 2% TOF reduction from R_M comes with an overall 1% WL increase, indicating that detours have to be made to get around the congested regions.



(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure 6.10: Congestion visualizations of **b1**.

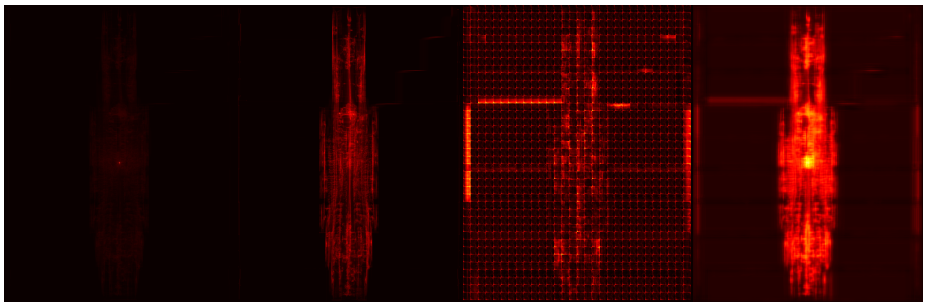


(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure 6.11: Congestion visualizations of **a1**.

6.6 Conclusion

A novel congestion estimation algorithm, MEDUSA, was developed. It consists of feature extraction and data encoding algorithm (an optimized version of DIT mentioned in Section 3.2.1), which produces "hyper-images". It solves the information interfering issue described in Chapter 4. MEDUSA also presents a simplified machine learning neural network which directly uses our encoded "hyper-images" as



(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure 6.12: Congestion visualizations of **n3**.

input to perform global routing congestion estimation. MEDUSA has the best estimation quality compared with the common RUDY [101] and the previous work c-DCGAN from Chapter 4. Moreover, routing quality has improved in both the initial and final stages.

The extended comparison shows that when using MEDUSA, the initial total overflow is reduced up to 33% when compared with other congestion estimation methods, and the number of routing iterations is reduced by 5%. This improvement in the initial stage does not impact the final routing quality significantly when MEDUSA is compared with c-DCGAN. Observations suggest that the multi-stage global routing algorithm from Chapter 5 may have already produced optimal final routing solutions when c-DCGAN is used, therefore the impact of MEDUSA on initial routing solutions cannot provide further significant final global routing optimizations. The initial improvements brought by MEDUSA may be shown at a later stage if more complex industrial designs (designs that are still not optimized by state-of-art routers) are used. Future research includes experimenting MEDUSA on new benchmarks. Routability optimization should not stop at just improving ML congestion estimation algorithms; they should be coupled with the development of global routing algorithms.

Table 6.3: Comparison of initial TOF using different techniques within the same global router

Design	Initial TOF $\times 10^4$			
	RUDY [101]	**c-DCGAN [125]**	**MEDUSA (str. 4)**	**MEDUSA (str. 1)**
a1	29.6	36.8	23.2	22.4
a2	20.8	20.4	14.4	14.1
a3	70.1	91.4	56.2	54.7
a4	23.4	32.4	17.9	16.7
a5	83.2	91.6	61.3	60.6
b1	29.6	33.4	21.7	21.4
b2	29.8	38.8	24.5	23.9
b3	19.7	19.6	14.7	14.6
b4	41.5	38.1	33.4	33.5
n1	7.8	8.9	6.1	6
n2	8.8	10.5	6.4	6.3
n3	49.0	41.6	40.3	40.1
n4	46.0	52.3	30.1	29.3
n5	82.5	99.4	63.0	61.5
n6	69.9	75.3	55.1	55
Total	611.7	690.5	468.3	460.1
Norm.	1.33	1.50	1.02	**1.00**
Excluding n3:				
Total	562.7	648.9	428.0	420.0
Norm.	1.34	1.55	1.02	**1.00**

Table 6.4: Comparison of final TOF using different techniques within the same global router

Design	Final TOF			
	RUDY [101]	**c-DCGAN [125]**	**MEDUSA (str. 4)**	**MEDUSA (str. 1)**
a1	0	0	0	0
a2	0	0	0	0
a3	0	0	0	0
a4	0	0	0	0
a5	0	0	0	0
b1	0	0	0	0
b2	8	8	8	4
b3	0	0	0	0
b4	180	172	182	176
n1	10	18	20	14
n2	0	0	0	0
n3	31094	31050	31122	31082
n4	162	160	170	158
n5	0	0	0	0
n6	0	0	0	0
Total	31454	31408	31502	31434
Norm.	1.00	**1.00**	1.00	1.00
Excluding n3:				
Total	360.0	358.0	380.0	352.0
Norm.	1.02	1.02	1.08	**1.00**

Table 6.5: Comparison of number of routing iterations using different techniques within the same global router

Design	# Routing Iterations			
	RUDY [101]	**c-DCGAN [125]**	**MEDUSA (str. 4)**	**MEDUSA (str. 1)**
a1	33	35	39	33
a2	27	27	31	28
a3	24	24	23	23
a4	17	17	18	17
a5	59	59	55	58
b1	65	64	67	65
b2	60	63	63	63
b3	28	28	28	28
b4	72	73	73	67
n1	73	76	72	71
n2	13	12	13	13
n3	90	103	92	90
n4	74	83	81	77
n5	36	36	37	39
n6	56	60	64	50
Total	727	760	756	722
Norm.	1.01	1.05	1.05	**1.00**
Excluding n3:				
Total	637.0	657.0	664.0	632.0
Norm.	1.01	1.04	1.05	**1.00**

Table 6.6: Comparison of runtime of using different global routers

Design	Runtime (s)				
	NCTU-gr [71]	**NTUgr2 [21]**	**NTHU2 [19]**	R_c	R_M **(str. 1)**
a1	98	170	236	207	207
a2	37	38	91	51	53
a3	149	146	187	263	262
a4	61	57	58	77	77
a5	352	483	557	611	704
b1	192	400	415	283	508
b2	158	6364	396	189	217
b3	95	195	302	126	124
b4	257	4672	2168	930	914
n1	110	13137	480	140	149
n2	33	26	30	30	32
n3	21100	60800	1.02E+06	2604	2490
n4	168	18451	1018	449	436
n5	265	1166	758	399	425
n6	221	486	1006	439	594
Total	2.33E+04	1.07E+05	1.03E+06	7.18E+03	7.18E+03
Norm.	3.25	14.90	143.45	**1.00**	**1.00**
Excluding n3:					
Total	2193.8	45789.6	7701.4	4194.0	4700.6
Norm.	0.47	9.74	1.64	**0.89**	1.00

Table 6.7: Comparison of final TOF using different global routers

Design	Final TOF				
	NCTU-gr [71]	**NTUgr2 [21]**	**NTHU2 [19]**	R_c	R_M **(str. 1)**
a1	0	0	0	0	0
a2	0	0	2	0	0
a3	0	0	0	0	0
a4	0	0	0	0	0
a5	0	0	0	0	0
b1	0	0	0	0	0
b2	4	2	84	8	4
b3	0	0	32	0	0
b4	512	212	266	172	176
n1	108	38	144	18	14
n2	0	0	0	0	0
n3	37100	31100	31300	31050	31082
n4	172	148	242	160	158
n5	0	0	18	0	0
n6	0	0	0	0	0
Total	37900	31500	32100	31408	31434
Norm.	1.21	1.00	1.02	**1.00**	1.00
Excluding n3:					
Total	796	400	788	358	352
Norm.	2.21	1.11	2.19	1.02	**1.00**

Table 6.8: Comparison of wirelength using different global routers

Design	Wirelength $\times 10^6$				
	NCTU-gr [71]	**NTUgr2 [21]**	**NTHU2 [19]**	R_c	R_M **(str. 1)**
a1	5.4	5.6	5.4	5.4	5.4
a2	5.3	5.4	5.2	5.2	5.2
a3	13.1	13.4	13.1	13.1	13.1
a4	12.2	12.3	12.2	12.2	12.2
a5	15.9	16.2	15.5	15.6	15.8
b1	6.0	5.9	5.6	5.6	5.8
b2	9.1	9.6	9.0	9.0	9.0
b3	13.2	13.5	13.1	13.1	13.1
b4	23.2	24.0	22.8	22.7	23.0
n1	4.7	4.9	4.6	4.6	4.6
n2	7.6	7.7	7.6	7.6	7.6
n3	10.8	17.3	10.9	10.7	10.7
n4	13.0	13.5	12.9	12.9	12.9
n5	23.5	23.9	23.1	23.2	23.1
n6	18.3	18.5	17.7	17.7	18.0
Total	181.3	191.7	178.7	178.6	179.5
Norm.	1.01	1.07	**1.00**	**1.00**	**1.00**
Excluding n3:					
Total	170.5	174.4	167.8	167.9	168.8
Norm.	1.01	1.03	**0.99**	**0.99**	1.00

Chapter 7

Conclusion

Probabilistic and global-routing based congestion estimations are imperative as they are predominantly deployed in most industrial EDA tools. ML frameworks have shown the potential as a replacement for traditional EDA approaches [49]. This work has explored the use of ML in congestion estimation. Combined with a multi-stage global routing algorithm, best-in-class results are achieved.

First, a ML algorithm using MARS for congestion estimation was developed. The congestion estimation is formulated as a non-linear regression problem, using benchmark properties as independent input variables to interpolate congestion output. A feature extraction algorithm at the post-placement stage is developed in order to generate input variable matrices for MARS. The MARS tool constructs a mathematical model using input variables such as pin position and net density to predict the congestion for ASIC designs. The model was integrated into the NTHU-Route 2.0 [19] and the congestion estimation is at least 94% faster than the traditional global-routing based approach. The global routing solution shows a 10% reduction in the $overflow$ and an average of 4.8% reduction in routing runtime. These results suggest that ML-based congestion estimation can be computed quickly, and also lead to an improvement in quality of global routing results and runtime.

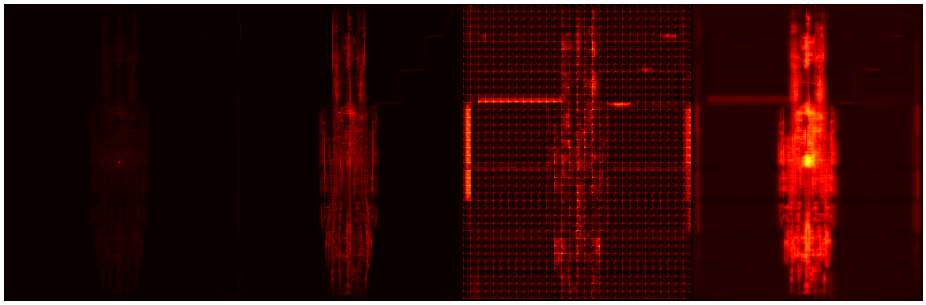
Second, a new ML network, called c-DCGAN, was developed. It is a customized GAN which consists of two NNs. One is an auto-encoder (generator **G**) and the other a decoder (discriminator **D**). An image-based feature extraction al-

gorithm, called DIT, is also developed, which has the ability of capturing two dimensional spatial information while the previous MARS does not. The DIT does not depend on the routing tools used and is able to generate images from all design file formats such as "LEF/DEF" [99]. The congestion estimation visualizations using c-DCGAN show a high similarity with the ground-truth images, in terms of shapes of congestion regions and severity of congestion. Quantitatively speaking, the quality of estimated congestion using c-DCGAN model has an average PCC of 0.813, indicating that the predicted congestion and the ground-truth are highly positively related. However, it produces boundary artifacts and can be difficult to train.

Third, a multi-stage global routing algorithm was developed, which divides the traditional routing strategy into four different optimization stages with respective cost functions. The final global routing solution shows the combined effort of both c-DCGAN congestion estimation and the multi-stage global routing algorithm. The runtime is up to 91% faster while maintaining the best quality in both TOF (up to 55% reduction) and *wirelength* (up to 3% reduction).

Fourth, a simplified CNN, called MEDUSA, was developed. The MEDUSA network structure is an encoder with only four convolution layers. A new feature extraction algorithm is developed which generates input data with more than 3 traditional channels of "red, green, and blue", called "hyper-images". These hyper-image inputs increase the numbers of features that are fed to MEDUSA. A post-processing algorithm was also developed which gives the freedom to trade-off between the prediction accuracy and runtime. The congestion estimation quality using the PCC metric proves that MEDUSA is the best (an average PCC of 0.959) among the popular approach RUDY [101], and c-DCGAN. Using MEDUSA with the multi-stage global routing algorithm, the experiments show that a better initial global routing solution can be achieved with an average 33% fewer initial TOF and 5% less routing iterations. When MEDUSA with the multi-stage global routing algorithm is compared with NTU-gr [21], NCTU-gr [71], and NTHU-Route 2.0 [19], the average final $overflow$ is reduced at most 55% while the *wirelength* is maintained at the same level and the final global routing runtime is 69% to 99% faster. However, using c-DCGAN is 11% faster than using MEDUSA.

7.1 Discussion of special benchmark n3



(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure 7.1: Revisit of congestion visualizations of **n3**.

There is a special case among all of the ISPD'08 benchmarks [6], named **n3**, as shown in Figure 7.1. The **n3** circuit is designed in such a way that the routing resource is very limited and restricted in the central area of the chip. The router takes a heavy effort to push nets to that central area and, as a result, this design has over 38000 TOF at the end of the global routing phase. It can be observed from the ground-truth image that it has a single congestion spot in the very center of the circuit and quickly fades away in all directions.

Due to the image cropping process described in Chapter 6, global information outside of the 64×64 input window is lost when estimating the congestion using the predictive models presented in this dissertation. Predictive models will only try to predict the congestion within the same area as the specific 64×64 input. During the prediction, the model uses experiences learned from other "reasonable designs" in the benchmark suite. Therefore, the congestion estimator does not know that nets in the "far away" areas will have to compete for the limited resources of the central region. This is the reason why the congestion predicted from the models is spread over the layout of n3. Similarly, when RUDY [101] makes a congestion prediction of the central spot, it is not aware that most nets across the chip will inevitably use the center of the circuit. Therefore, its output shows no bright spot as the ground-truth does. On the other hand, the window-sliding technique used in MEDUSA generates multiple inputs over the central area for the prediction and averages multiple outputs into a single prediction, allowing it to successfully pre-

dict the congested center area. This phenomenon from predicting n3 indicates the reliability and scalability of ML framework over traditional approaches. An ML framework is apparently more accurate in finding the challenging hidden relationships between circuit properties and congestion.

7.2 Cross-Comparison of All Described Congestion Predictive Models

A summary comparison of the congestion estimation quality between the most applied traditional method RUDY [101], MARS [124], c-DCGAN [125] and MEDUSA is shown in Table 7.1. Two metrics are used in the comparison: PCC and NRMSE. A PCC value indicates how similar the patterns or shapes the estimated congestion values are compared with ground-truth. On the other hand, the NRMSE shows how large the gap is between ground-truth and the estimated congestion values. An estimation can be considered accurate if it is similar to the ground-truth object in terms of both pattern and value.

To make this cross-comparison more meaningful, the same features in MEDUSA discussed in Chapter 6 are used to train the MARS model presented in Chapter 3. The only difference in this comparison between MARS and MEDUSA is the predictive model; therefore the metric scores reflect the impact brought by the models. Due to the nature of neural networks, any change in the format of input features will change the structure of the network. As a result, the c-DCGAN continues to use its features proposed in Chapter 4 in this comparison because the network structure of c-DCGAN cannot be changed once it is developed. All three models use the same ISPD'08 benchmark suite [6]. When one design is used for prediction, the rest of the 14 designs are used as training set.

MARS from Chapter 3 has a relative high PCC of 0.871, which is better than c-DCGAN but worse than MEDUSA. However, the NRMSE of MARS is 0.246, making it the worst amongst all other methods. This shows the limitation of primitive machine learning algorithms. Even though the estimated congestion from MARS moves towards the same direction as the actual congestion, the values still vary a lot because in MARS there is no consideration of spatial information. In other words, although MARS correctly predicts where the congestion happens, it

fails at forecasting how severe the congestion would be.

The c-DCGAN from Chapter 4 has the worst PCC of 0.813. This is mainly caused by the artifact boundaries over all design images (Appendix B) since the PCC score is calculated pixel-by-pixel between the estimated and the ground-truth congestion images, and those boundary artifacts pixels have no correlation with the ground-truth image. On the other hand, the NRMSE of 0.166 shows an improvement of c-DCGAN in closing the gap between the actual and the estimated congestion values when the spatial information is taken into account. The global routing solutions show improvements in terms of TOF reduction when c-DCGAN is used to replace the traditional congestion estimation in the proposed multi-stage global routing algorithm.

The MEDUSA has the best PCC and NRMSE values, at 0.959 and 0.079 respectively, because it resolves the boundary artifacts issue and considers more features. This means MEDUSA not only predicts the congestion areas correctly, but also yields the severity of those regions accurately. However, when MEDUSA and c-DCGAN are applied in the multi-stage global routing algorithm, MEDUSA does not guarantee better global routing solutions, as most metrics are similar to that of using c-DCGAN. In particular, the runtime of n3 is faster but b1 and n6 are slower, with c-DCGAN about 11% faster overall.

Moreover, even though c-DCGAN is worse than RUDY [101] in PCC and NRMSE, it does not mean this framework is inferior to RUDY [101]. Note that experiments were performed on relatively small designs compared with industrial benchmarks. The hypothesis is that when state-of-the-art and more complex designs are introduced, c-DCGAN could perform better because of its self-learning mechanism. RUDY [101] on the other hand, is a fixed systematic approach; its accuracy may degrade as its simple modeling algorithm may no longer capture the routing situations as designs and technology design rules get more complex.

7.3 Future Research Directions

This dissertation presents work on routability optimizations in three directions: (1) an improvement for existing ML-based congestion estimation; (2) a new ML framework for congestion estimation with integration with routers; and (3) a global

Table 7.1: Comparison of congestion estimation quality of all proposed algorithms

Design	PCC (perfect = 1.0)				NRMSE(perfect = 0.0)			
	RUDY [101]	**MARS [124]**	**c-DCGAN [125]**	**MEDUSA (stride 1)**	**RUDY [101]**	**MARS [124]**	**c-DCGAN [125]**	**MEDUSA (stride 1)**
a1	0.899	0.824	0.885	0.965	0.219	0.226	0.186	0.094
a2	0.888	0.889	0.822	0.947	0.164	0.170	0.143	0.077
a3	0.904	0.890	0.852	0.955	0.169	0.176	0.143	0.072
a4	0.882	0.854	0.767	0.933	0.165	0.176	0.150	0.075
a5	0.899	0.852	0.835	0.967	0.231	0.236	0.224	0.083
b1	0.895	0.792	0.866	0.979	0.281	0.309	0.295	0.097
b2	0.911	0.882	0.873	0.954	0.199	0.210	0.189	0.092
b3	0.906	0.907	0.828	0.958	0.135	0.135	0.118	0.058
b4	0.922	0.899	0.881	0.967	0.186	0.219	0.157	0.072
n1	0.930	0.932	0.897	0.958	0.124	0.134	0.131	0.072
n2	0.915	0.924	0.821	0.955	0.074	0.101	0.104	0.070
n3	0.844	0.785	0.269	0.939	0.027	0.929	0.074	0.060
n4	0.909	0.854	0.842	0.964	0.220	0.213	0.196	0.082
n5	0.902	0.859	0.839	0.962	0.242	0.257	0.211	0.093
n6	0.929	0.925	0.923	0.979	0.196	0.197	0.163	0.082
Avg.	0.902	0.871	0.813	**0.959**	0.175	0.246	0.166	**0.079**

routing algorithm. The work can be further expanded as follows:

- Experimental results show that ML-based congestion estimation performs better than traditional approaches, however the proposed two NN models, c-DCGAN and MEDUSA, show similar global routing qualities with their own good and bad, despite MEDUSA being more accurate. Further research is warranted regarding this phenomenon; understanding such behaviour could help improve MARS and c-DCGAN methods.

- As new technology nodes evolve, the realm of ML based congestion estimation is in a stage of infancy. The ML research literature develops quickly over time. Investigating not just accurate but appropriate NN models that can fit the congestion estimation task is an opportunity and these should be integrated into routers to test the improvements they can bring to the EDA

flow.

- Traditional routing algorithms perform path-finding on the two dimensional routing graph that is projected from a three dimensional space. Routers that work directly on a 3-dimensional plane are emerging [69]. An advanced feature extraction algorithm is needed in order to capture properties of latest three-dimensional chip designs such as vertical vias and routing layer assignments. If three-dimensional congestion information can be estimated, the three dimensional routing algorithm can further improve the quality of the global routing solution and hence shorten the overall routing runtime.

Routability optimization is one of the key factors throughout the entire back-end physical design. Congestion estimation can be applied to stages other than the routing, such as placement and floorplanning. The existing prediction methods will need modifications to work with different characteristics of said phases. For example, generating congestion maps in iterative placement is expensive in runtime. Therefore, fast feature extraction algorithms and prediction models are favored and worthwhile to research.

Bibliography

[1] Bookshelf overview. URL http://vlsicad.eecs.umich.edu/BK/overview.html. → pages 5, 46

[2] Ispd 2018 contest on initial detailed routing, . URL https://www.ispd.cc/contests/18/#benchmarks. → page 30

[3] Ispd 2019 contest on initial detailed routing, . URL https://www.ispd.cc/contests/19/#benchmarks. → page 30

[4] Technology node. URL https://en.wikichip.org/wiki/technology_node. → page 5

[5] Ieee 1800-2017 - ieee standard for systemverilog–unified hardware design, specification, and verification language. URL https://standards.ieee.org/standard/1800-2017.html. → page 2

[6] Ispd 2008 global routing contest, 2008. URL http://www.ispd.cc/contests/08/ispd08rc.html. → pages 34, 38, 43, 53, 57, 62, 97, 98, 119

[7] Global electronic design automation (eda) market report 2021-2026, Jul 2021. URL https://www.businesswire.com/news/home/20210705005267/en/Global-Electronic-Design-Automation-EDA-Market-Report-2021-2026. → page 1

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale

machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org. → page 53

[9] S. B. Akers. A modification of lee's path connection algorithm. *IEEE Transactions on Electronic Computers*, EC-16(1):97–98, 1967. doi:10.1109/PGEC.1967.264620. → page 1

[10] M. B. Alawieh, W. Li, Y. Lin, L. Singhal, M. A. Iyer, and D. Z. Pan. High-definition routing congestion prediction for large-scale fpgas. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 26–31, 2020. doi:10.1109/ASP-DAC47756.2020.9045178. → pages xiv, 23, 83

[11] C. Albrecht. Global routing by new approximation algorithms for multicommodity flow. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(5):622–632, 2001. doi:10.1109/43.920691. → page 3

[12] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar. *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications, USA, 1st edition, 2008. ISBN 0849372429. → page 11

[13] P. Berman and V. Ramaiyer. Improved approximations for the steiner tree problem. SODA '92, page 325–334, USA, 1992. Society for Industrial and Applied Mathematics. ISBN 089791466X. → page 10

[14] Bo Hu and M. Marek-Sadowska. Congestion minimization during placement without estimation. In *IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.*, pages 739–745, 2002. doi:10.1109/ICCAD.2002.1167614. → page 17

[15] Cadence. Encounter rtl compiler. URL http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx. → page 135

[16] Z. Cao, T. T. Jing, J. Xiong, Y. Hu, Z. Feng, L. He, and X. Hong. Fashion: A fast and accurate solution to global routing problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(4): 726–737, 2008. doi:10.1109/TCAD.2008.917590. → pages 3, 62

[17] R. C. Carden, Jianmin Li, and Chung-Kuan Cheng. A global router with a theoretical bound on the optimal solution. *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, 15(2): 208–216, 1996. doi:10.1109/43.486666. → page 3

[18] W.-T. J. Chan, P.-H. Ho, A. B. Kahng, and P. Saxena. Routability optimization for industrial designs at sub-14nm process nodes using machine learning. In *International Symposium on Physical Design*, pages 15–21, 2017. ISBN 978-1-4503-4696-2. doi:10.1145/3036669.3036681. → page 22

[19] Y. Chang, Y. Lee, and T. Wang. Nthu-route 2.0: A fast and stable global router. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 338–343, Nov 2008. → pages xv, 18, 25, 28, 39, 40, 41, 42, 43, 54, 55, 56, 57, 59, 62, 63, 67, 70, 71, 82, 86, 92, 93, 94, 95, 96

[20] H. Chen, C. Hsu, and Y. Chang. High-performance global routing with fast overflow reduction. In *2009 Asia and South Pacific Design Automation Conference*, pages 582–587, 2009. doi:10.1109/ASPDAC.2009.4796543. → pages 3, 62

[21] H. Chen, C. Hsu, and Y. Chang. High-performance global routing with fast overflow reduction. In *Asia and South Pacific Design Automation Conference*, pages 582–587, Jan 2009. doi:10.1109/ASPDAC.2009.4796543. → pages 18, 25, 53, 63, 66, 69, 70, 82, 86, 92, 93, 94, 96

[22] H.-M. Chen, M. D. F. Wong, H. Zhou, F.-Y. Young, H. H. Yang, and N. Sherwani. *Integrated Floorplanning and Interconnect Planning*, pages 1–18. Springer US, Boston, MA, 2001. ISBN 978-1-4757-3415-7. doi:10.1007/978-1-4757-3415-7_1. URL https://doi.org/10.1007/978-1-4757-3415-7_1. → page 1

[23] Chih-liang Eric Cheng. Risa: Accurate and efficient placement routability modeling. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 690–695, 1994. doi:10.1109/ICCAD.1994.629897. → page 17

[24] F. Chollet et al. Keras, 2015. URL https://github.com/fchollet/keras. → pages 53, 80

[25] C. Chu and Y. Wong. Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(1): 70–83, Jan 2008. ISSN 0278-0070. → pages 36, 37, 80

[26] J. Cong, L. He, C.-K. Koh, and P. H. Madden. Performance optimization of vlsi interconnect layout. *Integr. VLSI J.*, 21(1–2):1–94, Nov. 1996. ISSN 0167-9260. doi:10.1016/S0167-9260(96)00008-9. URL https://doi.org/10.1016/S0167-9260(96)00008-9. → page 10

[27] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3): 273–297, Sept. 1995. ISSN 0885-6125. doi:10.1023/A:1022627411411. URL https://doi.org/10.1023/A:1022627411411. → page 25

[28] W.-J. Dai. Hierarchical physical design methodology for multi-million gate chips. In *Proceedings of the 2001 International Symposium on Physical Design*, ISPD '01, page 179–181, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133472. doi:10.1145/369691.369766. URL https://doi.org/10.1145/369691.369766. → page 2

[29] J. A. Davis, V. K. De, and J. D. Meindl. A stochastic wire-length distribution for gigascale integration (gsi). i. derivation and validation. *IEEE Transactions on Electron Devices*, 45(3):580–589, 1998. doi:10.1109/16.661219. → page 17

[30] A. Deza, C. Dickson, T. Terlaky, A. Vannelli, and h. Zhang. Global routing in vlsi design: Algorithms, theory, and computational practice. *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing*, 80, 01 2012. → page 1

[31] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, Dec. 1959. ISSN 0029-599X. doi:10.1007/BF01386390. URL https://doi.org/10.1007/BF01386390. → page 1

[32] W. Donath. Placement and average interconnection lengths of computer logic. *IEEE Transactions on Circuits and Systems*, 26(4):272–277, 1979. doi:10.1109/TCS.1979.1084635. → page 17

[33] W. E. Donath and A. J. Hoffman. Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin*, 15(3):938–944, 1972. → page 4

[34] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, page 175–181. IEEE Press, 1982. ISBN 0897910206. → page 4

[35] C. J. Fisk, D. L. Caskey, and L. E. West. Accel: Automated circuit card etching layout. *Proceedings of the IEEE*, 55(11):1971–1982, 1967. doi:10.1109/PROC.1967.6027. → page 3

[36] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–67, 1991. ISSN 00905364. → pages 5, 25, 27, 34

[37] J.-R. Gao, P.-C. Wu, and T.-C. Wang. A new global router for modern designs. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, ASP-DAC '08, pages 232–237, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press. ISBN 978-1-4244-1922-7. → pages 65, 67

[38] M. Garey and D. Johnson. The rectilinear steiner tree problem in np complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977. → pages 10, 17

[39] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society for Industrial and Applied Mathematics*, 10(2):305–313, 1962. ISSN 03684245. URL http://www.jstor.org/stable/2099107. → page 3

[40] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Neural Information Processing Systems Vol. 2*, pages 2672–2680, 2014. → pages 25, 45, 49, 50, 51

[41] J. Griffith, G. Robins, J. Salowe, and T. Zhang. Closing the gap: near-optimal steiner trees in polynomial time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(11): 1351–1365, 1994. doi:10.1109/43.329264. → page 10

[42] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17(3):219–229, 1970. ISSN 00251909, 15265501. URL http://www.jstor.org/stable/2629091. → page 3

[43] M. Hanan. On steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 14(2):255–265, 1966. → page 10

[44] M. Hanan and J. M. Kurtzberg. A review of the placement and quadratic assignment problems. *SIAM Review*, 14(2):324–342, 1972. doi:10.1137/1014035. URL https://doi.org/10.1137/1014035. → page 3

[45] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136. → page 15

[46] X. He, T. Huang, L. Xiao, H. Tian, G. Cui, and E. F. Y. Young. Ripple: An effective routability-driven placer by iterative cell movement. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 74–79, Nov 2011. → page 19

[47] D. W. Hightower. A solution to line-routing problems on the continuous plane. In *Proceedings of the 6th Annual Design Automation Conference*, DAC '69, page 1–24, New York, NY, USA, 1969. Association for Computing Machinery. ISBN 9781450379298. doi:10.1145/800260.809014. URL https://doi.org/10.1145/800260.809014. → page 1

[48] M.-K. Hsu, S. Chou, T.-H. Lin, and Y.-W. Chang. Routability-driven analytical placement for mixed-size circuit designs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 80–84, Nov 2011. → page 19

[49] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong, X. Ning, Y. Ma, H. Yang, B. Yu, H. Yang, and Y. Wang. Machine learning for electronic design automation: A survey. *ACM Trans. Des. Autom. Electron. Syst.*, 26(5), June 2021. ISSN 1084-4309. doi:10.1145/3451179. URL https://doi.org/10.1145/3451179. → pages v, 25, 95

[50] W.-T. Hung, J.-Y. Huang, Y.-C. Chou, C.-H. Tsai, and M. Chao. Transforming global routing report into drc violation map with convolutional neural network. In *Proceedings of the 2020 International Symposium on Physical Design*, ISPD '20, page 57–64, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370912. doi:10.1145/3372780.3375557. → page 22

[51] F. Hwang, D. Richards, and P. Winter. *The Steiner Tree Problem*. ISSN. Elsevier Science, 1992. ISBN 9780080867939. URL https://books.google.ca/books?id=-_yKbY3X_jUC. → pages 10, 36

[52] F. K. Hwang. On steiner minimal trees with rectilinear distance. *SIAM Journal on Applied Mathematics*, 30(1):104–114, 1976. ISSN 00361399. URL http://www.jstor.org/stable/2100587. → page 10

[53] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning Vol. 37*, pages 448–456, 2015. → page 52

[54] D. Jansen. *The Electronic Design Automation Handbook*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 1441953698. → pages 1, 4

[55] Jhih-Rong Gao, Pei-Ci Wu, and Ting-Chi Wang. A new global router for modern designs. In *2008 Asia and South Pacific Design Automation Conference*, pages 232–237, 2008. doi:10.1109/ASPDAC.2008.4483948. → pages 3, 62

[56] A. B. Kahng. AI system outperforms humans in designing floorplans for microchips. *Nature*, 594(7862):183–185, June 2021. doi:10.1038/d41586-021-01515-9. URL https://doi.org/10.1038/d41586-021-01515-9. → pages v, 24

[57] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer Publishing Company, Incorporated, 1st edition, 2011. ISBN 9789048195909. → page 1

[58] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. Pattern routing: use and theory for increasing predictability and avoiding coupling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(7):777–790, July 2002. ISSN 0278-0070. doi:10.1109/TCAD.2002.1013891. → pages 63, 76, 80, 125, 131

[59] Ke-Ren Dai, Wen-Hao Liu, and Yih-Lang Li. Efficient simulated evolution based rerouting and congestion-relaxed layer assignment on 3-d global routing. In *2009 Asia and South Pacific Design Automation Conference*, pages 570–575, 2009. doi:10.1109/ASPDAC.2009.4796541. → pages 3, 25, 62, 63, 69, 70

[60] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970. doi:10.1002/j.1538-7305.1970.tb01770.x. → page 4

[61] M.-C. Kim, J. Hu, D.-J. Lee, and I. L. Markov. A simplr method for routability-driven placement. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 67–73, 2011. ISBN 978-1-4577-1398-9. → page 19

[62] R. Kirby, S. Godil, R. Roy, and B. Catanzaro. Congestionnet: Routing congestion prediction using deep graph neural networks. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 217–222, 2019. doi:10.1109/VLSI-SoC.2019.8920342. → page 23

[63] U. Kodres. *Geometrical Positioning of Circuit Elements in a Computer: A Paper to be Presented at the Fall General Meeting of the American Institute of Electrical Engineers, Chicago, Ill., 1959*. International Business Machines Corporation; Product Development Laboratory, 1959. → page 4

[64] B. S. Landman and R. L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Transactions on Computers*, C-20(12): 1469–1479, 1971. doi:10.1109/T-C.1971.223159. → page 17

[65] L. Lavagno, G. Martin, and L. Scheffer. *Electronic Design Automation for Integrated Circuits Handbook - 2 Volume Set*. CRC Press, Inc., USA, 2006. ISBN 0849330963. → pages 2, 17

[66] C. Y. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, 1961. doi:10.1109/TEC.1961.5219222. → page 1

[67] T.-H. Lee and T.-C. Wang. Robust layer assignment for via optimization in multi-layer global routing. In *Proceedings of the 2009 International Symposium on Physical Design*, ISPD '09, page 159–166, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605584492. doi:10.1145/1514932.1514968. URL https://doi.org/10.1145/1514932.1514968. → page 63

[68] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam, and J. Hu. Drc hotspot prediction at sub-10nm process nodes using customized convolutional network. In *Proceedings of the 2020 International Symposium on Physical Design*, ISPD '20, page 135–142, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370912. doi:10.1145/3372780.3375560. → page 22

[69] J. Liu, C.-W. Pui, F. Wang, and E. F. Y. Young. Cugr: Detailed-routability-driven 3d global routing with probabilistic resource model. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. doi:10.1109/DAC18072.2020.9218646. → page 101

[70] W. Liu, Y. Li, and C. Koh. A fast maze-free routing congestion estimator with hybrid unilateral monotonic routing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 713–719, Nov 2012. → page 25

[71] W. Liu, W. Kao, Y. Li, and K. Chao. Nctu-gr 2.0: Multithreaded collision-aware global routing with bounded-length maze routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(5):709–722, May 2013. ISSN 0278-0070. doi:10.1109/TCAD.2012.2235124. → pages 53, 82, 86, 92, 93, 94, 96

[72] W.-H. Liu, C.-K. Koh, and Y.-L. Li. Case study for placement solutions in ispd11 and dac12 routability-driven placement contests. In *Proceedings of the 2013 ACM International Symposium on International Symposium on Physical Design*, ISPD '13, pages 114–119, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1954-6. doi:10.1145/2451916.2451944. URL http://doi.acm.org/10.1145/2451916.2451944. → page 135

[73] Y. Liu, H. Yang, W. Yu, X. Cui, and J. Huang. An fpga timing routing algorithm based on pathfinder and rip-up and retry approach. 26:138–145, 01 2014. → page 3

[74] J. Lou, S. Thakur, S. Krishnamoorthy, and H. Sheng. Estimating routing congestion using probabilistic analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 21(1): 32–41, Jan 2002. ISSN 0278-0070. → pages xvii, 18, 126, 127

[75] C. L. Lu, C. Y. Tang, and R. C.-T. Lee. The full steiner tree problem. *Theor. Comput. Sci.*, 306(1):55–67, Sept. 2003. ISSN 0304-3975. → page 10

[76] D. Marcotte. Generalized cross-validation for covariance model selection. *Mathematical Geology*, 27:659–672, 07 1995. doi:10.1007/BF02093906. → page 34

[77] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Artificial Neural Networks - Volume Part I*, pages 52–59, 2011. ISBN 978-3-642-21734-0. → page 50

[78] C. Mead and L. Conway. *Introduction to VLSI Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1979. ISBN 0201043580. → page 1

[79] J. Melchert, B. Zhang, and A. Davoodi. A comparative study of local net modeling using machine learning. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, GLSVLSI '18, page 273–278, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357241. doi:10.1145/3194554.3194579. → pages xvi, 76, 77

[80] K. MIKAMI. A computer program for optimal routing of printed circuit connectors. *IFIPS Proc., 1968*, 1968. URL https://ci.nii.ac.jp/naid/10022177201/en/. → page 1

[81] M. D. Moffitt. Maizerouter: Engineering an effective global router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(11):2017–2026, 2008. doi:10.1109/TCAD.2008.2006082. → pages 3, 62

[82] E. Moore. *The Shortest Path Through a Maze*. Bell Telephone System. Technical publications. monograph. Bell Telephone System., 1959. URL https://books.google.ca/books?id=IVZBHAAACAAJ. → page 1

[83] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957. ISSN 03684245. URL http://www.jstor.org/stable/2098689. → page 1

[84] D. R. Musser and A. Saini. *The STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. Addison Wesley Longman Publishing Co., Inc., USA, 1995. ISBN 0201633981. → page 28

[85] R. Netto, S. Fabre, T. A. Fontana, V. Livramento, L. L. Pilla, L. Behjat, and J. L. Güntzel. Algorithm selection framework for legalization using deep convolutional neural networks and transfer learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2021. doi:10.1109/TCAD.2021.3079126. → page 21

[86] R. Otten. Automatic floorplan design. pages 261– 267, 07 1982. ISBN 0-89791-020-6. doi:10.1109/DAC.1982.1585510. → page 3

[87] M. Pan and C. Chu. Fastroute: A step to integrate global routing into placement. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 464–471, Nov 2006. → pages 19, 36, 37, 63

[88] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957. doi:10.1002/j.1538-7305.1957.tb01515.x. → page 9

[89] Z. Qi, Y. Cai, and Q. Zhou. Accurate prediction of detailed routing congestion using supervised data learning. In *IEEE International Conference on Computer Design (ICCD)*, pages 97–103, Oct 2014. → page 22

[90] D. Richards. Complexity of single-layer routing. *IEEE Trans. Comput.*, 33 (3):286–288, Mar. 1984. ISSN 0018-9340. doi:10.1109/TC.1984.1676428. URL https://doi.org/10.1109/TC.1984.1676428. → page 4

[91] S. M. Rubin. Appendix c: Gds ii format, 1994. URL https://www.rulabinsky.com/cavd/text/chapc.html. → page 4

[92] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010. → page 27

[93] M. Saeedi, M. S. Zamani, and A. Jahanian. Prediction and reduction of routing congestion. In *Proceedings of the 2006 International Symposium on Physical Design*, ISPD '06, pages 72–77, New York, NY, USA, 2006. ACM. ISBN 1-59593-299-2. doi:10.1145/1123008.1123023. URL http://doi.acm.org/10.1145/1123008.1123023. → page 19

[94] P. Saxena, R. S. Shelar, and S. S. Sapatnekar. Routing congestion in vlsi circuits: Estimation and optimization. 2007. → pages 15, 24, 25, 135, 137

[95] N. Selvakkumaran, P. N. Parakh, and G. Karypis. Perimeter-degree: A priori metric for directly measuring and homogenizing interconnection complexity in multilevel placement. In *Proceedings of the 2003 International Workshop on System-Level Interconnect Prediction*, SLIP '03, page 53–59, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136277. doi:10.1145/639929.639941. URL https://doi.org/10.1145/639929.639941. → page 17

[96] C.-W. Sham and E. Young. Congestion prediction in floorplanning. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, volume 2, pages 1107–1110, Jan 2005. → page 18

[97] C.-W. Sham, E. F. Y. Young, and J. Lu. Congestion prediction in early stages of physical design. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(1):12:1–12:18, Jan. 2009. ISSN 1084-4309. → page 18

[98] N. A. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, USA, 1993. ISBN 0792392949. → page 9

[99] signoff scribe. Lef, def & lib, Mar 2018. URL http://www.signoffsemi.com/lef-def-lib/. → pages 5, 46, 96

[100] P. Spindler and F. Johannes. Fast and accurate routing demand estimation for efficient routability-driven placement. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1–6, April 2007. doi:10.1109/DATE.2007.364463. → page 137

[101] P. Spindler and F. M. Johannes. Fast and accurate routing demand estimation for efficient routability-driven placement. In *Design, Automation Test in Europe*, pages 1–6, April 2007. doi:10.1109/DATE.2007.364463. → pages 25, 26, 47, 82, 83, 87, 88, 89, 90, 91, 96, 97, 98, 99, 100, 119, 120, 121, 122

[102] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. ISSN 1532-4435. → page 52

[103] L. Steinberg. The backboard wiring problem: A placement algorithm. *SIAM Review*, 3(1):37–50, 1961. ISSN 00361445. URL http://www.jstor.org/stable/2027247. → page 3

[104] S. M. Stigler. *The History of Statistics: The Measurement of Uncertainty Before 1900*. Harvard University Press: Cambridge, 1986. → page 9

[105] D. Stroobandt. A priori wire length estimates for digital design. 01 2001. doi:10.1007/978-1-4419-8499-9. → page 17

[106] T. G. Szymanski. Dogleg channel routing is np-complete. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(1):31–41, 1985. doi:10.1109/TCAD.1985.1270096. → page 4

[107] A. F. Tabrizi, N. K. Darav, S. Xu, L. Rakai, I. Bustany, A. Kennings, and L. Behjat. A machine learning framework to identify detailed routing short violations from a placed netlist. In *ACM/IEEE Design Automation Conference (DAC)*, pages 48:1–48:6, 2018. ISBN 978-1-4503-5700-5. → page 22

[108] T. Taghavi, F. Dabiri, A. Nahapetian, and M. Sarrafzadeh. Tutorial on congestion prediction. In *Proceedings of the 2007 International Workshop on System Level Interconnect Prediction*, SLIP '07, page 15–24, New York, NY, USA, 2007. Association for Computing Machinery. ISBN

9781595936226. doi:10.1145/1231956.1231961. URL https://doi.org/10.1145/1231956.1231961. → page 9

[109] N. Viswanathan, C. J. Alpert, C. Sze, Z. Li, G.-J. Nam, and J. A. Roy. The ispd-2011 routability-driven placement contest and benchmark suite. In *Proceedings of the 2011 International Symposium on Physical Design*, ISPD '11, pages 141–146, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0550-1. doi:10.1145/1960397.1960429. URL http://doi.acm.org/10.1145/1960397.1960429. → page 135

[110] J. Westra and P. Groeneveld. Is probabilistic congestion estimation worthwhile? In *Proceedings of the 2005 International Workshop on System Level Interconnect Prediction*, SLIP '05, page 99–106, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930337. → pages 23, 24, 27

[111] J. Westra, C. Bartels, and P. Groeneveld. Probabilistic congestion prediction. In *ACM International Symposium on Physical Design (ISPD)*, pages 204–209, 2004. ISBN 1-58113-817-2. → pages 18, 31, 125, 126, 131, 137

[112] J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964. → page 9

[113] T. Wu, A. Davoodi, and J. T. Linderoth. Grip: Scalable 3d global routing using integer programming. In *2009 46th ACM/IEEE Design Automation Conference*, pages 320–325, 2009. → page 3

[114] T. Wu, A. Davoodi, and J. T. Linderoth. A parallel integer programming approach to global routing. In *Design Automation Conference*, pages 194–199, 2010. → page 3

[115] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and Nvidia. Routenet: Routability prediction for mixed-size designs using convolutional neural network. In *International Conference on Computer-Aided Design*, pages 80:1–80:8, 2018. ISBN 978-1-4503-5950-4. doi:10.1145/3240765.3240843. → page 22

[116] Y. Xu, Y. Zhang, and C. Chu. Fastroute 4.0: Global router with efficient via minimization. In *Asia and South Pacific Design Automation Conference*, pages 576–581, 2009. ISBN 978-1-4244-2748-2. → page 80

[117] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Y. Young. Gan-opc: Mask optimization with lithography-guided generative adversarial nets. In *Design Automation Conference*, pages 131:1–131:6, 2018. ISBN 978-1-4503-5700-5. doi:10.1145/3195970.3196056. → page 50

[118] X. Yang, E. Bozorgzadeh, and M. Sarrafzadeh. Wirelength estimation based on rent exponents of partitioning and placement. In *ACM Workshop on System Level Interconnect Prediction (SLIP)*, pages 25–31, 2001. ISBN 1-58113-315-4. → page 17

[119] X. Yang, R. Kastner, and M. Sarrafzadeh. Congestion estimation during top-down placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 21(1):72–80, Jan 2002. ISSN 0278-0070. → page 17

[120] B. Yu, D. Z. Pan, T. Matsunawa, and X. Zeng. Machine learning and pattern matching in physical design. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 286–293, Jan 2015. → page 21

[121] C. Yu and Z. Zhang. Painting on placement: Forecasting routing congestion using conditional generative adversarial nets. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367257. doi:10.1145/3316781.3317876. URL https://doi.org/10.1145/3316781.3317876. → pages xiv, 23

[122] H. Zhou. Efficient steiner tree construction based on spanning graphs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(5):704–710, 2004. doi:10.1109/TCAD.2004.826557. → page 10

[123] Z. Zhou, P. Hallschmid, and A. Ivanov. Local congestion and blockage aware routability analysis using adaptive flexible modeling. In *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 438–439, 2016. doi:10.1109/ICECS.2016.7841230. → pages 31, 123

[124] Z. Zhou, S. Chahal, T. Ho, and A. Ivanov. Supervised-learning congestion predictor for routability-driven global routing. In *International Symposium on VLSI Design, Automation and Test*, pages 1–4. IEEE, 2019. → pages 40, 41, 42, 98, 100, 123

[125] Z. Zhou, Z. Zhu, J. Chen, Y. Ma, B. Yu, T. Ho, G. Lemieux, and A. Ivanov. Congestion-aware global routing using deep convolutional generative adversarial networks. In *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, 2019. → pages 5, 82, 83, 84, 89, 90, 91, 98, 100, 123

Appendix A

The Training Procedures of Presented Machine Learning Models

Three ML-based congestion estimation models are presented in this dissertation, MARS in Chapter 3, c-DCGAN in Chapter 4, and MEDUSA in Chapter 6. The training methods selected for these models are explained as follows.

A.1 MARS

A variation of Leave-One-Out validation was selected as the model evaluation method. Instead of choosing one subset as the test set and the rest of subsets as training set, MARS does the opposite. It uses one subset as the training set and the rest as the test set. The reason of having this reversed cross validation is to avoid a problem observed when too much training data was used. The MARS model is a primitive regression model in terms of number of hyper-parameters. A degradation in prediction ability was observed when more than 5 subsets were used during training. Therefore, the number of training set subsets was reduced. A total of 10 models were trained using different training sets and the one with the best prediction accuracy was selected and used.

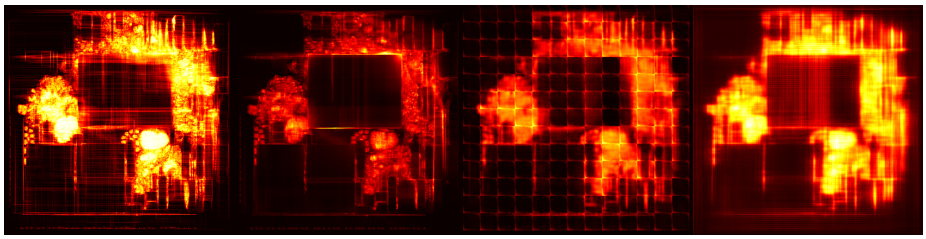
A.2 c-DCGAN and MEDUSA

The standard Leave-One-Out cross validation method was used as the model evaluation method for both c-DCGAN and MEDUSA. Models were trained on all the data except for one benchmark, and a prediction was made for that benchmark. Unlike MARS, no issue was observed as training set size increased. A total of 15 models were trained and used to predict congestion in the respective unseen benchmark.

Appendix B

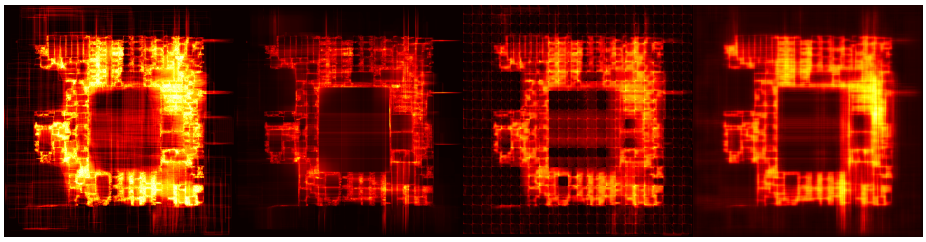
Congestion Estimation Visualizations of ISPD'08 benchmark suite

Congestion estimation visualizations for the remaining benchmarks of ISPD'08 [6] that are not shown in Chapter 4 and Chapter 6.



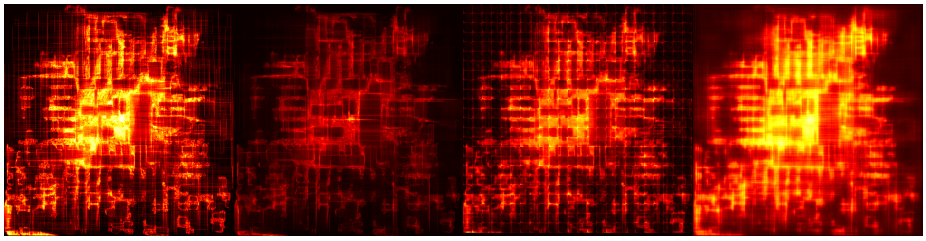
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.1: Congestion visualizations of **adaptec2**.



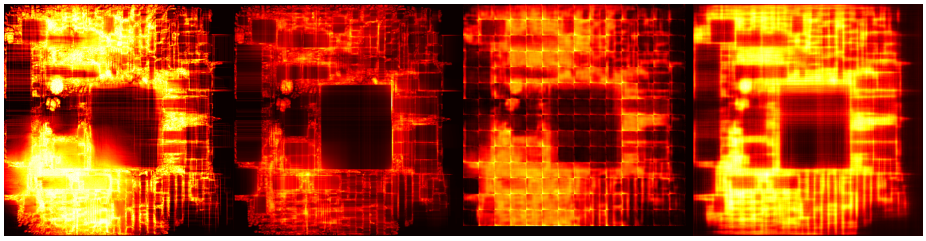
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.2: Congestion visualizations of **adaptec3**.



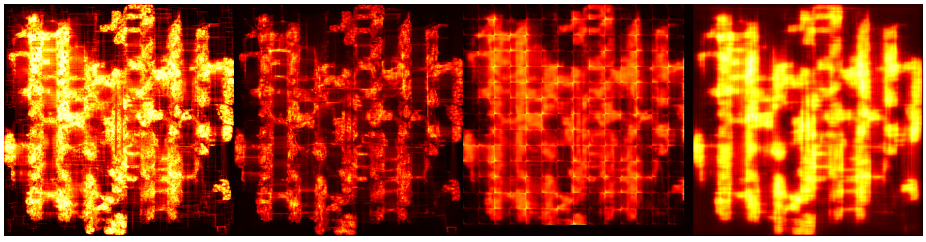
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.3: Congestion visualizations of **adaptec4**.



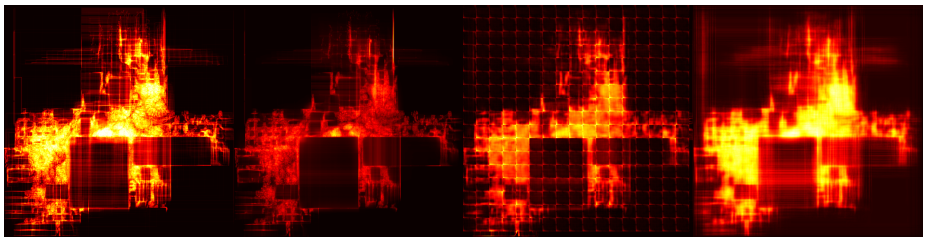
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.4: Congestion visualizations of **adaptec5**.



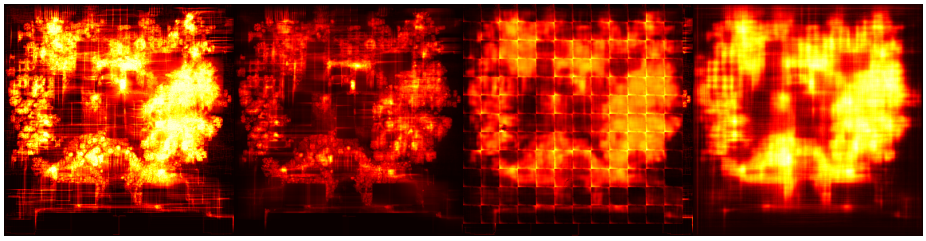
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.5: Congestion visualizations of **bigblue2**.



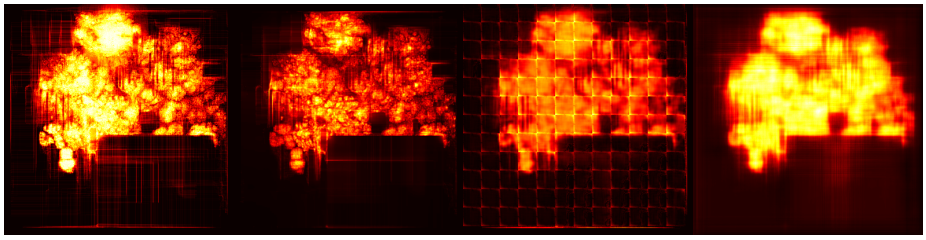
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.6: Congestion visualizations of **bigblue3**.



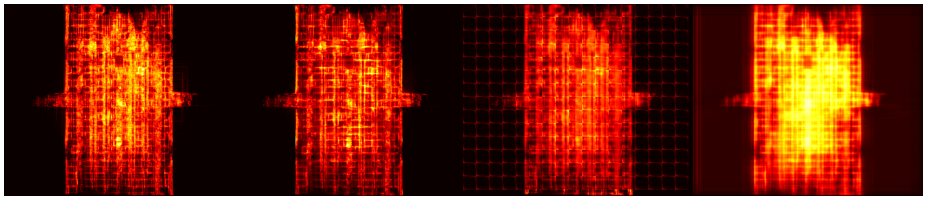
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.7: Congestion visualizations of **bigblue4**.



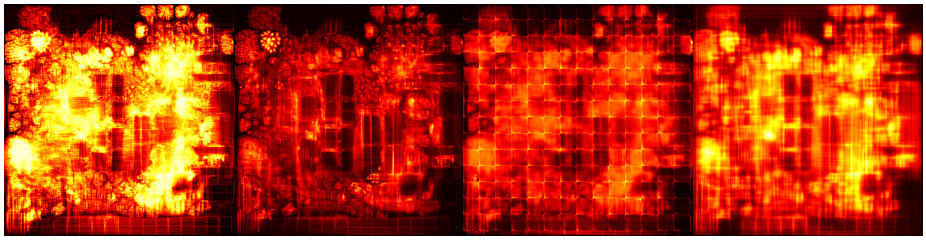
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.8: Congestion visualizations of **newblue1**.



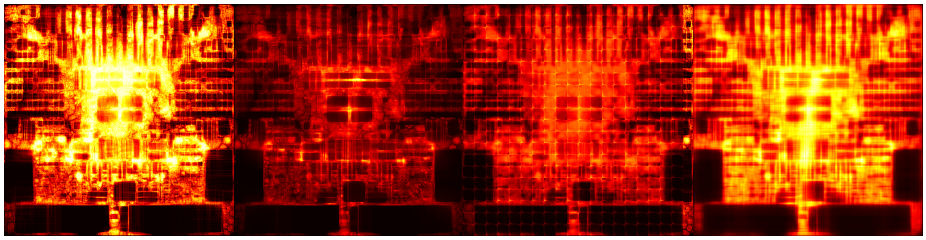
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.9: Congestion visualizations of **newblue2**.



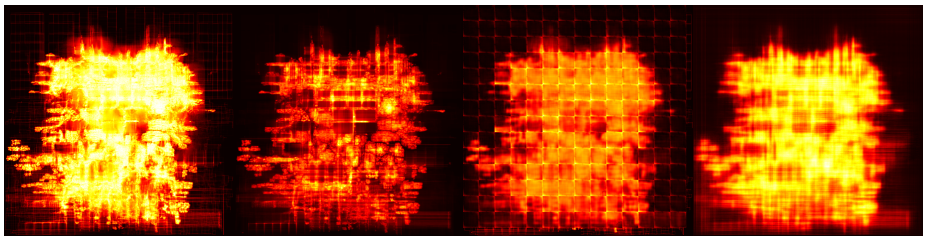
(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.10: Congestion visualizations of **newblue4**.



(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.11: Congestion visualizations of **newblue5**.



(a) Ground truth **(b)** RUDY[101] **(c)** c-DCGAN **(d)** MEDUSA

Figure B.12: Congestion visualizations of **newblue6**.

Appendix C

Local Congestion and Blockage Aware Routability Analysis using Adaptive Flexible Modeling (AFM)

The AFM is a probabilistic based congestion estimation algorithm appears in [123]. Although this algorithm uses no ML techniques, the routing shape modeling algorithm and congestion probabilistic distribution calculation introduced in this work led to a feature extraction in later works [124, 125] that involves ML techniques. The methodology and experimental results of the AFM are provided here.

C.1 Adaptive Flexible Modeling for Congestion Estimation

A novel adaptive local-congestion-aware algorithm, called AFM, is developed. The routing distributions are adapted dynamically as the algorithm runs and congestion information is generated as possible routes are considered and assessed. Thus, the congestion incurred by each route is based more realistically on the predicted congestion caused by all other routes and blockages. As a consequence, the proposed algorithm more realistically estimates congestion in areas where blockages or high

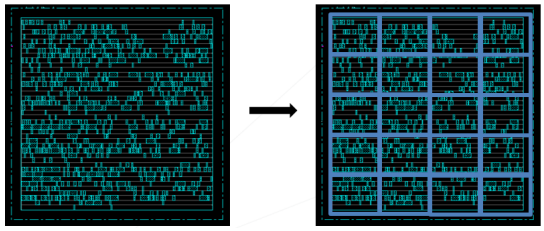


Figure C.1: A circuit design is tessellated in tiles, and tiles form a grid.

pin density occurs.

To estimate congestion, AFM makes use of a spatial grid that spans the layout and is used to store the cumulative congestion contributions of all nets. Note that the grid used in AFM is not the one generated by the global router (discussed in Section 2.1). The grid in AFM can be set to be the same as a global router uses in terms of size and resolution (number of tiles).

During the progression of the AFM, the pins of each net are partitioned into a set of two-pin pairs using a Minimum Spanning Tree. Each pin pair is used to construct the congestion value stored in grid tiles of the design. Since a net can be routed in many ways between its source and sink pins, a set of routing shapes is defined each with their own probability of occurrence. In traditional works, these probabilities were either fixed or uniformly distributed. In AFM, probabilities are dynamically changing based on current congestion stored in the grid. Also, the blockage in the design that makes routing or placement infeasible can be avoided automatically when applying AFM, because the probability of using blockages is 0 under the AFM modeling. In the following subsections, detailed descriptions of the design grid, routing shapes, and the algorithm used in the proposed congestion analysis approach are discussed.

C.1.1 Design Grid and Congestion Metric

In order to perform the fast estimation described, and to avoid slow global routing, a grid is applied to divide the placement into grid tiles, as is shown in Fig. C.1. The density of the grid tile sets the trade-off between runtime (the lower the grid density the faster the algorithm runs) and accuracy (higher density allows a better estimation result). Fig. C.2 shows an blockage within a single tile with a coordi-

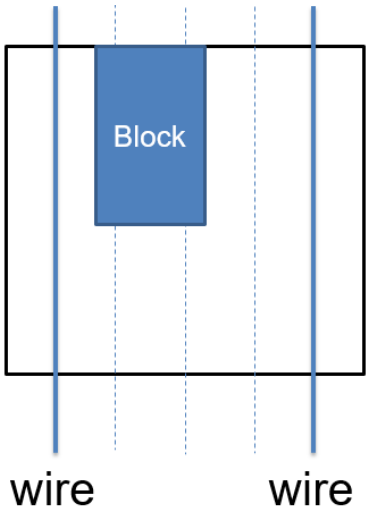


Figure C.2: An example of a tile in the grid with an blockage.

nation of (i, j) in the grid. The blockage has a length of half of the tile unit in vertical direction. Originally, there would be five vertical paths available for routing, indicating a initial *capacity* of 5 for both top and bottom edges of the tile, $C_{i,j}(e_{top}) = C_{i,j}(e_{bot}) = 5$. The blockage makes two of them partially inaccessible in the x direction. The partial inaccessibility, two paths with unit 1 have both been block 0.5 due to the blockage length, resulted in a total vertical reduction of 1. Therefore, the final vertical *capacity* $C_{i,j}$ of this grid tile has decreased to 4 from 5. The *utilization* of this tile $U_{i,j}$ is the number of wires that passes through it. In this case, the total *utilization* is 2. In general, the *utilization* of one direction is the net's length ($\Delta|x|$ or $\Delta|y|$) in that direction divided by tile *width* (w) or *height* (h), and the summation of the *utilization* from both directions yields the final *utilization* value of that single grid tile.

C.1.2 Routing Models

AFM adopts the modeling methodology from the work of Westra *et al.* [111], where the router tends to minimize not only the *wirelength*, but also the number of vias. The logic of using routing shapes with a maximum of two bends allows us to move closer to the actual "pattern routing" behavior of global routers [58]. Such

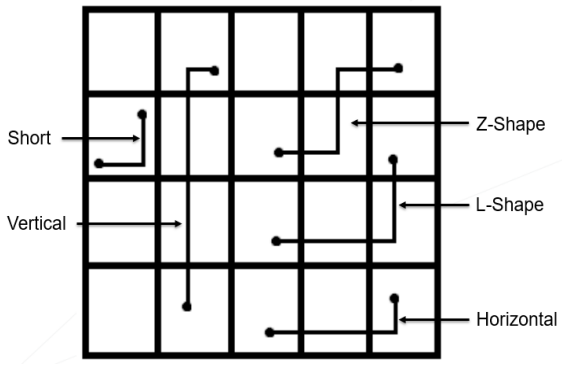


Figure C.3: Five routing shapes used for calculating the utilization.

technique is commonly used nowadays in both commercial and academic tools.

A bend, by definition, means that the direction of a routing wire changed to the perpendicular direction from its original one. In Fig. C.3, 5 different pin-pair routing shapes are shown with each routing shape having no more than two bends, while no over-the-box detour is considered. Every net contributes a probabilistic *utilization* to each grid tile occupied by its routing track. Also, by applying these routing shapes, routing *utilization* is generally gathered at the boundaries of the pin-pair bounding box instead of the central regions described in the work of Lou *et al.* [74]. In comparison, as shown in Fig. C.4, Westra *et al.* [111] provided experimental proof that routing *utilization*s gathered at the boundaries is a more practical solution. The first few iterations of a "pattern routing" global router would as well try to connect the pin-pair through edges of the routing region. Also, the comparison results from other works show that the modeling approach of Westra *et al.* is significantly more useful than the purely stochastic approach of Lou *et al.*

In the next subsection, the *utilization* calculation of each routing shape is discussed. For the convenience of analysis, assuming that for one pin-pair bounding box, one pin is located in the lower left corner and the other in the upper right of the box. Other configurations follow the same logic and accordingly, similar results can be obtained. Note that when referring to the position of a grid tile, the lower-left corner coordinate of that tile is used.

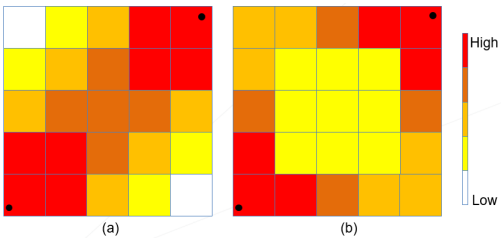


Figure C.4: Congestion distribution in different algorithms, (a) Lou *et al.* [74], and (b) the proposed AFM.

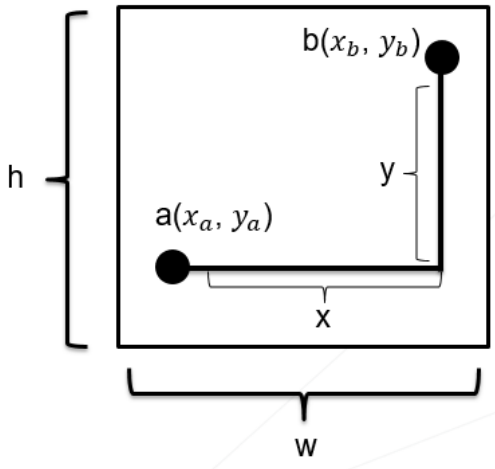


Figure C.5: An example of a short shape.

The Short shape

The short-shape is the simplest of all routing shapes, where two pins are both located in the same tile as shown in Fig. C.5, no other routing shapes are possible to be formed except the short shape. Here the *utilization* contribution is calculated by Equations C.1 and C.2:

$$U_h = \frac{x}{w} = \frac{x_b - x_a}{w} \tag{C.1}$$

$$U_v = \frac{y}{h} = \frac{y_b - y_a}{h} \tag{C.2}$$

where U_h is the horizontal *utilization* of two vertical edges of the tile, and U_v is the vertical *utilization* of two horizontal edges of the tile.

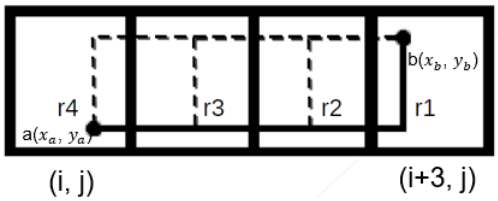


Figure C.6: An example of a horizontal shape.

Horizontal and Vertical shapes

Fig. C.6 shows the net in a horizontal routing shape, which only passes through one row. If one net occupies just one column, it is called a vertical routing shape instead. The two routing shapes follow the same logic, therefore only the analysis of the horizontal shape is shown. The horizontal *utilization* U_h for the left and right tiles are respective length fractions ($(x_{i+1} - x_a)/w$ and $(x_b - x_{i+2})/w$), and *utilization* is set to be 1 for all other middle tiles. The vertical *utilization* U_v is calculated using Equation C.3:

$$U_v = \frac{y}{h} \times P_{r_n} = \frac{y_b - y_a}{h} \times P_{r_n} \tag{C.3}$$

where P_{r_n} is the probability of choosing the corresponding route r_n over all route candidates. A detailed explanation of P_{r_n} configuration is given in Subsection C.1.2. The vertical portion of the net could be routed in any of the tiles, and it is depended on the route chosen.

The L shape

A net passing through more than two rows and two columns will have at least one bend and has two possible routes, r_1 and r_2, as shown in Fig. C.7. If the probability of choosing r_1 is $P(r_1)$, then $(1 - P(r_1))$ will be the probability of choosing r_2. The lower-left and upper-right tiles have fraction *utilization* contributions and all other tiles will have the *utilization* contributed to the corresponding direction with their probability $P(r_n)$. For example, the *utilization* of the lower-left tile is:

$$U_v = \frac{y}{h} \times P_{r_2} = \frac{y_a - y_j}{h} \times P_{r_2} \tag{C.4}$$

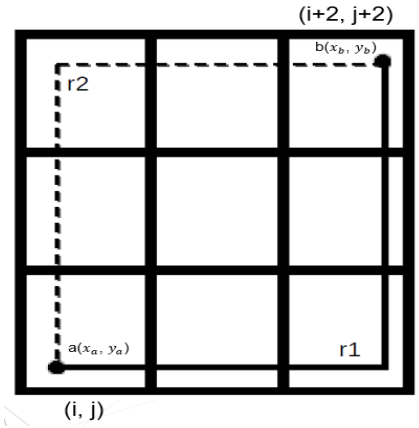


Figure C.7: An example of a L shape.

$$U_h = \frac{x}{w} \times P_{r_1} = \frac{x_{i+1} - x_a}{w} \times P_{r_1} \tag{C.5}$$

A middle tile (x_{i+2}, y_{j+1}) has *utilization*:

$$U_v = \frac{y}{h} \times P_{r_1} = P_{r_1} \tag{C.6}$$

Normally, an L shape is combined with a Z shape during the calculation of *utilization*. In special cases, when a net crosses exactly two rows and two columns, only an L shape is considered since no Z shape is routable.

The Z shape

If either the number of rows or columns is greater than 2 and the other is greater than 1, the Z shape applies. This is the most complicated routing shape, all possible routes are categorized into two sets, vertical routes and horizontal routes, decided by the direction of the middle portion of the route to which both pins are not directly connected. For example, as shown in Fig. C.8, vertical (r_1 and r_2) and horizontal r_3.

In order to better analyze how this shape works, the *utilization* of one route is separated into different portions. Assume one net r_n is from the vertical set, it can be partitioned into: the first tile of bottom row $U^{b,i}(r_n)$ and the last tile of top row

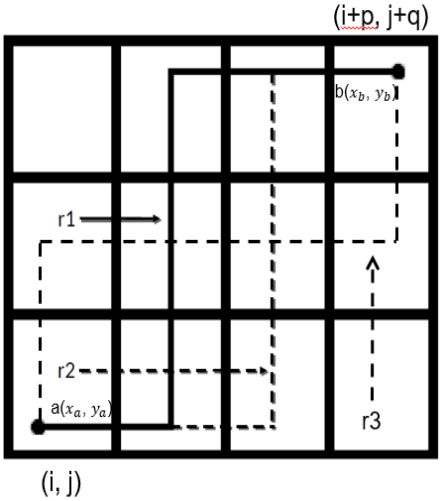


Figure C.8: An example of a Z shape.

$U^{t,i+p}(r_n)$ (where two pins located); the first tile of top row $U^{t,i+n}(r_n)$ and the last tile of bottom row $U^{b,i+n}(r_n)$ (where the bend occurred); all other tiles

Bottom and top row tiles (except the tile in which the bend occurred) have horizontal contribution only, while the middle column tiles have the opposite. Their *utilization*s are expressed as follows:

$$\begin{aligned} U_h^{b,i}(r_n) &= \frac{x}{w} \times P_{r_n} = P_{r_n} \\ U_h^{b,i}(r_n) &= \frac{x}{w} \times P_{r_n} = P_{r_n} \\ U_v^{m,(i,j)}(r_n) &= \frac{y}{h} \times P_{r_n} = P_{r_n} \end{aligned} \tag{C.7}$$

Another portion is where the source and sink are located. The contributed *utilization*s is based on the length fraction:

$$\begin{aligned} U_h^{b,i}(r_n) &= \frac{x_{i+1} - x_a}{w} \times P_{r_n} \\ U_h^{t,i+p}(r_n) &= \frac{x_b - x_{i+p-1}}{w} \times P_{r_n} \end{aligned} \tag{C.8}$$

Finally, the location where the bend occurs is noted. Assume the change of direction is in the middle of a tile (This also is the behavior of global routers that

are being mimicked), which means the *utilization* will be 0.5 for the horizontal and a corresponding length fraction for the vertical. The reason 0.5 is used here is also to better mimic the global router's behavior. The "pattern routing" [58] generally bends a wire in the central line of a grid tile. Therefore, the total *utilization*s can be obtained (vertical and horizontal) as:

$$\begin{aligned} U^{b,i+n}(r_n) &= (\frac{y_{j+1} - y_a}{h} + 0.5) \times P_{r_n} \\ U^{t,i+n}(r_n) &= (\frac{y_b - y_{j+q-1}}{h} + 0.5) \times P_{r_n} \end{aligned} \tag{C.9}$$

From the case in Fig. C.8, $p = 3$ and $q = 2$, the route r_1 does not have $U_b^x(r_n)$, after subtracting the tiles from both bottom and top rows where source, sink and bend are located. So the *utilization*s of the route r_1 are:

$$\begin{aligned} U_h^{t,i+2}(r_1) &= \frac{x}{w} \times P_{r_1} = P_{r_1} \\ U_v^{m,j+1}(r_1) &= \frac{y}{h} \times P_{r_1} = P_{r_1} \\ U_h^{b,i}(r_1) &= \frac{x_{i+1} - x_a}{w} \times P_{r_1} \\ U_h^{t,i+3}(r_1) &= \frac{x_b - x_{i+2}}{w} \times P_{r_1} \\ U_v^{b,i+1}(r_1) &= (\frac{y_{j+1} - y_a}{h} + 0.5) \times P_{r_1} \\ U_v^{t,i+1}(r_1) &= (\frac{y_b - y_{j+q-1}}{h} + 0.5) \times P_{r_1} \end{aligned} \tag{C.10}$$

The calculating of the *utilization* for other routes would be similar to the above, but not explicitly demonstrated here due to space limitations.

There is a likelihood factor λ that decides the probability of selecting an L shape routing shape instead of a Z shape routing shape for large pin-pair bounding boxes. For better comparison, let $\lambda = 0.6$, which is the same value used by Westra *et al.* [111].

Probability Distribution metric

Westra *et al.* [111] proposes a uniform distribution of routing shapes, which does not reflect practical routing behavior. Normally, routers are congestion-aware; they

adjust the decision based on prior congestion information. When one path is highly congested, routers will avoid worsening the situation by searching for other possible paths. In contrast, in the work of Westra *et al.*, any existing congestion is ignored and *utilizations* of new nets are continually added to already congested tiles, even if there is opportunity for a more preferable non-congested path.

In AFM, a key factor is introduced, that is referred to as grid tile availability. The availability of a tile located at (i, j) of the grid is expressed as $A_{(i,j)}$. The possibility of one route being chosen depends on how many resources are still available in that specific path. The availability of one route $A(r_n)$ is the summation of all exclusive tiles on its route. An exclusive tile on one route means it will not be passed through by any other possible routes of the same pin-pair. There will be a better chance of choosing a route with tiles having a high availability, while highly congested (lower availability) tiles result in a heavy penalty.

Assuming a pin-pair bounding box has lower-left corner coordinates (i, j) and upper-right corner coordinates $(i+p, j+q)$, the availability of each potential route as the value of total *utilization* divided by total *capacity* of all exclusive grid tiles that the route has passed through can be expressed as:

$$A(r_n) = \frac{\sum U_{x,y}}{\sum C_{x,y}}, (i \leq x \leq i+p, j \leq y \leq j+q) \tag{C.11}$$

The probability of each route can then be obtained as:

$$P_{r_n} = \frac{A_{r_n}}{\sum A} \tag{C.12}$$

where $\sum A$ is the summation of availability of all route candidates of a pin-pair.

In the example shown in Fig. C.9, the possibility of selecting each route can be calculated as:

$$\begin{aligned} P_{r_1} &= \frac{A_{r_1}}{A_{r_1} + A_{r_2}} = \frac{100}{150} \approx 0.67 \\ P_{r_2} &= \frac{A_{r_2}}{A_{r_1} + A_{r_2}} = \frac{50}{150} \approx 0.33 \end{aligned} \tag{C.13}$$

Such that, $\sum P = 1$.

The net ordering slightly affects the estimation result if the tile *capacity* is limited, routing shortest nets first yields lower congestion map, it is because longer

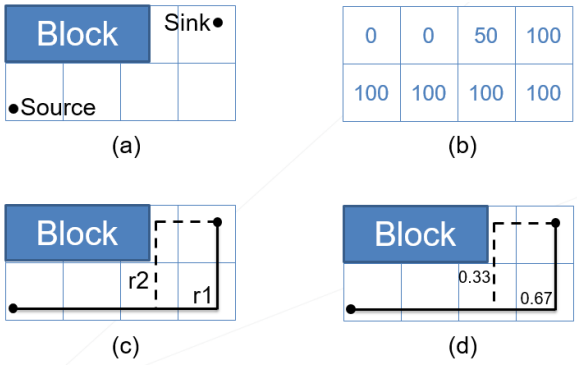


Figure C.9: (a) a pin-pair bounding box with blockage; (b) availability of each grid tile; (c) Possible paths; (d) respective availabilites.

nets have more options to route to avoid congested areas. There are two special cases that must be considered here. First, if one route has zero overall availability, or any of the grid tile in between has zero availability, then the entire route should be eliminated out from the candidate set of routes. This means that the tiles are blocked or are congested such that the entire route, which includes those tiles, are bad candidates. They must be avoided from being selected to route. Second, if all possible routes have zero overall availability, then the *utilization* will be uniformly distributed, because finding an optimal route would no longer be possible.

C.1.3 Adaptive Congestion Estimation (ACE) Algorithm

The process of estimating congestion begins with constructing the appropriate grid, the size of which is normally decided by the scale of the design and other factors.

The adaptive congestion estimation (ACE) algorithm, and the probability determination used are defined as follows:

The first step of this algorithm is to partition the multi-pin net into two-pin pairs. To perform this, the Minimum Spanning Tree (MST) technique is used. Another proper partition structure is Rectilinear Steiner Tree (RST). But if RST constructed during estimation phase is different from that during actual routing, the result may become even less efficient than if using MST. The ideal timing complexity of MST is $O(p \log p)$, where p is the number of total pin-pairs in one net. However, Prim's algorithm with complexity $O(p^2)$ is simpler to implement

Algorithm 2 Algorithm ACE

```
1:  procedure GENERATE CONGESTION MAP
2:      for each net n ∈ N do
3:          pair(a,b) = MST(n)
4:          for each pair(a,b) do
5:              shape = determineShape( grid, pair(a,b) )
6:              prob_vector = probabilityGenerator( grid, pair(a,b), shape )
7:              updateCongestionMap( grid, pair(a,b), shape, prob_vector )
8:          end for
9:      end for
10: end procedure
```

Figure C.10: The adaptive congestion estimation algorithm.

Algorithm 3 Route Probability Calculation

```
1:  procedure PRODUCE PROBABILITY DISTRIBUTION ARRAY
2:      for each path do
3:          for each tile in a path do
4:              if tile is only used in this path then
5:                  sum += avail
6:                  path_avail[current] += avail
7:              end if
8:          end for
9:      end for
10:     for each path do
11:         prob_vector[current] = path_avail[current]/sum
12:     end for
13: end procedure
```

Figure C.11: The algorithm of calculating the probability of choosing candidate routes of a given pin-pair.

and is commonly used as an alternative in practice. After obtaining the full set of pin-pairs, each pair is categorized into the shapes it fits, followed by the probability calculation of every possible route. Once this necessary information is gathered, the congestion map can then be updated.

Assuming that c is the total number of tiles in the design grid. The computation before the start of ACE needs an additional $O(c)$ to calculate the $capacity$ and to store in the grid map. If partial blockage occurs, $capacity$ is reduced by the amount in overlapped areas.

The number of shapes is fixed to 5 as described in previous sections. Therefore, the timing complexity for determining the pin-pair routing shape is $O(1)$. In a worst case scenario, every pin-pair bounding box would overlap with the entire design space, which would then trend towards $O(c \times p)$. In this case, after obtaining the estimated $utilization$, the design requires another $O(c \times p)$ to update the $utilization$s to the congestion map. In summary, for the ACE algorithm, the total computation of a design with n nets is $O(n(p^2 + cp))$.

In the relevant text authored by Saxena *et al.* [94] they mention that the total number of pins, p, in one net is usually a constant in practice due to circuit optimization fan-out constraints. If this is the case, the computation can be reduced to $O(nc)$, which means that in a given netlist with restricted fan-out rules, the complexity of the ACE algorithm is linear with respect to the design grid density.

C.2 Experimental Results

AFM was implemented in C++ and was run on a machine with i5-4200 CPU (1.6Ghz×4) and 12GB memory. Five benchmarks (cordic, mips9, leon, tds and sb1) were used for testing and comparison, where the first four benchmarks are real industrial chips and the sb1 design is provided by ISPD11 [109] contest. The converter [72] is used to convert the academic benchmark into DEF/LEF format so that it can be routed in the same commercial router (NanoRout® from Candence® Encounter®) as the other designs were. The proposed algorithm is applied to estimate the congestion using the placement only. For comparison, the estimated congestion map is generated first by using AFM and Westra's after the placement stage, and then retrieved the actual congestion map after Encounter [15] routing. By comput-

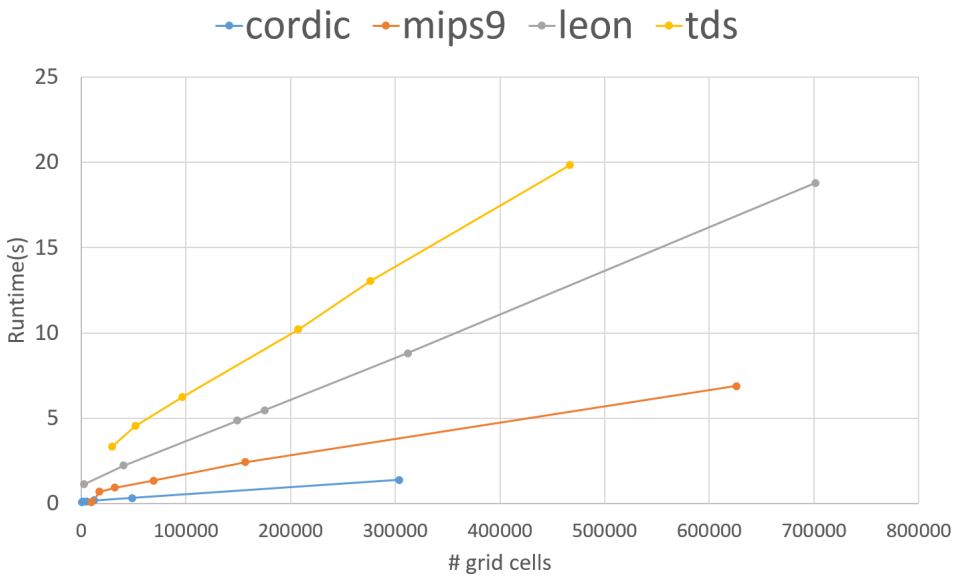


Figure C.12: The plot of number of tiles vs. runtime using the proposed AFM on four benchmarks.

ing the difference between grid-tile-wise congestion value of post-routing and the estimated congestion, an error percentage can be obtained to represent how close algorithms guess the routability of the design.

Because the linear timing assumption is based on the real design fan-out constraints. To verify timing, four benchmarks are tested that follow the fan-out constraints, and the runtime is recorded.

Fig. C.12 shows a comparison plot between the number of grid tiles and algorithm runtime, where it is clear that the proposed approach is running in linear time with respect to the total grid tile number. It is also observable from the chart that, for a given value in x-axis (number of tiles), the value of y-axis represents the relationship of net quantities of each design. The smallest design, *cordic*, has the lowest overall runtime, which is represented as the bottom line in the chart, and the line of *tds* is on top of all other designs since it is the largest of all four designs. Benchmark sb1 has very large fan-out nets, which means the time complexity will not follow the linear assumption.

Table C.1 summarizes the results of AFM and compares it with the previous

work from [111]. An error percentage between the previous work and the AFM is measured to validate the quality. The AFM reduces the error percentage for all five circuits. Standard derivation σ, although AFM is 10.69% worse for circuit "tds", it is better or almost the same for all the other designs. In benchmarks used by AFM, The interconnects of many nets are fallen into a straight line on the design, which means the area of the bounding box of those nets are zeros. Such cases cannot be considered when using RUDY [100]. The RUDY has huge error percentage due to incalculable RUDY score, therefore it is not shown in the comparison chart.

Table C.1: Results and comparison of the AFM with previous work

Design	#nets	Westra's		AFM		Improvement%	
		error%	σ	error%	σ	error%	σ
cordic	15470	5.662	1.666	2.914	1.547	+48.54%	+7.15%
mips	93777	8.390	8.654	6.841	8.745	+18.57%	-0.95%
leon	165018	5.99	1.891	5.483	1.571	+8.57%	+16.93%
tds	421949	7.78	3.503	6.964	3.903	+10.49%	-10.69%
sb1	822744	9.556	8.825	9.041	8.751	+5.39%	+0.84%
Average:		7.476	4.908	6.248	4.903	+19.37%	+2.49%

Fig. C.13 shows the visualized congestion map of tds and sb1. Congested hot spots are well predicted, large blockages in sb1 are avoided by the router, and the estimation also presents the same blockage-aware behavior. Considering these reliable results and the linear runtime for some circuits, the information from AFM can also be used in placement iterations or other tight loops for routability adjustment. Note that the estimation algorithm overestimates the actual congestion value in most regions due to the distribution mechanism. In reality, one net only contributes *utilization* to one specific path and other candidates remain unchanged, while ACE and all other algorithms tend to increase the *utilization* for every possible path. This issue can be dealt with by applying other processes after the estimation stage. For example, a congestion re-distribution [94] is discussed; however, any such extra action taken comes at the cost of runtime increases.

The congestion map of a global-routing-based algorithm is heavily affected by the grid tile size, because such algorithm counts only routes that pass through tile

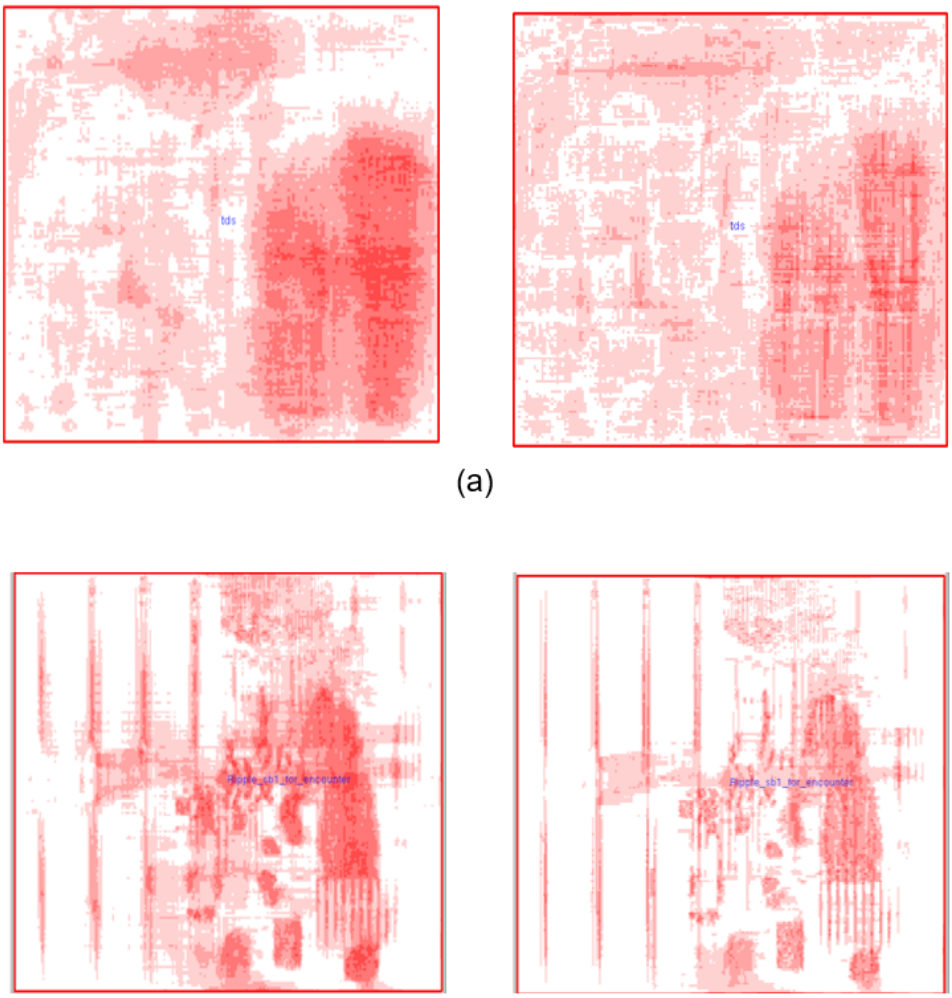


Figure C.13: Congestion map comparison. (a) left: actual congestion of tds; right: estimated congestion of tds, (b) left: actual congestion of sb1; right: estimated congestion of sb1.

edges but not within a tile. In other words, larger grid tile size worsen congestion estimation by global-routing based algorithms because routes that crossed tile edges are now included in larger tiles and wiring information is abandoned. However, AFM can accurately incorporate the wiring situation inside grid tiles, which

provides a result that has no penalty from grid tile sizing. Fig. C.14 shows that the congestion estimation of two circuits (tds and sb1) in two different grid tile sizes. The pattern is well preserved, no congested hot spot is lost, because of the local-congestion-awareness of AFM. Fig. C.15 shows the hot spot estimated by FAM matches the result after global routing by Candence® Encounter®, which suggests that AFM successfully mimics the behavior of the a global router.

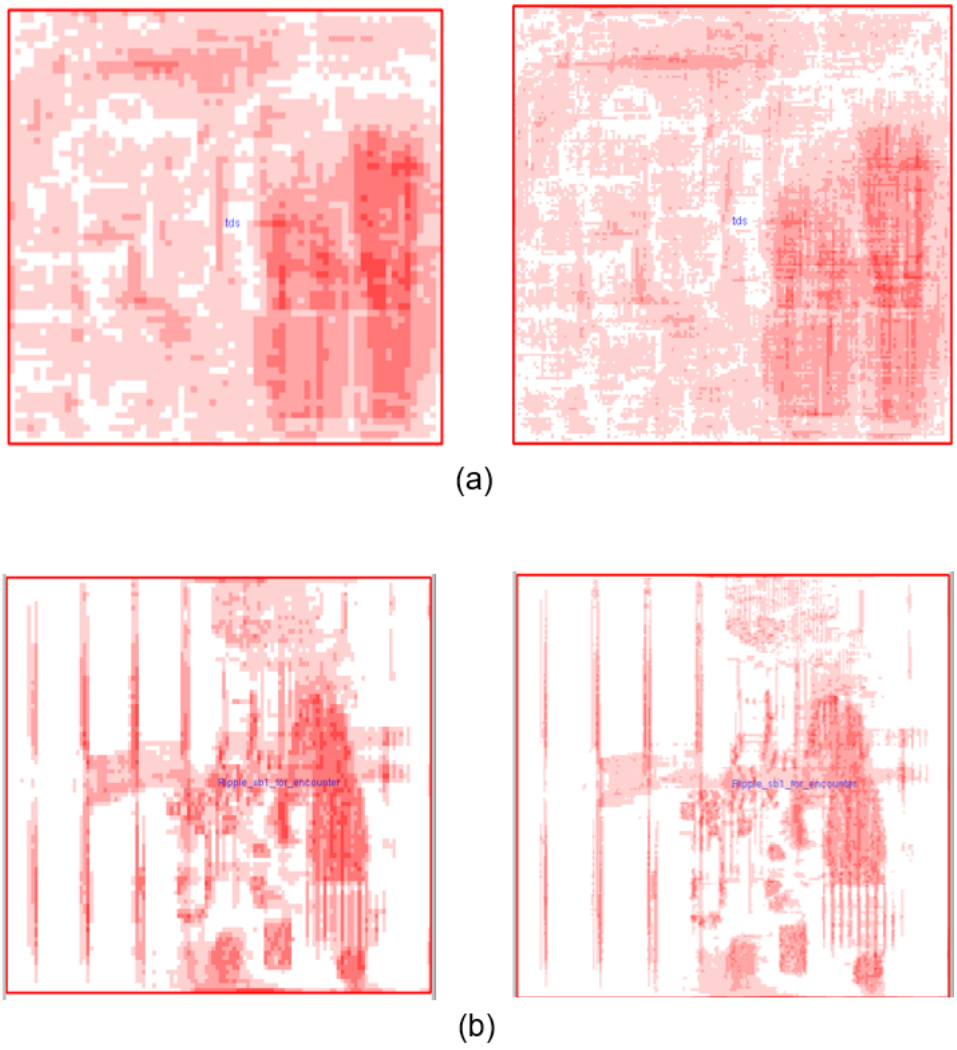


Figure C.14: Congestion map produced by AFM using different tile sizes.

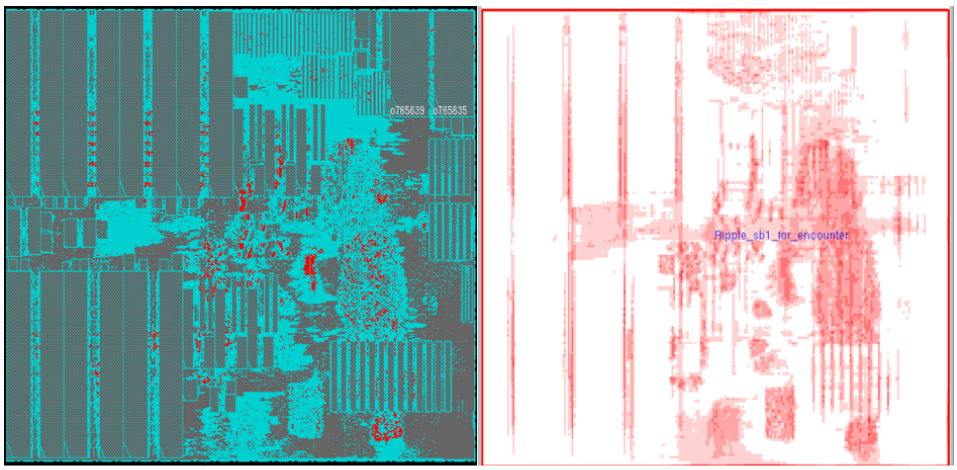


Figure C.15: (a) Congestion map by ”trial route” in Cadence® Encounter®, red is congestion (b) Congestion Estimation from the proposed AFM. Overflow areas in (a) are also the brightest in (b).