UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LUIS ENRIQUE MURILLO VIZCARDO

NEATRouter: A New Tool for 2D Global Routing

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Ricardo Augusto da Luz Reis

Porto Alegre
July 2024

CIP — CATALOGING-IN-PUBLICATION

Murillo Vizcardo, Luis Enrique

NEATRouter: A New Tool for 2D Global Routing / Luis Enrique Murillo Vizcardo. – Porto Alegre: PPGC da UFRGS, 2024.

71 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2024. Advisor: Ricardo Augusto da Luz Reis.

1. VLSI. 2. EDA. 3. Global Routing. 4. Physical Design. 5. NEAT algorithm. 6. Genetic Algorithm. 7. Microelectronics. I. da Luz Reis, Ricardo Augusto. II. Título.

“I am not in competition with anyone but myself. My goal is to improve myself continuously.”

— Bill Gates

ACKNOWLEDGMENT

I would like to express my gratitude to all the people who contributed to completing this work. Firstly, to my parents Jaime Felipe Murillo Galarza and Laura Susana Vizcardo Zevallos for their unconditional support and understanding during this stage of my academic life.

I am sincerely grateful to my advisor Dr. Ricardo Reis for the opportunity he gave me, for his guidance and support throughout the master's degree process. Also, I would like to thank all my professors and laboratory colleagues at UFRGS, especially Eder Monteiro and Mateus Fogaca for the knowledge transmitted during this academic stage.

Finally, I would like to thank Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) for the financial support.

NEATRouter: Uma nova ferramenta para roteamento global 2D

RESUMO

Devido ao avanço tecnológico em diversas áreas da vida cotidiana, a demanda por circuitos integrados mais complexos e compactos está em constante crescimento. Para abordar o projeto de tais circuitos, é essencial a inovação contínua em ferramentas de Automação de Projeto Eletrônico (EDA), que contribuem para reduzir os custos de fabricação e, consequentemente, os custos dos chips. Neste trabalho, apresentamos um novo algoritmo para a etapa de roteamento global de um circuito integrado. Nosso método usa o algoritmo Neuroevolution of Augmenting Topologies (NEAT) para gerar redes neurais capazes de encontrar o caminho mais curto e mais eficiente em termos de recursos para uma rede de 2 pinos. Os resultados de nossos experimentos sugerem que este método pode competir com sucesso com abordagens tradicionais como MazeRouter, gerando roteamentos 2D que se equalizam em termos de comprimento de fio e reduzem de 1% a 5% no uso dos recursos disponíveis de nossos casos de teste.

Palavras-chave: VLSI, EDA, Roteamento Global, Projeto físico, algoritmo NEAT, algoritmos genéticos, Microeletrônica.

ABSTRACT

Due to technological advancement in various fields of daily life, the demand for more complex and compact integrated circuits is constantly growing. To address the design of such circuits, continuous innovation in Electronic Design Automation (EDA) tools is essential, which contribute to reducing manufacturing costs and, consequently, chip costs. In this work, we present a novel algorithm for the global routing stage of an integrated circuit. Our method uses the Neuroevolution of Augmenting Topologies (NEAT) algorithm to generate neural networks capable of finding the shortest and most resource-efficient path for a 2-pin net. The results of our experiments suggest that this method can successfully compete with traditional approaches such as MazeRouter, generating 2D routings which equalize in terms of wirelength and reduce from 1% to 5% in the use of available resources of our testcases.

Keywords: VLSI. EDA. Global Routing. Physical Design. NEAT algorithm. Genetic Algorithm. Microelectronics.

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
ANN	Artificial neural network
BFS	Breadth First Search
DEF	Design Exchange Format
DQN	Deep Q network
DRL	Deep Reinforcement Learning
EDA	Electronic Design Automation
GA	Genetic Algorithm
GR	Global Routing
HPWL	Half Perimeter Wirelength
ICCAD	International Conference on Computer-Aided Design
ILP	Integer Linear Programming
ISPD	International Symposium on Physical Design
LEF	Library Exchange Format
ML	Machine Learning
NEAT	Neuroevolution of Augmenting Topologies
RL	Reinforcement Learning
RSMT	Rectilinear Steiner Minimum Tree
VLSI	Very-large-scale integration
WL	Wirelength

LIST OF FIGURES

Figure 2.1 Modern chip design flow (HUANG et al., 2021). This flow describes the steps from the definition of the requirements to the final manufacturing of the chip.....22
Figure 2.2 Representation of a grid and the gcells in global routing (HE et al., 2019). The routing area is divided into a cells where each cell is commonly called a gcell, then the global routing is constructed by a sequence of gcells which connect the pins of a net.....24
Figure 2.3 Architecture of an artificial neuron and a multilayered artificial neural network (ABRAHAM, 2005). A multilayered artificial neural network (b) is composed of an input layer that receives the information, hidden layers that will perform the operations and an output layer that will produce the final result. Each hidden layer is composed of a set of artificial neurons (a) which will receive the values and calculate an output using an activation function.....26
Figure 2.4 Flow of GA algorithm (HALDURAI; MADHUBALA; RAJALAKSHMI, 2016) 27
Figure 2.5 Structure of a genome (STANLEY; MIIKKULAINEN, 2002). The genome is composed of a list of the nodes and connections of the neural network. 28
Figure 2.6 Crossover operation between two NEAT genomes (STANLEY; MIIKKULAINEN, 2002) 29
Figure 2.7 Mutation operation on NEAT algorithm (STANLEY; MIIKKULAINEN, 2002) 30

Figure 3.1 FastRouter algorithm flow (PAN et al., 2012) 34
Figure 3.2 CUGR algorithm flow (LIU et al., 2020) 35
Figure 3.3 Pipeline for solving global routing with DQN (LIAO et al., 2019).....37

Figure 4.1 General Flow of NEATRouter 39
Figure 4.2 Ripup and Reroute Flow 40
Figure 4.3 Representation of the input values for the NEAT neural network. The gray, green, blue, and black gcells represent, respectively, the source pin position, the destination pin position, the neighbors of the current position, and the gcells that have no available resources.The green arrows represent the number of gcells in the four directions, from which the minimum resource value will be taken, and the red arrows represent gcells in the four positions, from which the maximum historical value will be taken. 43
Figure 4.4 Nets used for training NEAT neural network. All nets have their source pin located in the blue gcell, while their target pins are distributed in various directions to train the NEAT neural network for a wide diversity of scenarios. 45

Figure 5.1 Comparison of congested gcells using MazeRouter and NEATRouter for Test 5 50
Figure 5.2 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 5. 51
Figure 5.3 Comparison of congested gcells using MazeRouter and NEATRouter for Test 7 51

Figure 5.4 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 7. 52
Figure 5.5 Comparison of congested gcells using MazeRouter and NEATRouter for Test 11 53
Figure 5.6 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 11. 54

Figure A.1 Comparison of congested gcells using MazeRouter and NEATRouter for Test 1 61
Figure A.2 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 1 61
Figure A.3 Comparison of congested gcells using MazeRouter and NEATRouter for Test 2 62
Figure A.4 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 2 62
Figure A.5 Comparison of congested gcells using MazeRouter and NEATRouter for Test 3 63
Figure A.6 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 3 63
Figure A.7 Comparison of congested gcells using MazeRouter and NEATRouter for Test 4 64
Figure A.8 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 4 64
Figure A.9 Comparison of congested gcells using MazeRouter and NEATRouter for Test 6 65
Figure A.10 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 6....65
Figure A.11 Comparison of congested gcells using MazeRouter and NEATRouter for Test 8 66
Figure A.12 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 8....66
Figure A.13 Comparison of congested gcells using MazeRouter and NEATRouter for Test 9 67
Figure A.14 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 9....67
Figure A.15 Comparison of congested gcells using MazeRouter and NEATRouter for Test 10 68
Figure A.16 Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 10....68

LIST OF TABLES

Table 5.1 Characteristics of the test cases generated for the experiments. Test case is the name of the test, Grid size is the size $N \times N$ of the routing area, Net number is the number of nets that the test has, Layer number and Layer capacity is the number of available layers and the capacity of each layer respectively. 48
Table 5.2 Comparison of the results using MazeRouter algorithm and NEATRouter. The WL is the wirelength used to route all nets, #iter is the number of iterations that the algorithm performed to remove congestion, Avg. used is the average used resources of a ggrid and runtime is the time that takes the algorithm to do the total routing. 49

CONTENTS

1 INTRODUCTION 19
2 FUNDAMENTAL CONCEPTS AND BACKGROUND 21
2.1 Electronic Design Automation (EDA) 21
2.2 Physical Design 21
2.3 Global Routing 23
2.4 Machine Learning 24
2.5 Artificial Neural Network 25
2.6 Genetic Algorithms 26
2.7 Neuroevolution of augmenting topologies algorithm (NEAT) 28
3 RELATED WORK 31
3.1 EDA algorithms developed in the research laboratory 31
3.2 Existing methods to solve the Global routing problem 33
3.3 Methods that apply Maching Learning algorithms to the Global Routing problem 36
4 NEAT TO GLOBAL ROUTING 39
4.1 Implementation of MazeRouter 41
4.2 History calculation for a gcell 41
4.3 NEAT neural network 41
4.3.1 Initial Configuration 42
4.3.2 Input nodes 42
4.3.3 Output nodes 42
4.3.4 Fitness Function 43
4.3.5 Trainning method 45
4.4 Routing with NEAT neural network 46
5 EXPERIMENTS AND RESULTS 47
5.1 Dataset 47
5.2 Experimental Results 47
5.3 Performance Evaluation 48
5.4 Results 48
5.5 Discussion 49
6 CONCLUSION AND FUTURE WORKS 55
REFERENCES 57
APPENDIX A — CONGESTION AND INDIVIDUAL COMPARISON GRAPHS FOR THE TEST CASES 61
A.0.1 Test 1 61
A.0.2 Test 2 62
A.0.3 Test 3 63
A.0.4 Test 4 64
A.0.5 Test 6 65
A.0.6 Test 8 66
A.0.7 Test 9 67
A.0.8 Test 10 68
APPENDIX B — RESUMO EXPANDIDO EM PORTUGUÊS 69

1 INTRODUCTION

At this time, technology is having a significant impact on various aspects of daily life. For example, in the field of health, the use of artificial respirators has been introduced, while in transportation, we already have vehicles capable of driving without human intervention. In research, the development of increasingly powerful computers allows complex calculations to be carried out, and in the field of entertainment, smartphones have become essential tools for staying informed. This technological integration into our daily lives has generated the need to improve the manufacturing of integrated circuits. These components, which contain millions of elements to perform complex operations, must also occupy a small space to be used in compact devices, such as hand watches.

Electronic Design Automation (EDA) is the field charged with developing software to support the automation of integrated circuit design. Since the manufacturing costs of integrated circuits are increasing due to their high complexity, innovation in EDA technologies is required. In the design process of an integrated circuit, the routing task plays a crucial role, since it is responsible for establishing the connections between the signals of the pins of an integrated circuit using the metal layers. However, performing routing is complex because integrated circuits are composed of millions of components in very small areas. For this reason, the routing stage is divided into two sub-stages: global routing and detailed routing. The global routing stage is responsible for performing the initial routing of all the nets of an integrated circuit, reducing both wirelength and congestion. This first routing is known as guides, which then serve as input to the second stage. In detailed routing, guides are used as a guide to perform the final routing, verifying the design rules. In this way, the definitive routing is obtained for each net of the integrated circuit.

Within the global routing stage, multi-pin nets are divided into 2-pin segments, which are routed separately. This simplifies the routing task, since the congestion and wirelength of each segment of the entire net can be controlled separately. Algorithms like MazeRouter (LEE, 1961) and its variants, which find the shortest path between two points, are used to perform routing of these segments. MazeRouter has several advantages, such as its ease of implementation and its ability to give better results in terms of wirelength. Among its disadvantages is that in large areas routing can be computationally expensive in terms of time and memory. Furthermore, it does not always consider other important factors to optimize results.

Machine Learning (ML), a field of artificial intelligence, offers systems capable of learning concepts without the need for explicit programming. This process is carried out most of the time through operations executed in a neural network, which analyze the information provided to identify patterns, thus facilitating the generation or prediction of new information. Thanks to its effectiveness and the continuous improvement of the results obtained, ML techniques are applied in various fields, such as engineering, medicine, video games, finance, marketing, entertainment, and many others.

In this work, we propose NEATRouter as a two-dimensional global routing algorithm for 2-pin nets, with the objective of generating high-quality routes in terms of wirelength and congestion. Our algorithm focuses on complementing/replacing the MazeRouter algorithm, which is used in several state-of-the-art global routing tools. This approach is based on the Neuroevolution of Augmenting Topologies (NEAT) algorithm, a variant of genetic algorithms designed to generate neural networks capable of optimizing specific problems. We adapt NEAT algorithm to train neural networks that optimize two-dimensional routing of simple 2-pin nets. These neural networks will determine the set of movements necessary for routing a net, with the goal of reducing congestion and wirelength throughout the routing area. To address congestion, we implemented the Rip-up and Re-route (R&R) algorithm, which, in case of congestion, repeats the routing of all nets, increasing the cost of using congested areas. This approach allows nets to explore alternative paths, releasing resources for other nets. In each iteration of Rip-up and Re-route, we apply the fittest neural networks trained by the NEAT algorithm for routing. In case the method cannot route a net, we resort to the MazeRouter algorithm to ensure the connection of all nets. In summary, our proposal combines the power of NEAT algorithm, genetic algorithms, and classical routing techniques such as MazeRouter to effectively address global routing of 2-pin nets, prioritizing routing quality and runtime efficiency.

We perform a comparison between the results obtained using our algorithm and those obtained by exclusively using the MazeRouter routing method. This evaluation was carried out considering crucial aspects such as wirelength, resources used and runtime. Our results demonstrate that our algorithm has the potential to optimize the utilization of routing resources. In some cases, we see improvements even in wirelength, although this benefit may be accompanied by an increase in runtime. For our algorithm to deal with cases with a larger number of nets (real designs), it must be complemented by an algorithm that divides the total number of nets into sets, which will be routed by NEATRouter and then perform global routing of all the sets.

2 FUNDAMENTAL CONCEPTS AND BACKGROUND

In this chapter, we provide essential information on the key concepts and methodologies relevant to our research. We begin by defining Electronic Design Automation (EDA) and explore Physical Design and its stages. We then highlight the importance of global routing, addressing its challenges and the traditional algorithms used at this stage. In addition, we present definitions of Machine Learning, Artificial Neural Networks and Genetic Algorithms, necessary to understand the NEAT algorithm. Similarly, we analyze in detail the NEAT algorithm, its operations and applications. With this information, we seek to establish a solid foundation for understanding our work.

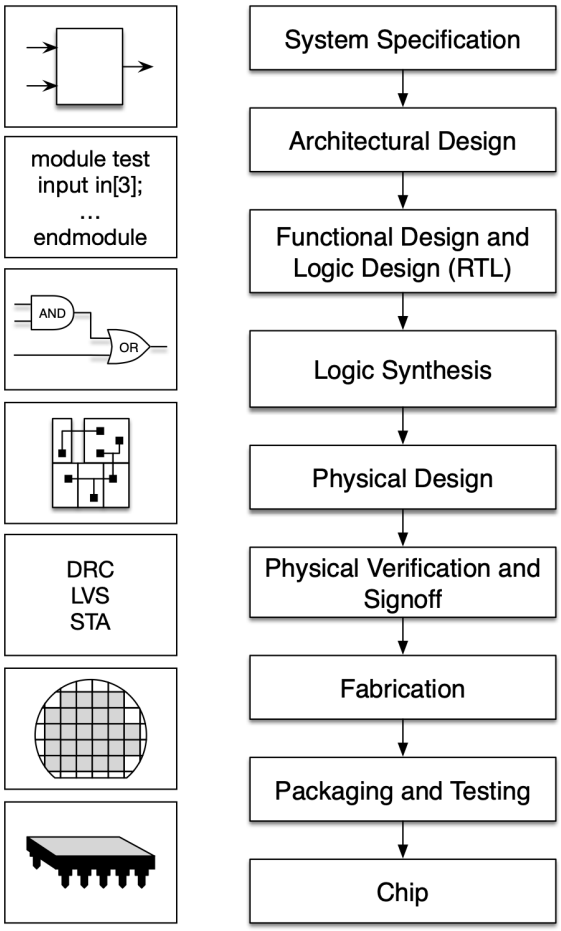
2.1 Electronic Design Automation (EDA)

Electronic Design Automation (EDA) is one of the most important areas in electronic engineering. In the past few decades, it has been witnessed that the flow of chip design became more and more standardized and complicated (HUANG et al., 2021). The EDA is composed of a set of software and hardware tools that play a fundamental role in automating the various stages within the electronic circuit design industry. The modern circuit design flow is shown in Figure 2.1, this flow includes requirements definition, logical design, physical design, simulation, verification, and finally manufacturing. EDA tools help simulate and analyze the behavior of a circuit before it is physically manufactured. This allows potential problems to be identified and corrected at an early stage of design, saving costs and time in final production. Due to the complexity of each circuit, the existence of loops in the flow is increasingly common, which serve to recalculate a given solution with new information. For example, within the physical design, in the global routing stage a result without congestion may not be found when all the nets are routed, so it will be necessary to re-execute the positioning with this new information and thus modify the positions of some components to be able to solve the problem.

2.2 Physical Design

In the physical design stage all macros, cells, gates, transistors, etc., with fixed shapes and sizes per fabrication layer are assigned spatial locations (placement) and have

Figure 2.1: Modern chip design flow (HUANG et al., 2021). This flow describes the steps from the definition of the requirements to the final manufacturing of the chip.



appropriate routing connections (routing) completed in metal layers. The result of physical design is a set of manufacturing specifications that must subsequently be verified (KAHNG et al., 2011).

In this phase of the process, we meticulously carry out tasks while strictly following established design rules. These rules encapsulate the inherent physical constraints within the manufacturing environment. They are crucial for ensuring an accurate and efficient representation of the circuit, taking into account the limitations imposed by the production medium. Within this context, every decision regarding the arrangement of functional blocks, the interconnection of elements, and the distribution of electrical power is critical. These decisions collectively contribute to achieving a final design that not only meets stringent performance requirements but also optimizes the manufacturability of the integrated circuit.

The physical design phase directly and significantly impacts various key aspects of the circuit, such as its performance, area, power consumption, reliability, and manufacturing yield. An illustrative example of this influence is the presence of extensive routing in the

circuit, as this inevitably results in prolonged signal delays.
Due to its high complexity, physical design is divided into several stages:

- *Partitioning*: Divides the circuit into small modules, which can be analyzed separately.
- *Floorplanning*: Involves determining both the shape and arrangement of the modules, and also addresses the allocation of locations for external ports and macro blocks.
- *Power and ground routing*: Distributes the power and ground nets across the chip.
- *Global Placement*: Find the approximate positions of each component, without considering details such as overlaps.
- *Detailed Placement*: Find the exact positions of each component, which seeks to meet specific constraints.
- *Clock network synthesis*: Determines the buffering, gating and routing of the clock signal to meet prescribed skew and delay requirements.
- *Global Routing*: The purpose of this process is to perform a pre-routing of the nets with the aim of minimizing overflow and wire length. Subsequently, this solution (guides) will be utilized by the detailed routing for the final connection of each net.
- *Detailed Routing*: Assigns routes to specific metal layers and routing tracks within the global routing resources.
- *Timing closure*: Optimizes circuit performance by specialized placement and routing techniques.

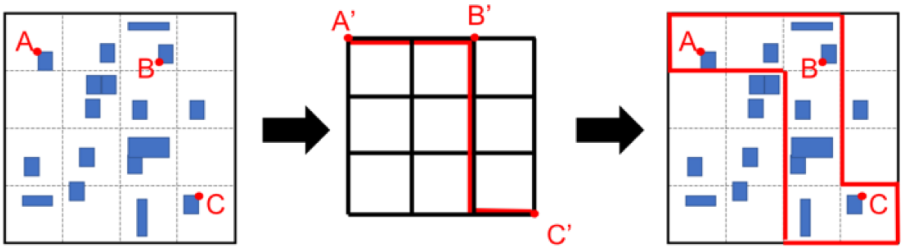
2.3 Global Routing

In the global routing phase, the primary objective is to perform a pre-routing of all nets, aiming to minimize issues such as overflow, excessive wirelength, and signal delay in the design. As the current IC industry continues to advance, the task of global routing becomes increasingly challenging. This difficulty is due to the fact that circuits are composed of an increasing number of nets which must be routed in increasingly smaller areas, needing proper management of available resources. In this context, it becomes crucial to address routing strategically, aiming to minimize the wirelength of each net and avoiding potential congestions or overflows in the process.
The quality of global routing solution directly affects chip area, speed, manufacturability,

power consumption and the number of iterations required to complete the design cycle. Hence this step plays an important role in determining circuit performance (TANG et al., 2020).

Figure 2.2 shows a net composed of three pins (A, B and C). The final routing of the net must connect these pins. To achieve this, the routing area is divided into cells, where each cell is known as a gcell. After identifying which cells pins A, B and C are in, we seek to find a set of gcells that connect those cells that contain the pins. These final routes, also referred to as guides, play a crucial role as they are utilized by detailed routing to complete the precise definition of paths intended to connect various nets within the integrated circuit design. This comprehensive process aims to ensure efficient connectivity, minimizing potential interferences, and optimizing the overall performance of the circuit. Some of the algorithms currently used to address the global routing problem include FastRoute(PAN et al., 2012), which initially divides the problem by routing in a 2D space and then transforms it into three-dimensional routing. Another algorithm is CUGR(LIU et al., 2020), which performs routing directly in 3D space using a 3D maze router. Finally, a parallelization approach is employed in SPRoute2.0 (HE et al., 2022) and FastGR (LIU et al., 2023) to effectively solve the global routing problem, resulting in improved execution time compared to other algorithms.

Figure 2.2: Representation of a grid and the gcells in global routing (HE et al., 2019). The routing area is divided into a cells where each cell is commonly called a gcell, then the global routing is constructed by a sequence of gcells which connect the pins of a net.



2.4 Machine Learning

Machine Learning (ML) is an area of artificial intelligence which aims to develop algorithms and models that can learn patterns and make decisions based on information without human intervention. Within artificial intelligence (AI), machine learning has emerged as the method of choice for developing practical software for computer vi-

sion, speech recognition, natural language processing, robot control, and other applications.(JORDAN1; MITCHELL, 2015)

Machine Learning algorithms use information and experience to improve your performance on a specific task. The learning process can be defined as a training process whose objective is to improve performance when performing a given task. For example, to learn to recognize images of dogs and cats, the task will be to label an image as "dog" or "cat." The accuracy of the classifier in assigning the labels appropriately will define the performance of the algorithm. Training involves using a collection of images labeled "dog" if it is a dog and "cat" if it is a cat. During this process, the algorithm gains experience and learns patterns that it then uses to make decisions with new information.

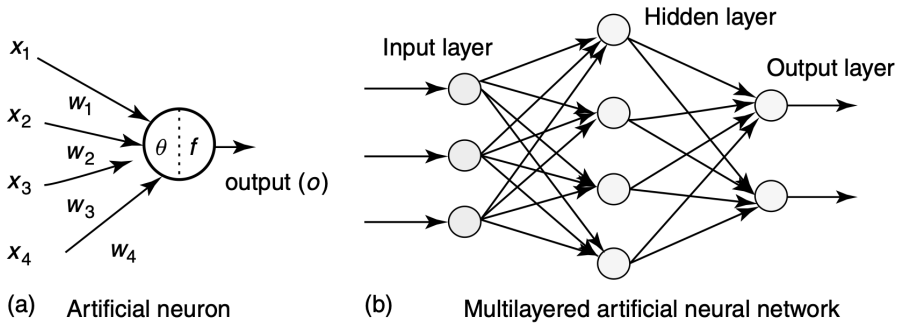
Conceptually, machine-learning algorithms can be viewed as searching through a large space of candidate programs, guided by training experience, to find a program that optimizes the performance metric. (JORDAN1; MITCHELL, 2015)

2.5 Artificial Neural Network

Artificial neural networks (ANN) have been developed as generalizations of mathematical models of biological nervous systems (ABRAHAM, 2005). It is designed to perform specific tasks by simulating the interconnections of artificial neurons. These artificial neurons work together to learn patterns of information, continually modifying the weights of their connections through an iterative process. This iterative adjustment of connections is known as training, which aims to improve the performance of the ANN in various applications, such as pattern recognition, classification of information, prediction of new information from known data, among others.

The Figure 2.3 shows the architecture of an artificial neuron and also the architecture of an ANN. An artificial neuron is composed of input values, to which a weight is multiplied and then added in a function called activation function. The output of the artificial neuron is calculated in the simplest case by comparing the result of the activation function with a threshold (θ). An ANN is composed of multiple layers, which include the input layer, output layer, and hidden layers. Each of these layers is composed of interconnected artificial neurons, and it is through these connections that the ANN learns and processes information. The input layer receives the initial information, the hidden layers process the information through operations, and the output layer produces the final result of the ANN.

Figure 2.3: Architecture of an artificial neuron and a multilayered artificial neural network (ABRAHAM, 2005). A multilayered artificial neural network (b) is composed of an input layer that receives the information, hidden layers that will perform the operations and an output layer that will produce the final result. Each hidden layer is composed of a set of artificial neurons (a) which will receive the values and calculate an output using an activation function.



2.6 Genetic Algorithms

Genetic algorithms (GA) are search algorithms based on the principles of natural selection and genetics, introduced by J Holland in the 1970's and inspired by the biological evolution of living beings (HALDURAI; MADHUBALA; RAJALAKSHMI, 2016). The GA are commonly applied to optimization and search problems due to their ability to consistently generate improved solutions in each iteration. These algorithms draw inspiration from biological genetic processes, utilizing operators such as selection, recombination, and mutation to evolve and progressively refine a population of potential solutions over multiple generations. This iterative and evolutionary approach provides a distinctive advantage by ensuring that over time, the GA converges towards more optimal solutions, making it a valuable tool in the effective resolution of complex optimization and search problems.

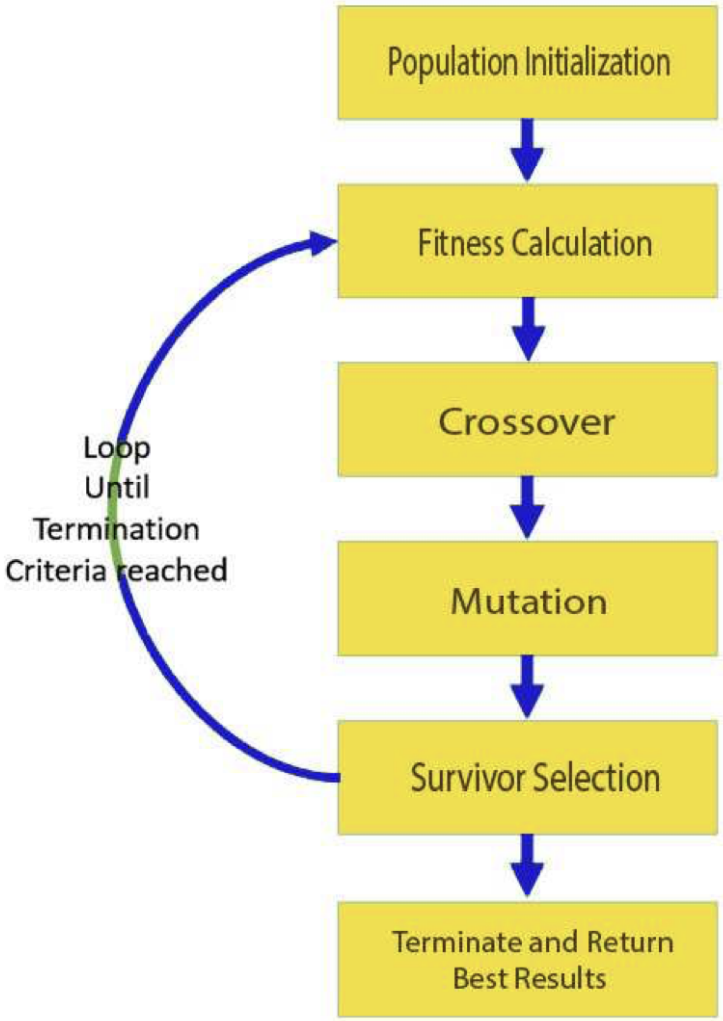
The flow of a genetic algorithm is defined by the following steps:

- *Initialization (first generation)*: Creation of an initial population of potential problem solutions (individuals). This process is mostly done randomly.
- *Fitness Evaluation*: Each individual is assessed based on its fitness to solve the problem. Fitness evaluation involves using a function that quantifies how good a solution is in terms of desirable result.
- *Selection*: Individuals with higher fitness are more likely to be selected as parents to create the next generation.
- *Crossover*: Selected individuals are combined with each other to create the next

generation. This operation is performed using techniques of genetic crossover or recombination. The new generation inherits characteristics from the parents.

- *Mutation*: In certain iterations, a mutation is introduced to the offspring to add variability to the population.
- *Replacement*: The new generation replaces the less fit individuals, ensuring an improvement in the overall quality of the population.

Figure 2.4: Flow of GA algorithm (HALDURAI; MADHUBALA; RAJALAKSHMI, 2016)



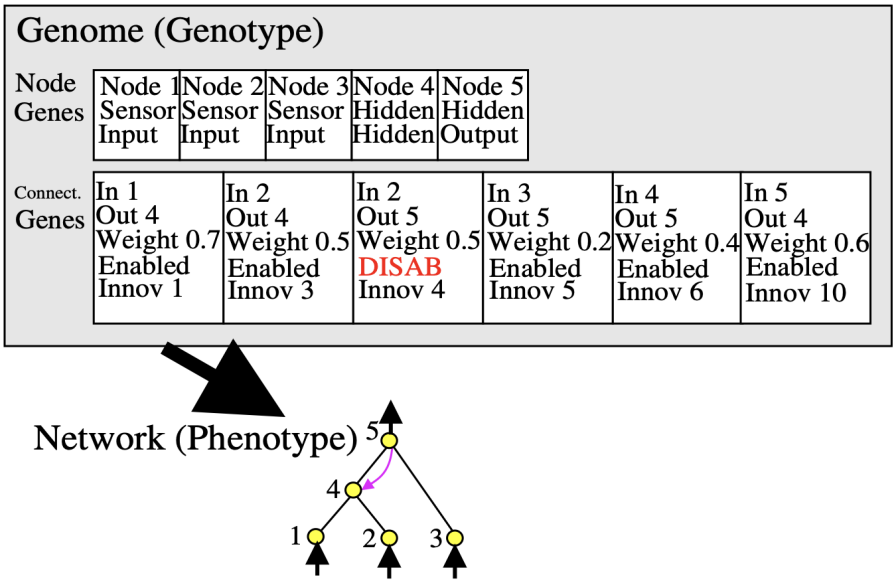
These steps will be repeated over several generations, allowing the population to evolve and progressively improve over time, as illustrated in the Figure 2.4. At the end of the process, the algorithm is expected to converge toward a solution that approaches the optimum for the given problem.

2.7 Neuroevolution of augmenting topologies algorithm (NEAT)

NEAT (STANLEY; MIIKKULAINEN, 2002) is a genetic algorithm which encodes a neural network as an individual (genome). This allows neural networks to learn from a reward function, instead of back propagation, avoiding the need for large data sets for training. In each new generation the algorithm selects the neural networks with the highest fitness, which will be used to generate the new population, the less fit neural networks will be eliminated from the population. There is a probability that a neural network will suffer a mutation, which consists of modifying its structure by adding or eliminating nodes and connections.

Figure 2.5 shows the structure of a genome, it includes a list of nodes which stores the total number of nodes that the neural network has and a list of all connections of the neural network. Each connection will store the index of the two nodes it connects, the weight of the connection, a flag to indicate whether the connection is enabled or not and an innovation number which will be useful during the crossover operation.

Figure 2.5: Structure of a genome (STANLEY; MIIKKULAINEN, 2002). The genome is composed of a list of the nodes and connections of the neural network.

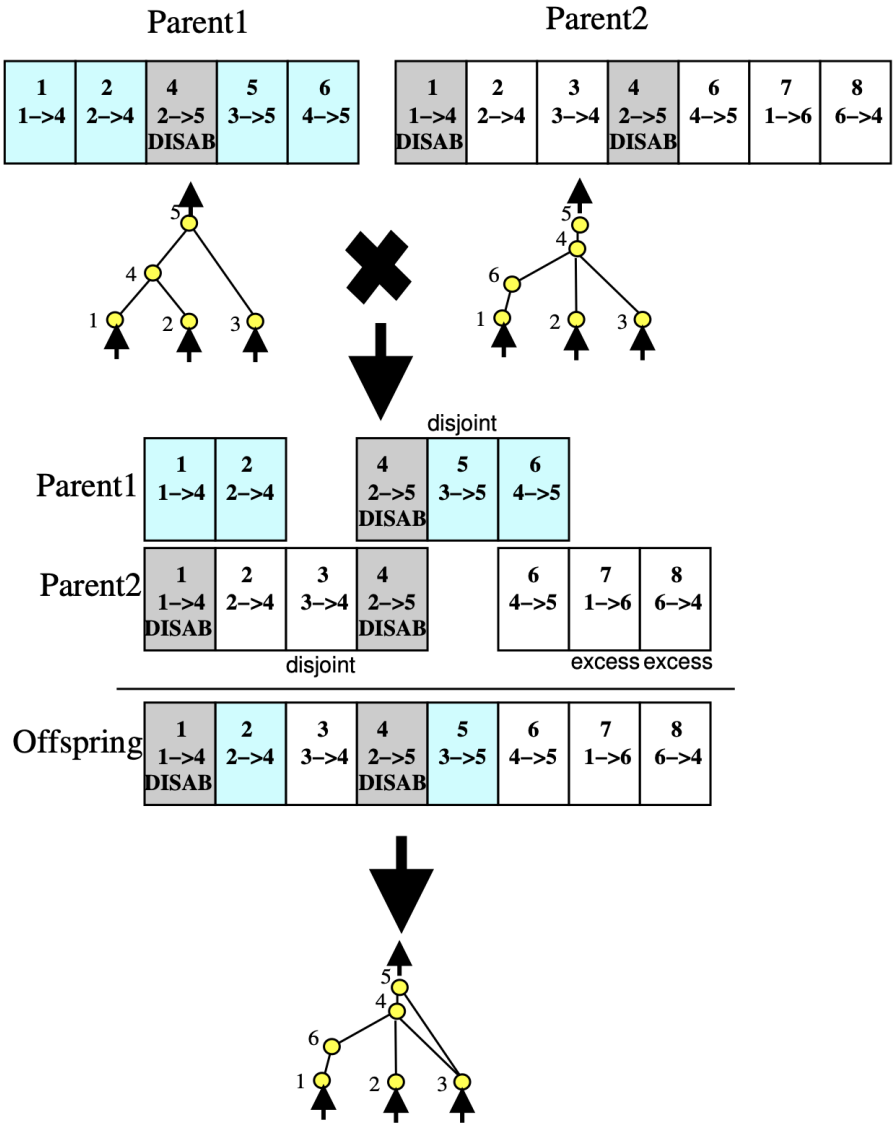


The operations of the NEAT algorithm are explained below:

- *Crossover*: In order to carry out the crossover operation, each connection of the neural network stores an innovation number which indicates at what time this connection was created. Then, the crossover between two neural networks will be carried out by joining the connections with the same innovation number. If a con-

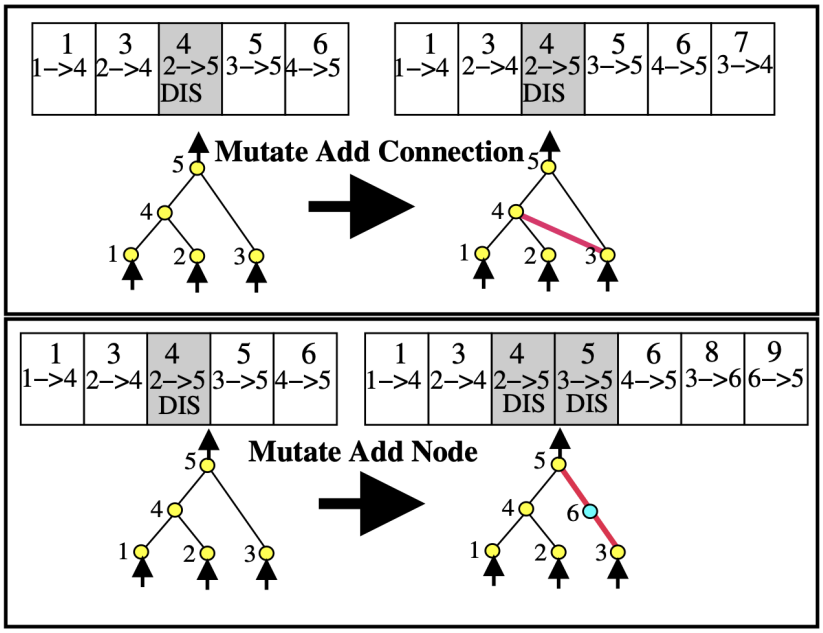
nection does not exist in the other neural network, it will be inherited, if it belongs to the parent with more fitness. (Fig.2.6)

Figure 2.6: Crossover operation between two NEAT genomes (STANLEY; MIIKKULAINEN, 2002)



- *Mutation*: It can change both connections weight and neural network structure. Connection weight mutation in which there is a probability of changing the value of an existing connection between two nodes. Structural mutation, which can be done in two ways, the first is to create a new connection between two disconnected nodes and the second is to create a new node which is done by disabling an existing connection between two connected nodes to connect them to the new node creating two new connections. Both types of mutation are shown in the Figure 2.7.
- *Speciation method*: That helps protect the topology of the new generations by dividing them into groups with similar topologies and making them compete with

Figure 2.7: Mutation operation on NEAT algorithm (STANLEY; MIIKKULAINEN, 2002)



each other, in this way the generated topologies have the opportunity to innovate before competing with individuals from other groups.

The NEAT algorithm has proven to be a useful one for the optimization of neural networks; its applications are found in fields such as robotics, video games and the optimization of control systems. In (WILLIGEN; HAASDIJK; KESTER, 2013), NEAT has been applied to optimize the control of autonomous vehicles, improving adaptive decision making according to the specific conditions of each scenario. In (PATRASCU; IANCU, 2023)(PEREZ-LIEBANA; ALAM; GAINA, 2020)(WITTKAMP; BARONE; HINGSTON, 2008), NEAT is applied to decision making in well-known video games, demonstrating its ability to dynamically adapt to different strategies according to environmental conditions. Finally, in (WEN et al., 2017) the NEAT algorithm is applied to train neurocontrollers capable of moving a robotic arm towards a nearby target position.

3 RELATED WORK

In this chapter, we provide a review of related work in VLSI focused on the global routing stage. First, we present a brief summary of some work developed by colleagues in the research laboratory. Then, we review existing works in the state of the art that aim to solve the global routing problem. Finally, we will provide an analysis of works that employ methods, such as NEAT or Reinforcement Learning, with the goal of addressing the global routing problem.

3.1 EDA algorithms developed in the research laboratory

In (REIMANN, 2013), the author presents a global routing implementation for integrated circuits capable of addressing the problems proposed in the ISPD 2007 and 2008 competitions. His algorithm uses the rip-up and re-route technique together with monotone and maze routing approaches. He have also implemented a new method that orders the nets during the rip-up and re-route phase. To compare the results, he used a version of his algorithm that prioritizes routing quality, combined with Minimum Steiner Trees (MST) for the construction of the routing trees. The results obtained show that the algorithm experiences a slight increase in the total wirelength and in the runtime in more complex designs that require solutions without violations. Furthermore, he explain that a large part of their results is due to the importance of having an identifier of congestion regions, which helps avoid unnecessary cable detours and accelerates the convergence of the algorithm.

In (NUNES, 2013), the author presents techniques applied in the GR-WL algorithm (REIMANN, 2013) for the delimitation and treatment of areas with high demand for resources in global routing. These techniques are based on first identifying areas with high demand for connections and then protecting them during the routing phase to avoid congestion. He also describes parameters for applying cost pre-increase in areas with high demand for connections. In the results, it is shown that these techniques reduce the total congestion compared to the original implementation of the algorithm. Furthermore, it is observed that congested area delimitation techniques decrease the execution time of global routing. However, a disadvantage of these techniques is the increase in the total cable length in some experiments due to the dispersion of connections, which allows congestion reduction.

In (MONTEIRO, 2023), the author carries out a literature review on global routing, explores its application in different stages of the physical project flow of a circuit and finally presents an adaptation of the implementation of the FastRoute algorithm (PAN et al., 2012) to carry out experiments in a real flow. These adaptations include support for modern input and output file, as well as a new representation of routing resources. Additionally, a method to correct antenna violations is presented. The results demonstrate the importance of these adaptations in the FastRouter algorithm to generate routes that comply with the project rules, are congestion-free, have a minimum cable length and require a minimum number of vias.

In (TUMELERO, 2015), the author first studies how sequential global routing works, what steps it is composed of, and what its limitations are with the provided benchmarks. Then, he implements a version of the parallelized routing algorithm, which is based on dividing the total number of networks according to their wirelength into small groups that will be routed independently and simultaneously. To achieve this, he proposes a data dependency detector that will allow him to identify the routing areas that are being modified by two tasks in parallel. He demonstrates that his method, together with specialized parallelization hardware, has the potential to reduce the execution time of global routing.

In (OLIVEIRA, 2021), the author presents a comprehensive study on the detailed routing problem in VLSI, analyzing in detail the state of the art of algorithms that address the problem both sequentially and parallel. Subsequently, he proposes a parallel detailed routing flow for VLSI that could serve as a valuable reference for future implementations of more algorithms in this field.

In (PLACIDO, 2016), the author investigates the global placement of cells in integrated circuits, followed by the implementation of a global placement tool based on a quadratic analytical algorithm. Although the execution time of the proposed algorithm was not significantly high and managed to distribute the cells in all the circuits of the test set, the wirelength (calculated by HPWL) obtained was higher than that of other algorithms such as FastPlace and SimPL. The conclusions indicated that these algorithms, in addition to performing the task of global positioning of cells, are specialized in directly optimizing the HPWL.

Other works related to global positioning, such as (HENTSCHKE, 2002), (FOGACA, 2016) and (FOGACA, 2020), focus on analyzing existing algorithms and, in some cases, implementing improvements to increase the quality of the obtained results.

3.2 Existing methods to solve the Global routing problem

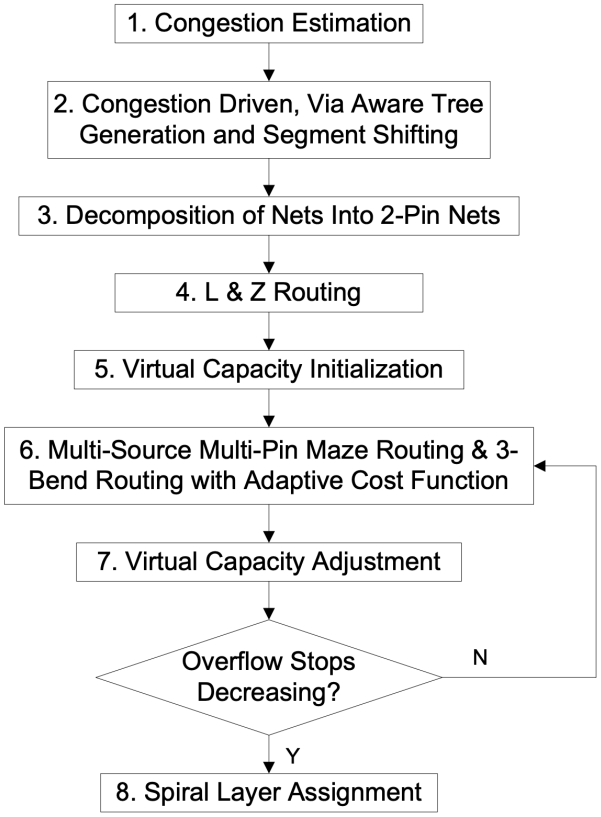
The BoxRouter2.0 (CHO et al., 2007) proposes optimizing net routing in integrated circuit designs, with the aim of minimizing both the wirelength and the number of necessary paths. Initially, routing is addressed in a two-dimensional (2D), using a specialized A* algorithm that finds efficient routes and minimizes wirelength. Subsequently, the conversion to three-dimensional (3D) routing is performed using a layer assignment method based on Integer Linear Programming (ILP).To validate the effectiveness of the method, exhaustive experiments are performed using the ISPD 1998 (ALPERT, 1998) standard benchmarks. The results show that the proposed method achieves congestion-free routing with a minimum wirelength.

An improved version of the original NTHU-Route (GAO; WU; WANG, 2008) algorithm is presented in (CHANG; LEE; WANG, 2008). The difference between this improved version and the original algorithm is clearly reflected in terms of routing solution quality and execution time. The authors have achieved this advancement by introducing innovative features, such as a new history-based cost function. This feature effectively helps identify highly congested edges during the routing process, allowing them to be rerouted more efficiently during the ripup and reroute phases. In addition, a net ordering method has been implemented that contributes significantly to the identification and resolution of congested areas, thus improving the overall effectiveness of the algorithm. To validate the effectiveness of these improvements, the authors carried out experiments using the ISPD98 (ALPERT, 1998) and ISPD07 (NAM et al., 2007) benchmarks. The results obtained clearly show that the proposed method outperforms the original algorithm in terms of overflow, wirelength and execution time, thus demonstrating its effectiveness and relevance in the context of global routing optimization in VLSI.

In FastRouter (PAN et al., 2012), the authors proposed an algorithm that addresses global routing by initially performing 2D routing and then extending it to 3D routing through the layer assignment algorithm. FastRouter is a Global Routing algorithm that decomposes the problem into several stages, Figure 3.1 shows the complete flow of the algorithm. First, a construction of congestion-driven and via-sensitive Steiner topologies is performed for each net in the design, followed by segment shifting techniques. Then, the tree structures are decomposed into 2-pin nets and pattern routing is executed using L and Z shapes to perform the first routing. Subsequently, the virtual capacity of the design is initiated based on the first routing. This will be useful to guide the iterative stage of rip-up

and reroute and address the congestion problem. During the rip-up and reroute stage, they used two techniques: 3-bend routing and multi-source multi-sink maze routing, with the aim of avoiding the generation of congestion and minimizing the use of vias. Finally, after obtaining a 2D routing, they converted it to a 3D routing using the spiral layer assignment algorithm.

Figure 3.1: FastRouter algorithm flow (PAN et al., 2012)

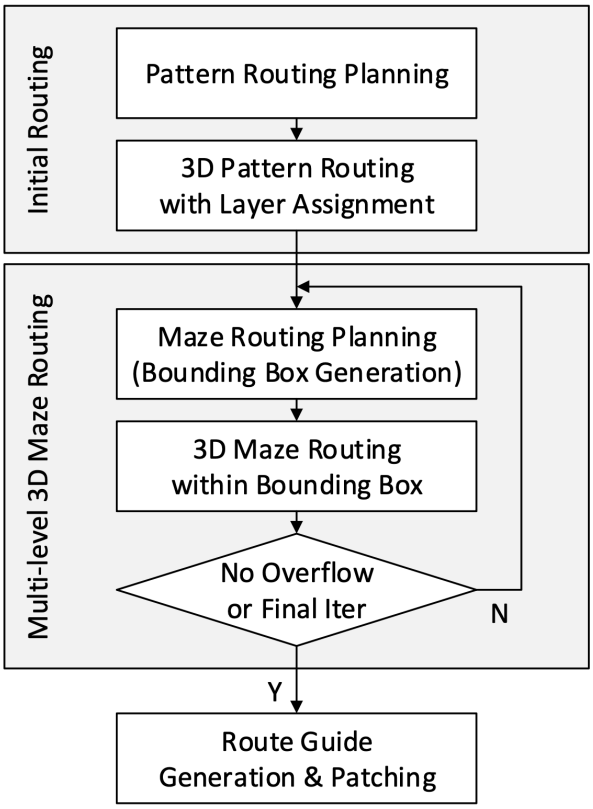


To carry out their experiments, they used the benchmarks from the ISPD08 contest (NAM; SZE; YILDIZ, 2008). The implementation of the algorithm was carried out in the C programming language, using the FLUTE library (CHU; WONG, 2007) to generate the Rectilinear Steiner Minimum Tree (RSMT). FastRouter managed to reduce the wirelength and runtime compared to the algorithms classified in the top positions of the ISPD08 contest. They considered the possibility of improving the maze routing stage to achieve a balance between reducing congestion and maintaining a small wirelength.

In CUGR (LIU et al., 2020) is an algorithm that solves the global routing problem directly in a 3D area. To achieve this, the authors proposed an algorithm that decomposes the routing problem into three phases: initial routing, 3D maze routing, and generation of route guides. Figure 3.2 shows the algorithm flow in the first stage, the algorithm uses

3D pattern routing to make an initial connection. Decompose each multi-pin net into a set of 2-pin nets using FLUTE (CHU; WONG, 2007) to generate the Rectilinear Steiner Minimum Tree (RSMT), and then employ a dynamic programming algorithm to perform pattern routing and layer assignment simultaneously. Nets with violations then go through multiple iterations of rip-up and reroute using a 3D maze router. The last step involves the generation of guides and insertion of patches. Patches are additional guide regions that help resolve some specific conditions.

Figure 3.2: CUGR algorithm flow (LIU et al., 2020)



They performed their experiments using the ICCAD19 contest (SCHLICHTMANN et al., 2019) benchmarks, using Dr. CU2.0 (LI et al., 2019) to generate the detailed routing solution. The implementation was carried out in the C++ programming language, using the Geometry Boost and Rsyn (FLACHA et al., 2017) libraries to carry out the parsing of the LEF/DEF files. They compared the results obtained with the two algorithms that occupied the first places in the ICCAD19 contest. They managed to match the wirelength results and the via score of these algorithms, considerably improving the final runtime. CUGR uses sophisticated techniques that contribute to the optimization of wirelength, number of vias and congestion. With its 3D pattern routing technique, the algorithm is

capable of routing most nets quickly and efficiently. They demonstrated that their global router competes with the top participants of the ICCAD19 contest in terms of performance and runtime.

In NCTU-GR 2.0 (LIU et al., 2013), a heuristic maze-type routing algorithm is proposed that aims to improve the wirelength estimation, reduce the execution time and minimize the final wirelength. NCTU-GR 2.0 uses Rectilinear Steiner Minimum Tree (RSMT) to generate routing trees containing paths with short wirelengths. Additionally, a multi-threaded version of the same collision-aware global routing algorithm is presented. In the obtained results, it is observed that the algorithm manages to reduce both the wirelength and the execution time compared to other algorithms.

3.3 Methods that apply Maching Learning algorithms to the Global Routing problem

In (RINNARV; BRINK, 2017), the author applied the NEAT algorithm to address the problem of nets routing in integrated circuits. In his research he adapted the NEAT algorithm to route 2-pin nets and compared his obtained results with industry standard algorithms. To carry out his experiments, he generated two-dimensional circuits represented by two-dimensional grids, where each position has a value of 0 if it is not routable and 1 if it is routable. The input values for the NEAT neural networks include all the values of the two-dimensional grid ($N \times N$), in addition to the position of each agent and the position of its destination point. The output nodes are defined in an array that represents the preferred movements that a net can make during its routing. In its evaluation method, he defined the fitness function to reduce the wirelength and the number of vias. He used the same function to compare his results with the results of the standard A* and BFS algorithms.

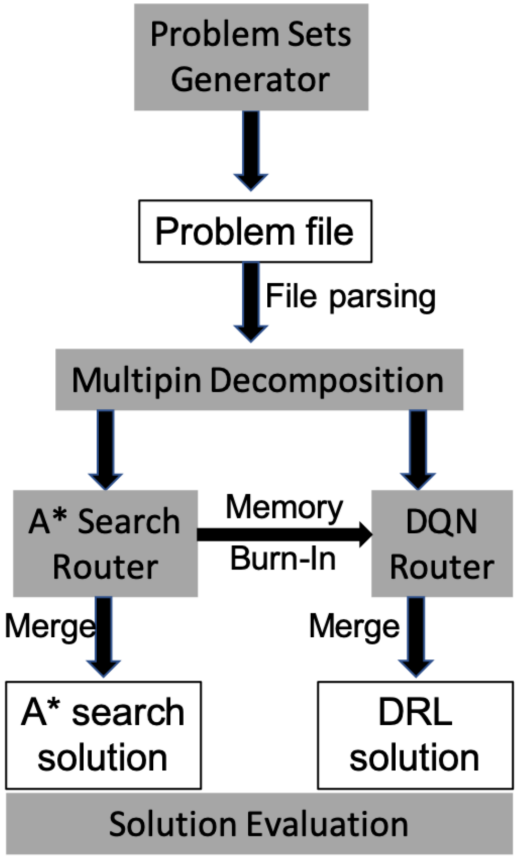
The conclusion reached by the author is that the routings generated by the NEAT algorithm do not compete with the industry standard algorithms A* and BFS since in some test cases the final routing ends with disconnected nets or with detours that impact the the wirelength of the design. He also highlights that a better definition of the fitness function could equal or even improve net routings, since neural networks are able to take into account more parameters compared to standard algorithms.

In (LIAO et al., 2019), the authors applied Deep Reinforcement Learning (DRL) to ad-

dress the global nets routing problem in integrated circuits. They used a Deep Q-network (DQN) as their main Reinforcement Learning (RL) algorithm to route the nets simultaneously. They defined 12 variables as input to the Deep Q-network, the first 3 representing the current position of the agent in x, y and z coordinates. The next 3 variables reflect the distance from the current position to the target in the x, y and z directions. The latter variables encode the capability information of all edges that the agent can traverse in its next step. The output of the network is encoded into 6 values, which represent the preferred direction for next movement during routing of a net. The algorithm and evaluation flow are shown in Figure 3.3.

For the experiments, they generated test cases with various characteristics, which were routed with both their proposed RL algorithm and the standard A* algorithm. The results of both algorithms were evaluated based on the total congestion and the total wirelength of the routing. They concluded that the results produced by the DQN algorithm are superior to those obtained by the A* router, highlighting the importance of fine tuning the variables of the DQN algorithm.

Figure 3.3: Pipeline for solving global routing with DQN (LIAO et al., 2019)



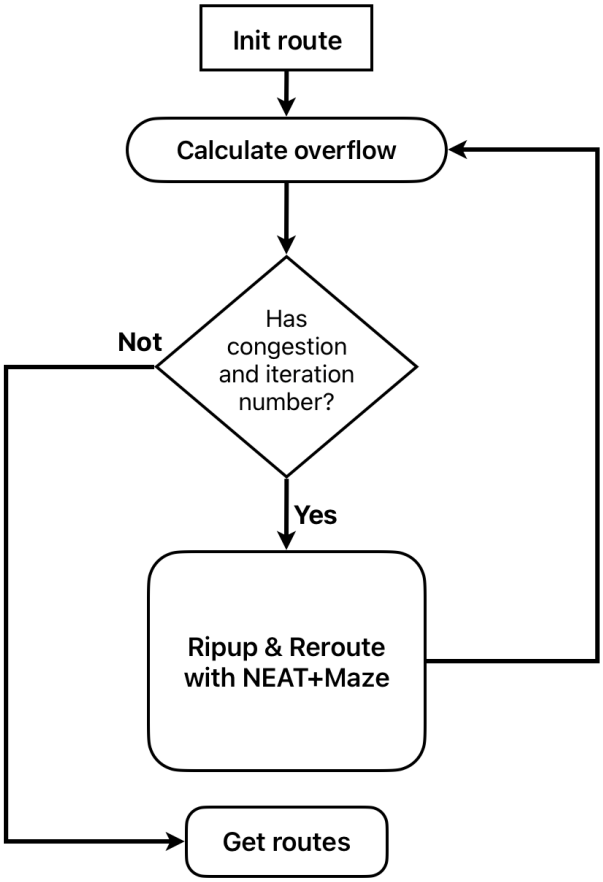
In this chapter, the works presented provide an understanding of the state of the

art in the task of global routing of integrated circuits. The review of these works not only shows the opportunity to improve current methods, but also highlights the need to continue innovating with new techniques to optimize the task of global routing in complex circuits. In the next chapter we will present our proposed method, which seeks to improve MazeRouter, an algorithm widely used in current literature for global routing of integrated circuits.

4 NEAT TO GLOBAL ROUTING

In this chapter, we introduce an innovative algorithm specifically designed for 2D routing 2-pin nets NEATRouter. This algorithm incorporates the Ripup and Reroute method, in conjunction with the gcell history calculation method, to effectively address congestion issues. NEATRouter will run a specified number of iterations (below a predefined maximum) until successfully routing all nets without congesting any gcell within the design (grid). At the end of its execution, the algorithm will provide the complete routing of each of the nets in detail. This complete set of routing data is critical for further analysis and detailed evaluations of the quality and efficiency of the algorithm in different contexts and test scenarios. The Figure 4.1 shows the complete flow used by the NEATRouter algorithm.

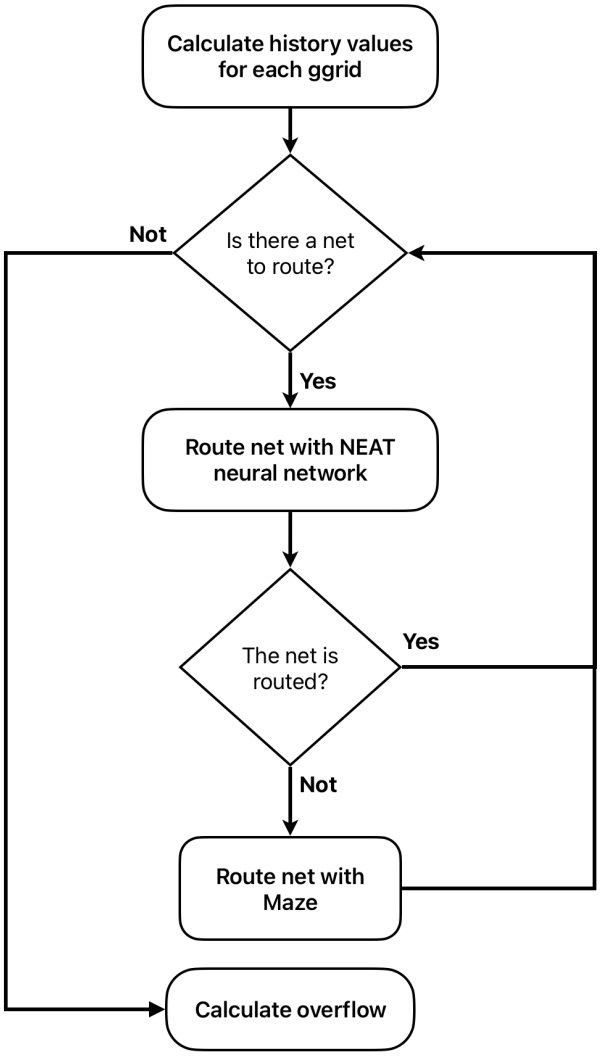
Figure 4.1: General Flow of NEATRouter



In the "Ripup and Reroute" phase, we will proceed to route each net individually. Initially, we will calculate the history of each gcell, introducing an additional cost to gcells that experienced congestion in the last routing. This is intended to encourage the exploration of alternative routes by nets, ensuring that only those nets that need it use

those specific gcells. Next, we will disconnect each net to reroute, thus freeing the used resources. The algorithm will route each net using first a NEAT neural network previously trained to handle 2-pin nets, obtaining the routing environment information as input data. In the case that the NEAT neural network is unable to route the net, the MazeRouter algorithm will be used to ensure valid routing. After all nets are routed, the algorithm will re-evaluate congestion across the entire grid. If any gcell shows congestion, the same steps will be repeated. The maximum number of iterations is set to 50, in case of not achieving congestion-free routing within this limit, the algorithm will provide congestion routing. The Figure 4.2 shows the complete flow of the Ripup and Reroute phase detailing each step performed.

Figure 4.2: Ripup and Reroute Flow



In the following sections we provide a more detailed explanation of each part of the algorithm as well as how the NEAT neural networks were trained.

4.1 Implementation of MazeRouter

We have implemented the MazeRouting algorithm (LEE, 1961), widely used for routing problems. In the implementation of the algorithm, we employ the A* algorithm to improve the execution time, adding a heuristic to the cost. To calculate the cost of using a gcell, we apply the equation 4.1.

$$Cost_{pos} = WL_{pos} + dist(pos, pos_{target}) + history_{pos} \tag{4.1}$$

where pos is the 2D position of the gcell to use, pos_{source} is the 2D position of the source of the net, pos_{target} is the 2D position of the destination, WL_{pos} is the current net wirelength, $history_{pos}$ is the history value of a gcell which will be 0 in the case of the gcell has not had congestion or otherwise $history_{pos} > 0.0$ and the function $dist(a, b)$ returns the Manhattan distance between 2D points a and b.

4.2 History calculation for a gcell

The usefulness of the history value of a gcell lies in transmitting information from iteration to iteration, adding a cost to gcells that experienced congestion in the last iteration. In this way, the algorithm will determine which gcell to avoid using, helping to eliminate congestion in the final routing. To calculate the history value of each gcell after routing all the nets, we use the equation 4.2 defined in (KAHNG et al., 2011).

$$history_{pos} = \frac{numberOfNets_{pos}}{capacity_{pos}} \tag{4.2}$$

where pos is the current 2D position of the gcell, $numberOfNets_{pos}$ is the number of nets which use the gcell in the pos for their routing and $capacity_{pos}$ is the total capacity of that gcell in the pos.

4.3 NEAT neural network

In the next section, we will detail how we implement the training and the generation of the NEAT neural networks, which are subsequently used to route the nets. For the implementation we use the NEAT-python library (MCINTYRE et al.,) which has the

NEAT algorithm implemented in the Python 3 programming language.

4.3.1 Initial Configuration

For the configuration of the NEAT algorithm, we use the ReLU function as the activation function for the nodes. We set a probability of 50% to add a new connection and 40% to add a new node. Additionally, nodes in the initial population will be disconnected, allowing NEAT neural networks to form their own structures from scratch.

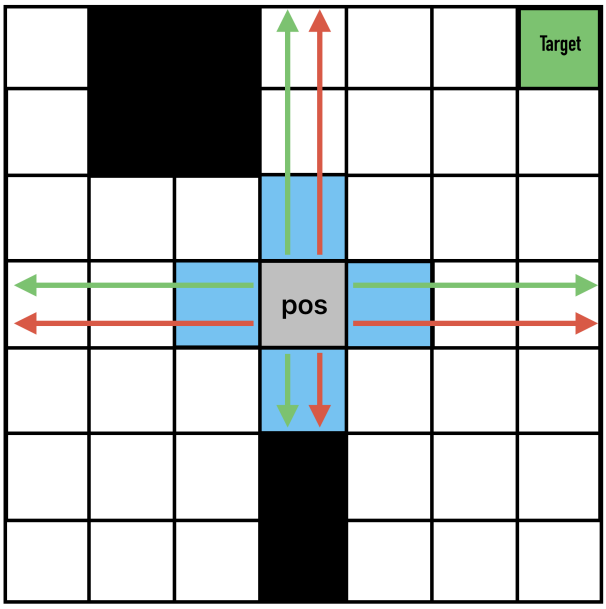
4.3.2 Input nodes

As for the values assigned to the input nodes, these are presented in a list that covers the minimum amount of resources available on the four sides (north, south, east and west) of the position. In addition, the measurement of the distance from the four neighbors of the gcell in the current position to the desired goal is incorporated. This measurement is complemented with the maximum history value between the aforementioned neighbors of the gcell at the current position. This inclusion of detailed information provides the algorithm with a comprehensive view of available resources and distance to the target, thus allowing more informed decisions in the routing process. In the figure 4.3, the gray colored gcell indicates the position of the source pin, while the green gcell represents the position of the target pin. The blue squares represent the neighbors of the current position, from which the minimum distance to the target will be calculated. Black gcells indicate areas with no resources available for routing. The green arrows represent the number of gcells in the 4 directions, from which the minimum resource value will be taken. On the other hand, the red arrows represent gcells in the 4 positions, from which the maximum history value will be taken. These values will serve as input data for the NEAT neural network.

4.3.3 Output nodes

During the routing process, output nodes play a critical role in generating a complete list of four values, each representing a possible movement, either north, south, east or west. The final choice of the next move is determined by identifying the maximum value

Figure 4.3: Representation of the input values for the NEAT neural network. The gray, green, blue, and black gcells represent, respectively, the source pin position, the destination pin position, the neighbors of the current position, and the gcells that have no available resources.The green arrows represent the number of gcells in the four directions, from which the minimum resource value will be taken, and the red arrows represent gcells in the four positions, from which the maximum historical value will be taken.



in this list. This strategic approach not only facilitates efficient evaluation, but also equips the algorithm with the ability to make informed decisions while navigating through the routing procedure. By assigning numerical values to each possible move, the algorithm can effectively weigh its options and select the most promising direction, contributing to a more dynamic and adaptive routing mechanism.

4.3.4 Fitness Function

The fitness function plays a crucial role in the evaluation of each individual, with the best of a generation being defined by the maximum fitness value achieved. To determine the fitness of a NEAT neural network, we carry out a comprehensive definition that involves the evaluation of specific rewards.

- The *connection reward* is used to score how close a net is to being connected, the Equation 4.3 will return the value of 1.0 when the net is routed and connected to its target and otherwise it will return the Manhattan distance to the target point divided

by the value of the maximum distance from the grid.

$$reward_{con} = 1.0 - \frac{dist(pos, pos_{target})}{height + width} \tag{4.3}$$

where $dist(a, b)$ returns the Manhattan distance between 2D points a and b, pos is the current 2D position of net, pos_{target} is the target 2D position of the net, $height$ is the height of the grid and $width$ is the width of the grid.

- The *wirelength reward* is used to score the final routing wirelength of a net, this value is calculated only if the net is connected to its target pin. The score is defined by the Equation 4.4, this will return a direct proportion between the Half-Perimeter Wirelength and the wirelength of a net. Routes with the lowest wirelength will be evaluated with values close to 1.0, while those with the highest wirelength will be close to 0.0.

$$reward_{WL} = \frac{HPWL_{net}}{WL_{net}} \tag{4.4}$$

where $HPWL_{net}$ is the Half-perimeter wirelength of the net and WL_{net} is the wirelength of the net route.

- The *history reward* is used to score the total amount of history used to route a net, this reward will only be calculated after the net is connected to its target pin. The score is defined in the Equation 4.5, it will return a value between 0.0 and 1.0 which is the direct proportion between the sum of the history used by the net routing and the total sum of the history in the grid.

$$reward_{h} = \frac{usedHistory_{net}}{totalHistory} \tag{4.5}$$

where $usedHistory_{net}$ is the sum of the history used by the route of the net and $totalHistory$ is the total sum of the history in the grid.

Finally, when the routing fails to connect the net to its pin node, the total fitness of the individual will be calculated using equation 4.6.

$$Fitness_{total} = HW \times (\theta \times reward_{con}) \tag{4.6}$$

If, on the other hand, the final routing manages to connect the net to its target pin, then the fitness of the individual who created the routing will be calculated using equation 4.7.

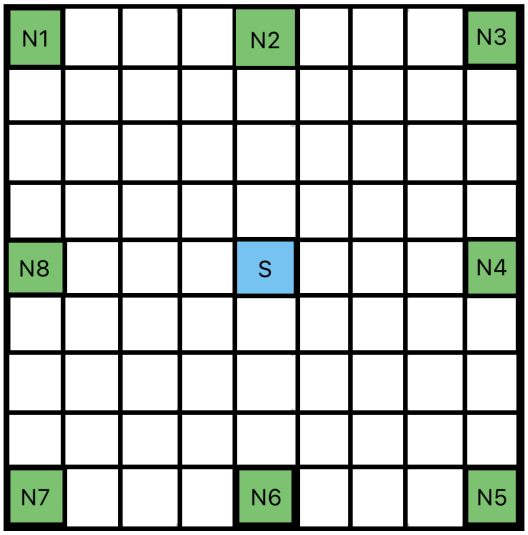
$$Fitness_{total} = HW \times (\theta \times reward_{con} + \alpha \times reward_{WL} + \beta \times reward_{h}) \quad (4.7)$$

where HW is defined as $height \times width$ and the values of θ, α, and β are coefficients that help balance the training of the NEAT neural network.
This comprehensive approach not only allows for a more precise assessment of the quality of the neural network, but also favors the identification of the individual best suited to advance to the next generation in the evolutionary process.

4.3.5 Trainning method

We developed the training method for the NEAT neural networks using eight nets as input data. These nets were placed on a grid of size 9×9, and the history values of each gcell were filled with random decimal numbers in the range of 0.0 to 10.0. Although they share the same starting position, the nets have varied objectives. The figure 4.4 shows the positions of the eight nets used in training, strategically selected to guarantee the generation of a NEAT neural network prepared for a wide diversity of cases. The final

Figure 4.4: Nets used for training NEAT neural network. All nets have their source pin located in the blue gcell, while their target pins are distributed in various directions to train the NEAT neural network for a wide diversity of scenarios.



fitness of a NEAT neural network is calculated as the sum of the fitnesses when routing the eight nets. The values of the variables for the fitness function were defined as follows:

1.6 for θ, 0.7 for α and 0.7 for β. With this, the NEAT neural networks give priority to connecting all the pins (θ) while reducing the wirelength (α) and congestion (β) in a balanced manner. Finally, the individual with the best fitness after 1000 generations, that is, the one who most efficiently routes the eight nets, will be selected to route other designs.

4.4 Routing with NEAT neural network

At the end of the training phase, a NEAT neural network will be obtained, the one that has achieved the best fitness. This NEAT neural network will be used in the Algorithm 1 to route the nets in the test cases. At the beginning, an agent is created (line 3), which will be the intermediary between the environment (routing area) and the NEAT neural network. During the routing process, the agent will collect the information from the environment and add it to the NEAT neural network (line 5). With these data the next movement will be calculated (line 6) and the agent will update the current position (line 7). This process will be repeated until the target position is reached or there are no longer resources to perform the routing. If the target position is reached, then the agent will calculate the path found for that net (line 9) which will be returned by the function (line 12).

Algorithm 1: Route with NEAT neural netowrk

Input: Pin position pos, resources $rscs$ and history $hist$ information
Output: Net route $route$

```
1  function routeNEAT (pos, rscs, hist)
2      neatNN ← createNEATNN()
3      agent ← createAgent(pos, rscs, hist)
4      while agent.isActive() do
5          inputNEAT ← agent.getNEATInput()
6          nextMove ← neatNN.getNextMove(inputNEAT)
7          agent.updatePosition(nextMove)
8      if agent.isRoute() then
9          route ← agent.getRoute()
10     else
11         route ← [∅]
12     return route;
13 end function
```

5 EXPERIMENTS AND RESULTS

In this chapter, we will first explain the data set used to carry out the experiments, as well as the metrics used to make the comparisons. Then, we will provide a comprehensive overview of the results obtained in our experiments, focusing on comparing them with the Mazerouter algorithm, since this is used in most global routing algorithms.

5.1 Dataset

To test the algorithm, we generate test cases randomly. We try to ensure that routing each test with MazeRouter requires mandatory use of iterations to remove congestion. Detailed information about these cases in the table 5.1. The Grid size refers to the height and width of the routing area, the Net count is the number of 2-pin nets that the test case has and Layer count and Layer capacity is the number of available layers and the capacity of each layer respectively.

This decision to use random test cases serves a dual purpose in our evaluation. Firstly, it allows us to observe how the algorithm performs across diverse scenarios and under various testing conditions. By introducing randomness, we can assess the algorithm's adaptability and effectiveness in handling different inputs.

Secondly, random test cases enable a more comprehensive evaluation of the algorithm's generalizability. Exposing it to unpredictable scenarios provides a realistic representation of its real-world applicability, ensuring competence in a broader spectrum of potential use cases.

5.2 Experimental Results

To evaluate the efficiency of our method, we perform a comparison by routing the test cases using exclusively the MazeRouter algorithm and then employing NEATRouter, which leverages pre-trained NEAT neural networks to facilitate convergence toward more effective routing in fewer iterations. In summary, we want to see if the NEAT neural network helps make routing more efficient and faster compared to just using the MazeRouter. During the training of the NEAT neural network, 1000 generations were carried out, and the best individuals from the 100th, 250th, 500th and 1000th generations were selected.

Table 5.1: Characteristics of the test cases generated for the experiments. Test case is the name of the test, Grid size is the size $N \times N$ of the routing area, Net number is the number of nets that the test has, Layer number and Layer capacity is the number of available layers and the capacity of each layer respectively.

Test case	Grid size	Net count	Layer count	Layer capacity
Test 1	10	200	5	10
Test 2	10	200	5	10
Test 3	10	200	5	10
Test 4	10	200	5	10
Test 5	10	200	5	10
Test 6	10	500	5	20
Test 7	10	500	5	20
Test 8	10	500	5	20
Test 9	10	500	5	20
Test 10	10	500	5	20
Test 11	10	1000	5	40

5.3 Performance Evaluation

In the process of evaluating and comparing results, we have selected the final routing wirelength and total design congestion as our primary metrics. In addition to these, we have also considered secondary metrics, such as the comparison of runtime between the two approaches and the number of iterations needed to eliminate congestion. These metrics offer us a comprehensive and detailed view of the efficiency and performance of both approaches, allowing us to carry out a comprehensive evaluation of their strengths and limitations in different aspects of the routing process.

5.4 Results

The table 5.2 addresses things like the wirelength (WL), number of iterations necessary to remove the overflow (#iter), the average percentage of resources used to route all nets (Avg. used) and how long each algorithm takes to run (runtime). These numbers give us a good idea of how both approaches perform.

By taking a closer look at these numbers, we sought to draw crucial conclusions about the performance of each approach. The results in bold indicate the best values of the metrics for each testcase. This will allow us to better understand the positive aspects and areas of our method that could be improved, being essential for our final conclusions in the research.

Table 5.2: Comparison of the results using MazeRouter algorithm and NEATRouter. The WL is the wirelength used to route all nets, #iter is the number of iterations that the algorithm performed to remove congestion, Avg. used is the average used resources of a ggrid and runtime is the time that takes the algorithm to do the total routing.

	MazeRouter				**NEATRouter**			
Tests	**# iter**	**WL**	**Avg. used**	**Runtime(s)**	**# iter**	**WL**	**Avg. used**	**Runtime(s)**
Test 1	19	1519	57.4	**0.88**	19	1519	57.4	9.595
Test 2	5	1581	**58.9**	**0.428**	**3**	1581	61.5	2.067
Test 3	5	**1615**	61.7	**0.444**	**1**	1617	**59.7**	0.995
Test 4	2	1551	56.5	**0.312**	**1**	1551	**55.3**	0.523
Test 5	34	1666	61.4	1.574	**0**	**1588**	**56.2**	**0.559**
Test 6	**8**	3771	61.9	**1.035**	11	3771	**59.6**	8.197
Test 7	8	**3764**	60.0	**1.01**	8	3766	**59.8**	14.343
Test 8	3	3886	**61.7**	**0.56**	**2**	3886	61.8	1.67
Test 9	12	3756	62.8	**1.342**	**10**	3756	**62.4**	17.568
Test 10	3	**3879**	60.3	**0.538**	**0**	3881	**57.4**	1.776
Test 11	3	7730	62.5	**0.901**	**1**	7730	**59.9**	2.241

5.5 Discussion

The results obtained indicate that our proposed algorithm can match or improve the results of the MazeRouter algorithm. The NEATRouter algorithm tends to provide results in a smaller number of iterations, achieving a decrease in both congested areas and the total length of cables in routing.

In order to gain a deeper understanding, we selected three specific tests to closely examine their results and deepen our understanding of the algorithm's behavior. By focusing on these particular cases, we seek to obtain a more complete view of how our algorithm performs in various situations. This approach allows us to analyze the strengths, weaknesses and patterns observed, thus contributing to a more detailed evaluation of the effectiveness and efficiency of the algorithm in specific routing contexts.

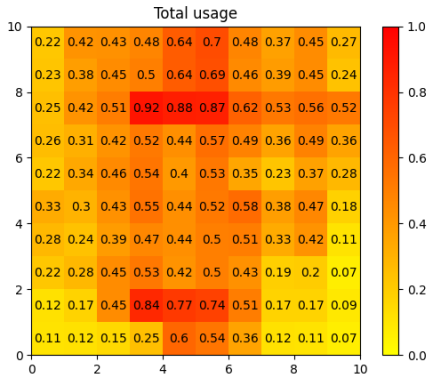
We begin our analysis with Test 5. The choice of this test is based on NEATRouter's ability to more efficiently find an optimal route, without congestion, which significantly reduces the need to perform additional iterations. This specific case will give us a more detailed view of how the implementation of NEATRouter can positively impact routing effectiveness, allowing us to explore in depth the advantages it offers in terms of speed and efficiency.

The Figure 5.1 shows the comparison of the final congestion of Test 5 after executing both methods, as can be seen when applying NEATRouter to route some nets, it is possible to reduce the use of resources in some congestion areas, thus helping the routing of

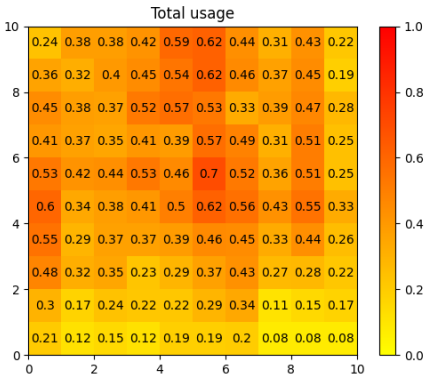
other nets is more efficient. The figure 5.2 illustrates the routing process for Test 5. It

Figure 5.1: Comparison of congested gcells using MazeRouter and NEATRouter for Test 5

(a) MazeRouter congestion



(b) NEATRouter congestion



is observed that the best individual of 100th generation is able to route around 3 nets in each iteration, matching the results of MazeRouter. The best 250th generation individual can route between 8-10 nets in each iteration, contributing to a more effective reduction of wirelength and congestion compared to MazeRouter. In 500th generation, the best individual manages to route 24 nets on the first attempt, addressing congested areas and eliminating the need for additional iterations. However, in 1000th generation, although this individual can route a greater number of nets, he worsens the routing of other nets, resulting in the need for additional iterations to address congestion.

The second test that we will analyze is Test 7. By applying the NEATRouter algorithm, we observe that, although the final routing may increase the wirelength, a reduction in resource use is achieved in each ggrid. This approach avoids overcongestion in specific areas of the grid, thus offering a balance between increased wirelength and efficiency in resource distribution.

The Figure 5.3 compares the final congestion after routing Test 7 with both methods. When we used NEATRouter to route some nets, we observed a reduction in resource usage on certain ggrids. However, we noticed that some nets experience a deterioration in their routing, resulting in increased resource usage in other areas of the grid. The Figure 5.4 presents the routing process for Test 7. The best individual of generation 100 manages to route 19 nets, obtaining similar results to MazeRouter in the same number of iterations. The best individual of generation 250 routes 28 nets, but this approach deteriorates the routing, making it difficult to improve and resulting in the maximum number of iterations (50) with congestion. The best individual of generation 500 reduces the number

Figure 5.2: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 5.

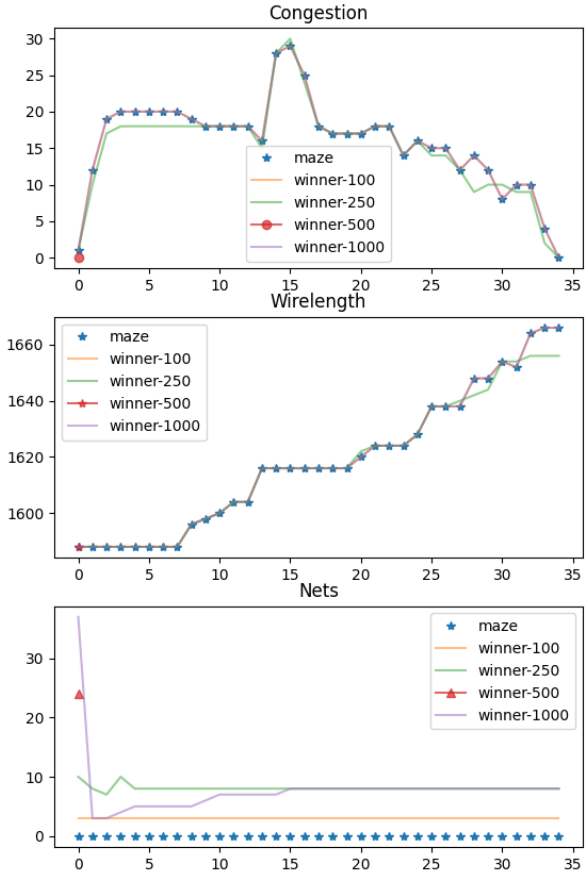
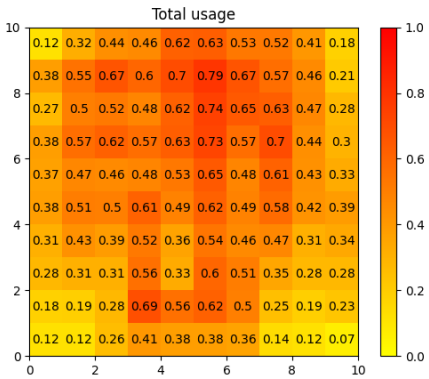
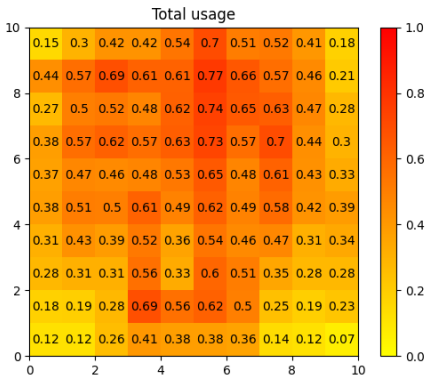


Figure 5.3: Comparison of congested gcells using MazeRouter and NEATRouter for Test 7

(a) MazeRouter congestion



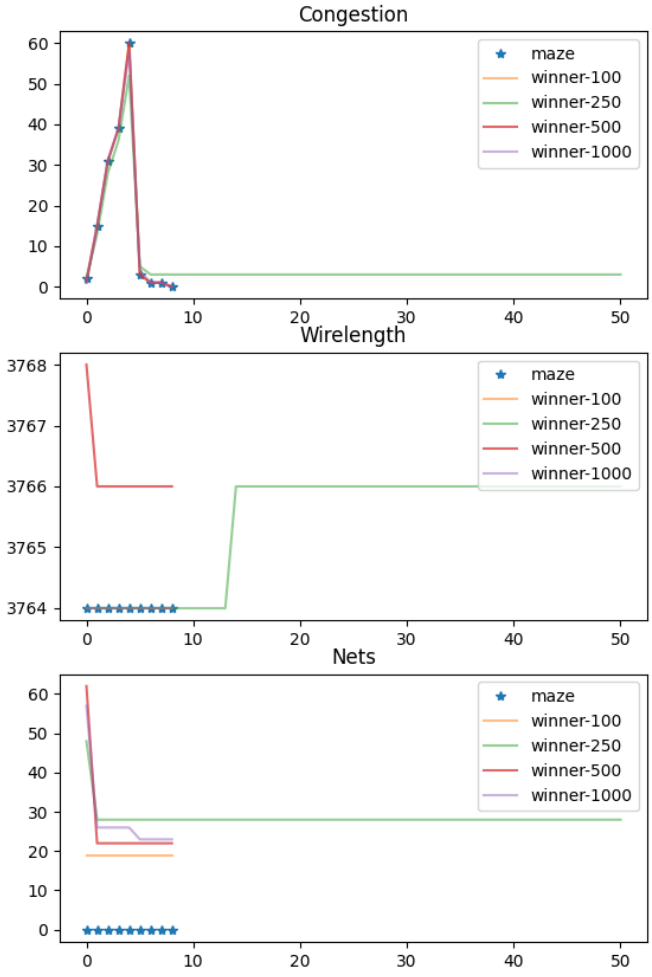
(b) NEATRouter congestion



of routed nets to 22, managing to eliminate congestion in the same number of iterations as MazeRouter, although with a slight increase in the final execution time. Finally, the best individual of the 1000 generation is able to route between 57 and 23 nets, which is enough to match the results of MazeRouter.

In the last test case the Test 11, we put our algorithm to a test by doubling the number of

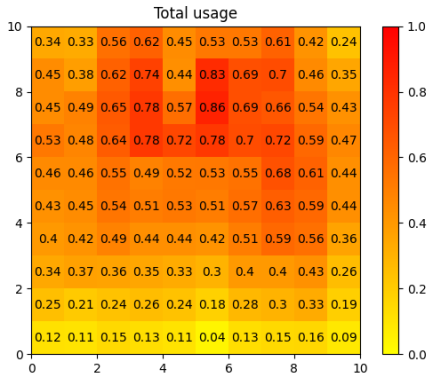
Figure 5.4: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 7.



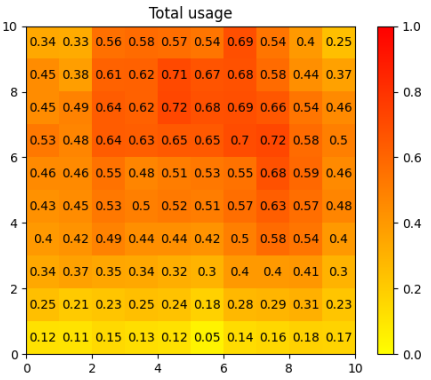
nets. Figure 5.5 shows the graph of the final congestion when using both methods. Once again, we observe that the congestion in the ggrids with our method behaves in a balanced way, avoiding overutilization of the resources of one of the ggrids. Figure 5.6 shows the behavior of the different individuals during Test 11. In the case of the individual from the

Figure 5.5: Comparison of congested gcells using MazeRouter and NEATRouter for Test 11

(a) MazeRouter congestion

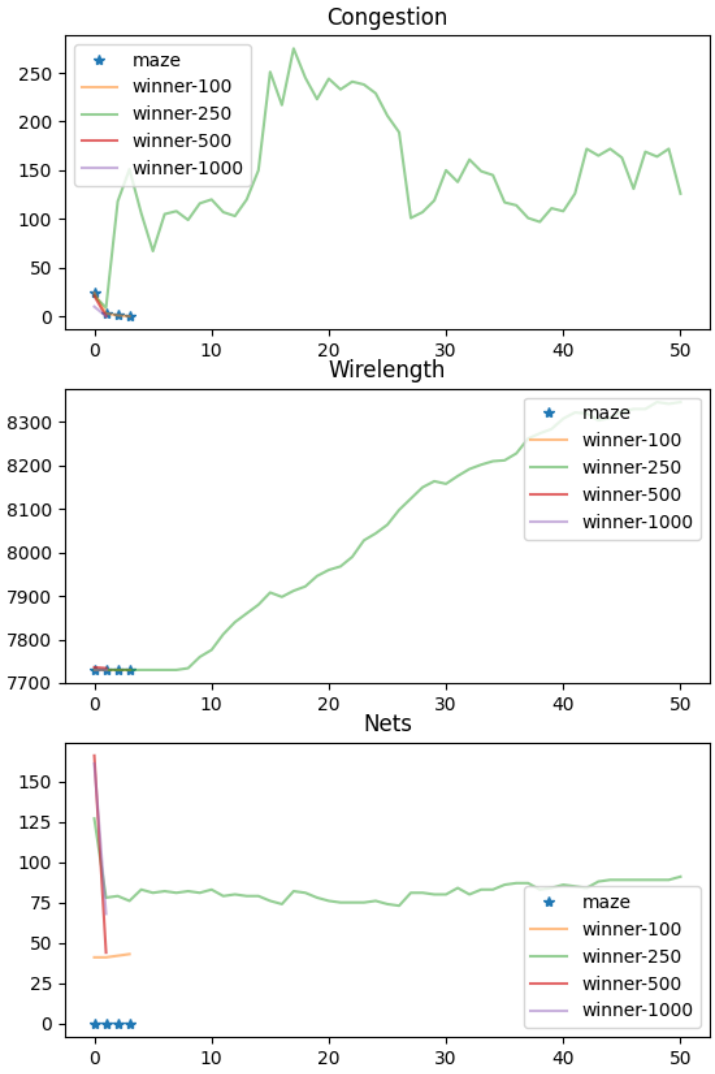


(b) NEATRouter congestion



250th generation, it is observed that he cannot find a solution without congestion. This is because the routing of certain nets is not being carried out optimally, causing incorrect management of resources. For the other individuals, a similar behavior to that of the MazeRouter is noted in terms of wirelength, but the reduction in congestion seems to be more pronounced in the 500th and 1000th generation individuals. In this case, the individual with the best result will be the one from the 1000th generation, managing to route 161 nets. By analyzing the test cases, we observe that the algorithm improves the results in terms of resources used and wirelength, in addition to reducing the number of iterations. However, during overtraining, individuals tend to deteriorate results on some tests. This is because they modify the routing of nets that are not initially in congestion zones, thus worsening the situation for the rest of the nets. Good control over which nets should or should not be routed by a NEAT neural network is crucial to obtain positive or negative results. We highlight that the most effective strategy to route with NEAT neural networks is to apply them in nets that present congestion in their routing. We also observed that different configurations can generate better or worse quality individuals, which underlines the importance of an optimal initial configuration for the NEAT algorithm and obtaining better routing results. Congestion and individual comparison graphs for the other test cases can be found in the appendix A.

Figure 5.6: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 11.



6 CONCLUSION AND FUTURE WORKS

In this work, we present NEATRouter as an innovative method to perform two-dimensional global routing of 2-pin nets. NEATRouter is implemented using the NEAT algorithm, which is a genetic algorithm used to optimize neural networks to perform a specific task. The results show that the proposed method tends to be efficient, facilitating the search for the shortest paths for a net, which impacts the final wirelength and reduces the use of resources in the routing area.

We performed a series of experiments to evaluate our method in different scenarios and compare it with another method called MazeRouter. This comparison was made because MazeRouter is an algorithm that is commonly used as part of several global routing algorithms, and can be complemented/replaced by our method. Based on the results, we can conclude that our method manages to find more optimal paths, reducing the number of iterations necessary to eliminate congestion. Furthermore, we observe that the resource distribution across the entire circuit tends to be more balanced compared to the MazeRouter.

As the execution time of the algorithm can become high with a large number of nets, our proposed algorithm previously requires the use of an algorithm that divides the total nets into sets. These sets will be routed by NEATRouter and then subjected to global routing using another algorithm. A contribution of our work is to reduce congestion, which is fundamental to improve routing.

As future work, our study suggests that the use of neural networks can improve the quality of routing, so the application of machine learning (ML) techniques has significant potential to achieve better results. For example, the Reinforcement Learning (RL) technique, which learns by reward stimulation to perform various tasks, could be useful for deciding movements during global routing. Another interesting point to investigate is the appropriate choice of input information for neural networks, as this can lead to better movement decisions.

REFERENCES

ABRAHAM, A. Artificial neural networks. In: ____. **Handbook of Measuring System Design**. John Wiley and Sons, Ltd, 2005. chp. 129. ISBN 9780471497394. Available from Internet: <https://onlinelibrary.wiley.com/doi/abs/10.1002/0471497398.mm421>.

ALPERT, C. J. The ispd98 circuit benchmark suite. In: **Proceedings of the 1998 International Symposium on Physical Design**. New York, NY, USA: Association for Computing Machinery, 1998. (ISPD '98), p. 80–85. ISBN 158113021X. Available from Internet: <https://doi.org/10.1145/274535.274546>.

CHANG, Y. J.; LEE, Y. T.; WANG, T. Nthu-route 2.0: A fast and stable global router. **IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**, IEEE, 2008. Available from Internet: <https://doi.org/10.1109/ICCAD.2008.4681595>.

CHO, M. et al. Boxrouter 2.0: architecture and implementation of a hybrid and robust global router. **IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**, IEEE, 2007. Available from Internet: <https://doi.org/10.1109/ICCAD.2007.4397314>.

CHU, C.; WONG, Y.-C. Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 27, 2007. Available from Internet: <https://doi.org/10.1109/TCAD.2007.907068>.

FLACHA, G. et al. Rsyn: An extensible physical synthesis framework. **International Symposium on Physical Design**, ACM, 2017. Available from Internet: <https://doi.org/10.1145/3036669.3038249>.

FOGACA, M. P. A new quadratic formulation for incremental timing-driven placement. UFRGS lume, 2016. Available from Internet: <http://hdl.handle.net/10183/164067>.

FOGACA, M. P. Finding placement-relevant clustersWith fast modularity-based clustering. UFRGS lume, 2020. Available from Internet: <http://hdl.handle.net/10183/215350>.

GAO, J. R.; WU, P. C.; WANG, T. C. A new global router for modern designs. **2008 Asia and South Pacific Design Automation Conference**, IEEE, 2008. Available from Internet: <https://doi.org/10.1109/ASPDAC.2008.4483948>.

HALDURAI, L.; MADHUBALA, T.; RAJALAKSHMI, R. A Study on Genetic Algorithm and its Applications. **International Journal of Computer Science and Engineering**, v. 4, 2016.

HE, J. et al. Sproute 2.0: A detailed-routability-driven deterministic parallel global router with soft capacity. In: **2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2022. p. 586–591.

HE, J. et al. SPRoute: A Scalable Parallel Negotiation-based Global Router. **EEE International Conference on Computer-Aided Design**, IEEE/ACM, 2019. Available from Internet: <https://doi.org/10.1109/ICCAD45719.2019.8942105>.

HENTSCHKE, R. F. Algoritmos para o posicionamento de células em circuitos VLSI. UFRGS lume, 2002. Available from Internet: <http://hdl.handle.net/10183/2598>.

HUANG, G. et al. Machine Learning for Electronic Design Automation: A Survey. **ACM Transactions on Design Automation of Electronic Systems**, ACM, v. 26, 2021. Available from Internet: <https://doi.org/10.1145/3451179>.

JORDAN1, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, v. 349, p. 255–260, 2015. Available from Internet: <http://www.cs.cmu.edu/~tom/pubs/Science-ML-2015.pdf>.

KAHNG, A. et al. **VLSI Physical Design: From Graph Partitioning to Timing Closure**. Springer Netherlands, 2011. ISBN 9789048195916. Available from Internet: <https://books.google.com.br/books?id=DWUGHyFVpboC>.

LEE, C. Y. An algorithm for path connections and its applications. **IRE Transactions on Electronic Computers**, EC-10, 1961. Available from Internet: <https://doi.org/10.1109/TEC.1961.5219222>.

LI, H. et al. Dr. cu 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction. **IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**, 2019.

LIAO, H. et al. A deep Reinforcement Learning approach for Global Routing. **Jornal of Mechanic Design**, v. 142, 2019. Available from Internet: <https://doi.org/10.1115/1.4045044>.

LIU, J. et al. CUGR: Detailed-Routability-Driven 3D Global Routing with Probabilistic Resource Model. **ACM/IEEE Design Automation Conference (DAC)**, ACM/IEEE, 2020. Available from Internet: <https://cwpui.com/doc/c10.pdf>.

LIU, S. et al. Fastgr: Global routing on cpu–gpu with heterogeneous task graph scheduler. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 42, n. 7, p. 2317–2330, 2023.

LIU, W. et al. 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**, IEEE, v. 32, 2013. Available from Internet: <https://doi.org/10.1109/TCAD.2012.2235124>.

MCINTYRE et al. **neat-python**. Available from Internet: <https://github.com/CodeReclaimers/neat-python>.

MONTEIRO, E. M. R. Roteamento global de circuitos VLSI. UFRGS lume, 2023. Available from Internet: <http://hdl.handle.net/10183/258049>.

NAM, G.-J.; SZE, C.; YILDIZ, M. The ispd global routing benchmark suite. In: **Proceedings of the 2008 International Symposium on Physical Design**. New York, NY, USA: Association for Computing Machinery, 2008. (ISPD '08), p. 156–159. ISBN 9781605580487. Available from Internet: <https://doi.org/10.1145/1353629.1353663>.

NAM, G.-J. et al. Ispd placement contest updates and ispd 2007 global routing contest. In: **Proceedings of the 2007 International Symposium on Physical Design**. New York, NY, USA: Association for Computing Machinery, 2007. (ISPD '07), p. 167. ISBN 9781595936134. Available from Internet: <https://doi.org/10.1145/1231996.1232029>.

NUNES, L. de M. Redução de congestionamento em roteamento global de circuitos VLSI. UFRGS lume, 2013. Available from Internet: <http://hdl.handle.net/10183/151338>.

OLIVEIRA, A. S. Initial detailed routing algorithms. UFRGS lume, 2021. Available from Internet: <http://hdl.handle.net/10183/221873>.

PAN, M. et al. FastRoute: An efficient and high-quality global router. Hindawi Publishing Corporation, 2012. Available from Internet: <https://doi.org/10.1155/2012/608362>.

PATRASCU, C.-B.; IANCU, D.-T. Neat algorithm for simple games. In: **2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)**. [S.l.: s.n.], 2023. p. 1–6.

PEREZ-LIEBANA, D.; ALAM, M. S.; GAINA, R. D. Rolling horizon neat for general video game playing. In: **2020 IEEE Conference on Games (CoG)**. [S.l.: s.n.], 2020. p. 375–382.

PLACIDO, H. Posicionamento global de células em circuitos VLSI. UFRGS lume, 2016. Available from Internet: <http://hdl.handle.net/10183/147659>.

REIMANN, T. J. Roteamento global de circuitos VLSI. UFRGS lume, 2013. Available from Internet: <http://hdl.handle.net/10183/71269>.

RINNARV, J.; BRINK, P. Machine learning - neuroevolution for designing chip circuits/pathfinding. **Master of Science in Engineering - ComputerScience and Technology**, 2017. Available from Internet: <urn:nbn:se:kth:diva-210178>.

SCHLICHTMANN, U. et al. Overview of 2019 cad contest at iccad. In: **2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2019. p. 1–2.

STANLEY, K. O.; MIIKKULAINEN, R. Efficient Evolution of Neural Network Topologies. **Congress on Evolutionary Computation**, IEEE, 2002. Available from Internet: <https://doi.org/10.1109/CEC.2002.1004508>.

TANG, H. et al. A Survey on Steiner Tree Construction and Global Routing for VLSI Design. **IEEE Access**, IEEE, v. 8, 2020. Available from Internet: <https://doi.org/10.1109/ACCESS.2020.2986138>.

TUMELERO, D. Exploração de paralelismo no roteamento global de circuitos VLSI. UFRGS lume, 2015. Available from Internet: <http://hdl.handle.net/10183/119081>.

WEN, R. et al. Neuroevolution of augmenting topologies based musculor-skeletal arm neurocontroller. In: **2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)**. [S.l.: s.n.], 2017. p. 1–6.

WILLIGEN, W. H. van; HAASDIJK, E.; KESTER, L. J. H. M. Evolving intelligent vehicle control using multi-objective neat. In: **2013 IEEE Symposium on Computational Intelligence in Vehicles and Transportation Systems (CIVTS)**. [S.l.: s.n.], 2013. p. 9–15.

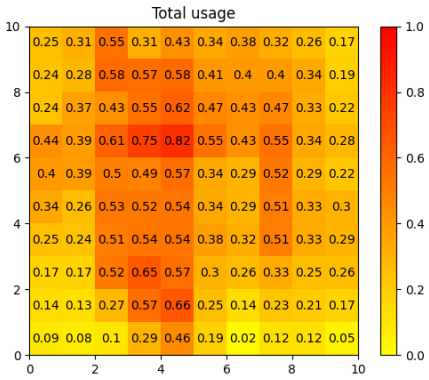
WITTKAMP, M.; BARONE, L.; HINGSTON, P. Using neat for continuous adaptation and teamwork formation in pacman. In: **2008 IEEE Symposium On Computational Intelligence and Games**. [S.l.: s.n.], 2008. p. 234–242.

APPENDIX A — CONGESTION AND INDIVIDUAL COMPARISON GRAPHS FOR THE TEST CASES

A.0.1 Test 1

Figure A.1: Comparison of congested gcells using MazeRouter and NEATRouter for Test 1

(a) MazeRouter congestion



(b) NEATRouter congestion

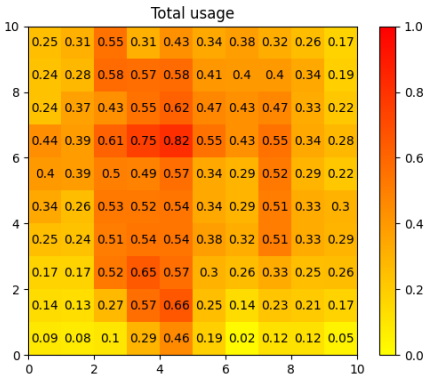
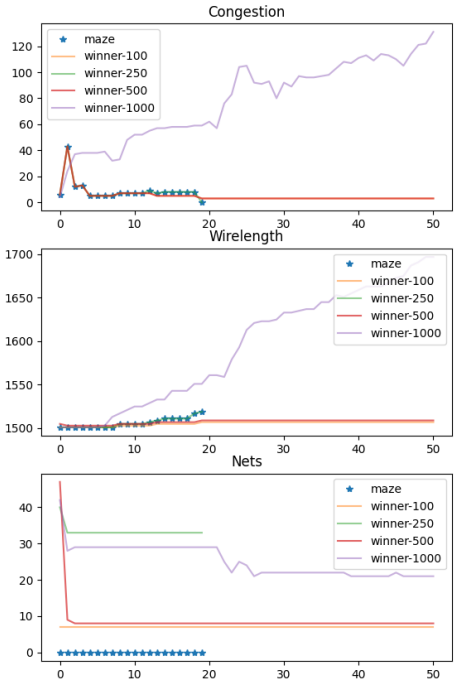


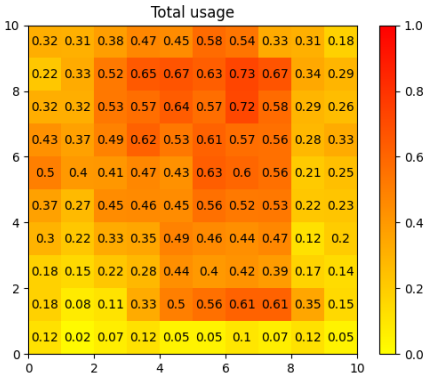
Figure A.2: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 1



A.0.2 Test 2

Figure A.3: Comparison of congested gcells using MazeRouter and NEATRouter for Test 2

(a) MazeRouter congestion



(b) NEATRouter congestion

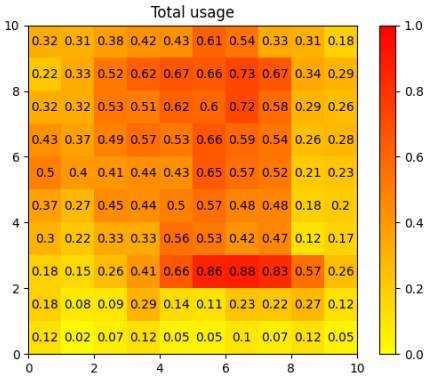
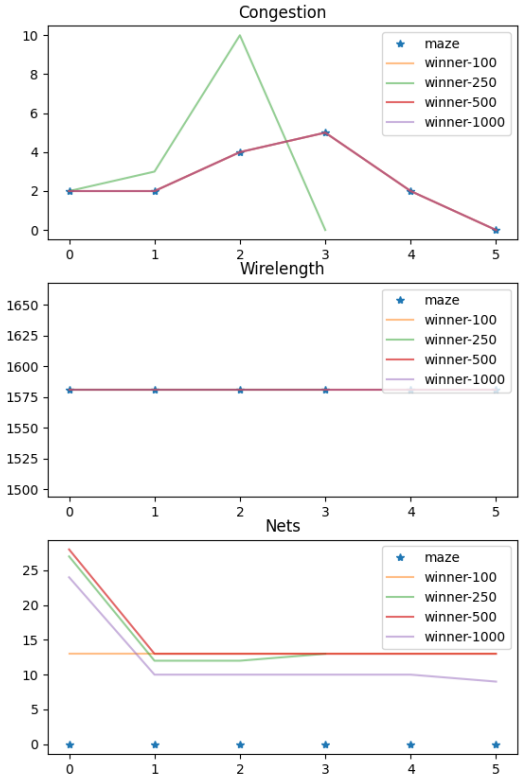


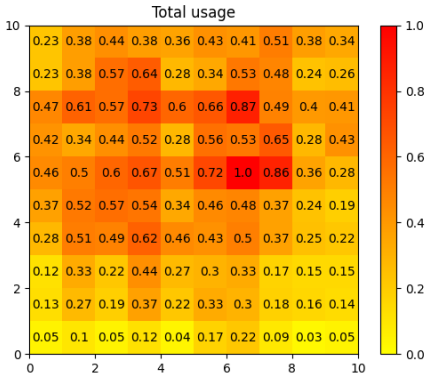
Figure A.4: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 2



A.0.3 Test 3

Figure A.5: Comparison of congested gcells using MazeRouter and NEATRouter for Test 3

(a) MazeRouter congestion



(b) NEATRouter congestion

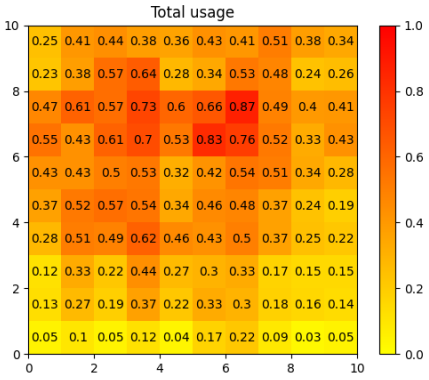
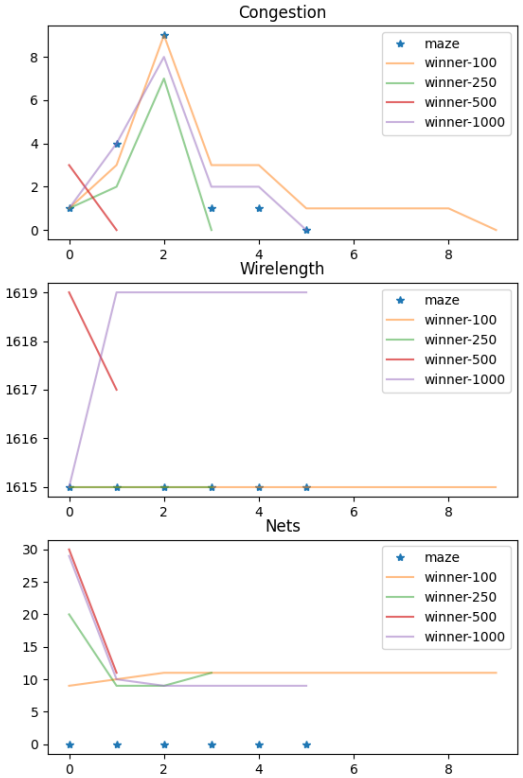


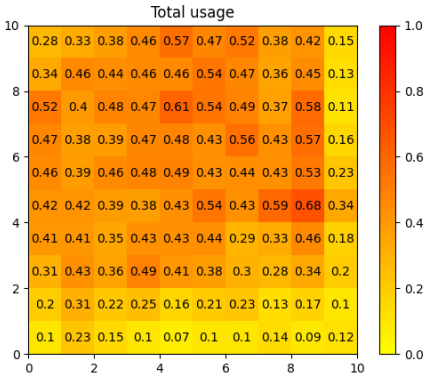
Figure A.6: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 3



A.0.4 Test 4

Figure A.7: Comparison of congested gcells using MazeRouter and NEATRouter for Test 4

(a) MazeRouter congestion



(b) NEATRouter congestion

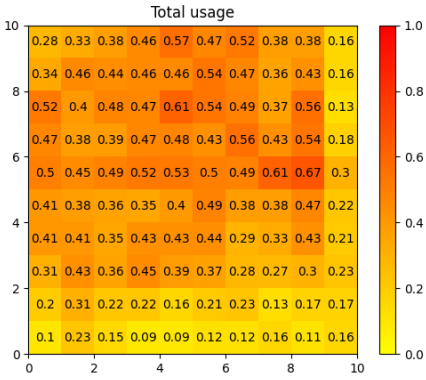
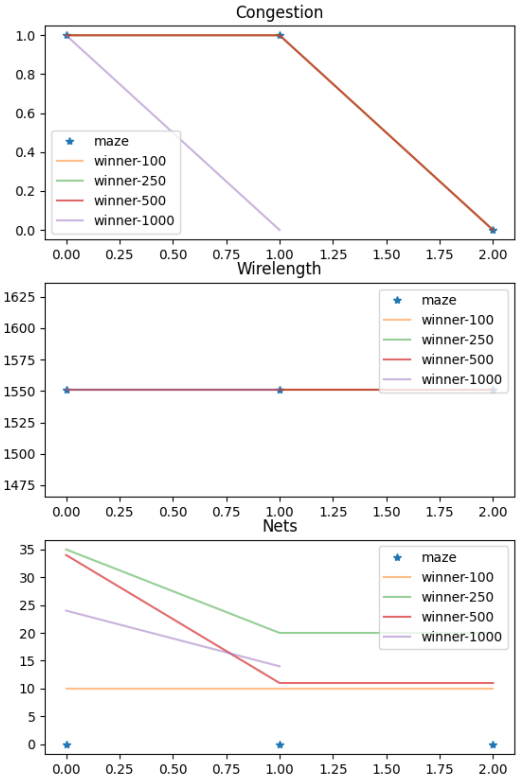


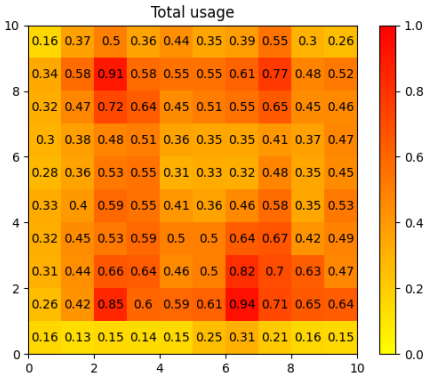
Figure A.8: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 4



A.0.5 Test 6

Figure A.9: Comparison of congested gcells using MazeRouter and NEATRouter for Test 6

(a) MazeRouter congestion



(b) NEATRouter congestion

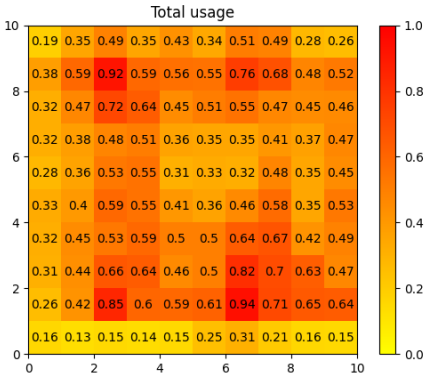
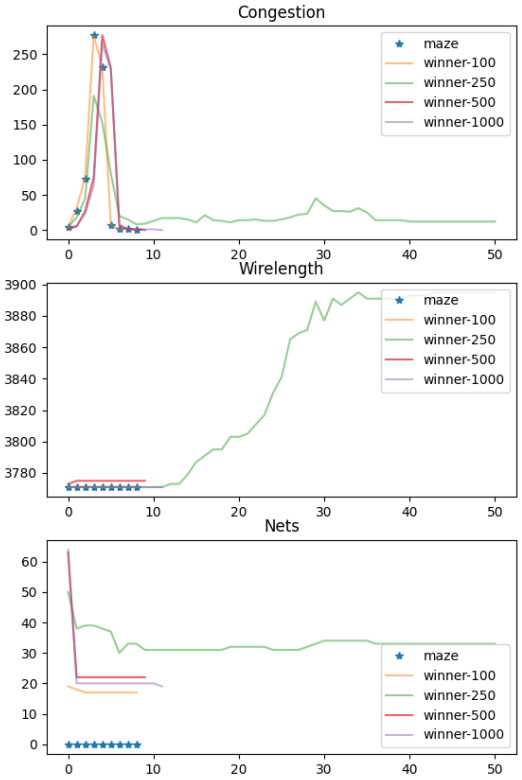


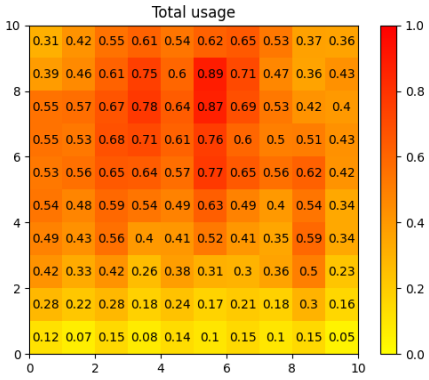
Figure A.10: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 6



A.0.6 Test 8

Figure A.11: Comparison of congested gcells using MazeRouter and NEATRouter for Test 8

(a) MazeRouter congestion



(b) NEATRouter congestion

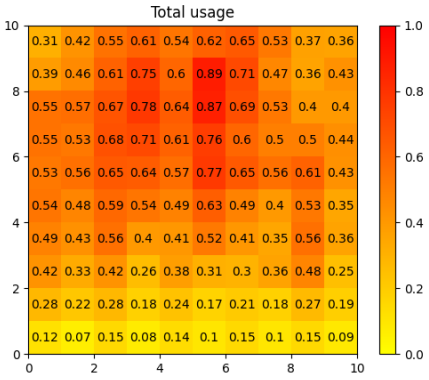
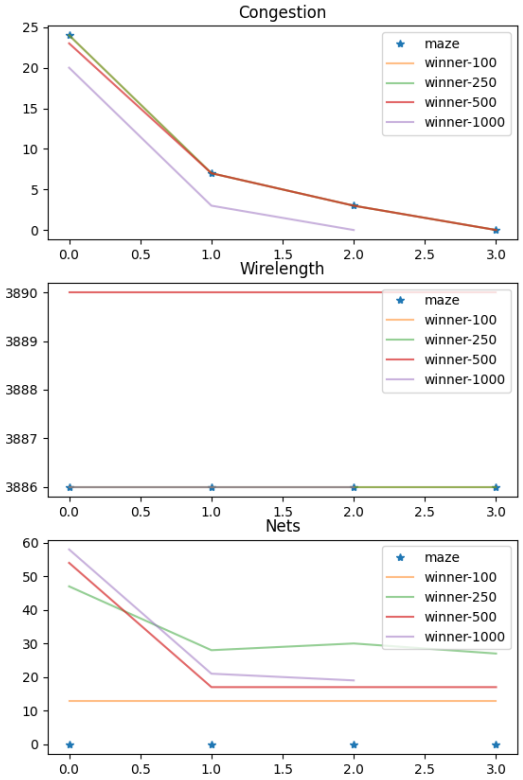


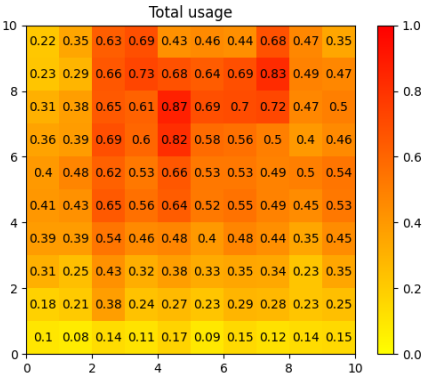
Figure A.12: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 8



A.0.7 Test 9

Figure A.13: Comparison of congested gcells using MazeRouter and NEATRouter for Test 9

(a) MazeRouter congestion



(b) NEATRouter congestion

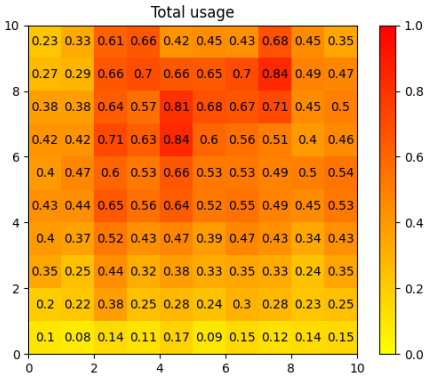
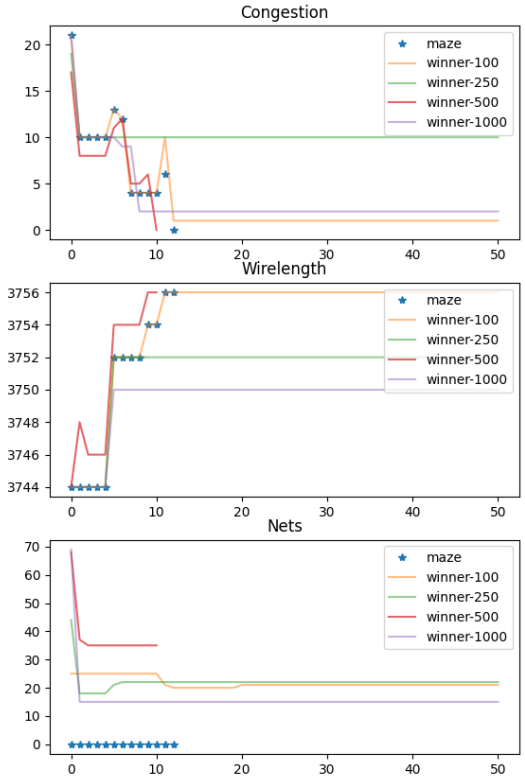


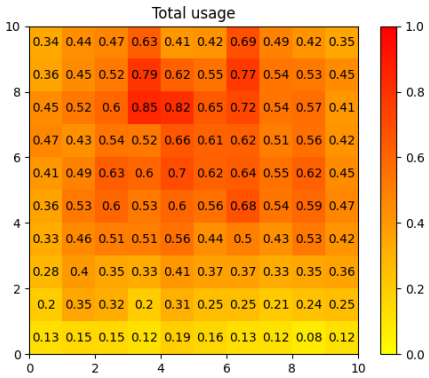
Figure A.14: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 9



A.0.8 Test 10

Figure A.15: Comparison of congested gcells using MazeRouter and NEATRouter for Test 10

(a) MazeRouter congestion



(b) NEATRouter congestion

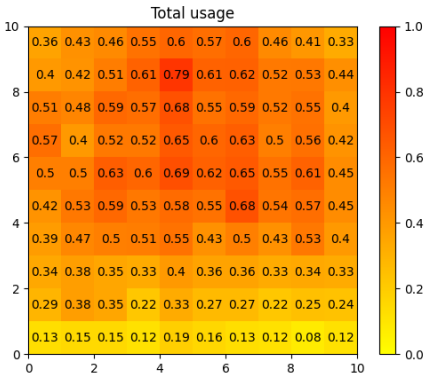
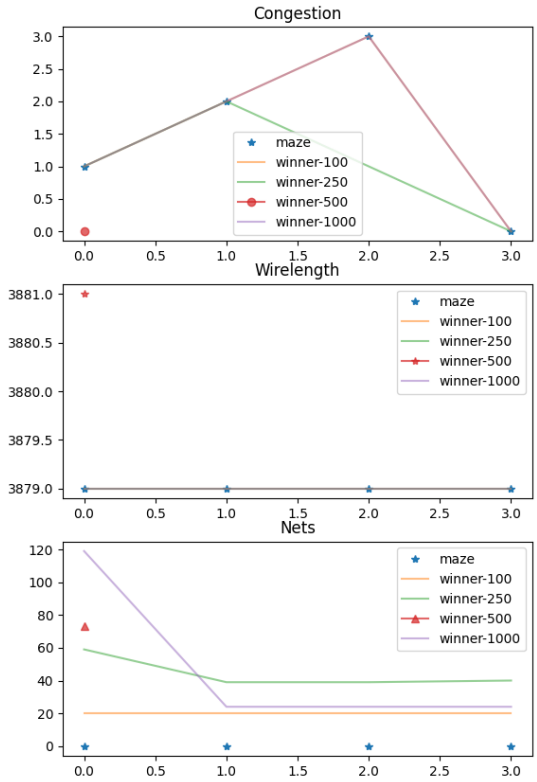


Figure A.16: Comparison of the values of congestion, wirelength and nets routed by NEATRouter in the ripup and reroute process using Mazeroute and the best NEAT neural networks to route the test 10



APPENDIX B — RESUMO EXPANDIDO EM PORTUGUÊS

Título da Dissertação de Mestrado: NEATRouter: Uma nova ferramenta para roteamento global 2D.

Resumo: O desenvolvimento tecnológico tem um impacto significativo em nossas vidas, ajudando-nos em muitos aspectos do nosso dia a dia. Este impacto gera a necessidade de melhorar constantemente a fabricação de circuitos integrados, tornando-os cada vez mais potentes para realizar operações complexas e com dimensões menores para que possam ser integrados em diversos cenários. A inovação dos circuitos integrados é importante para o avanço de áreas como a inteligência artificial, a Internet das Coisas e a computação quântica.

A área de Automação de Projeto Eletrônico (EDA) é responsável pelo desenvolvimento de ferramentas de software que facilitam o projeto de circuitos integrados. Essas ferramentas são essenciais para lidar com a complexidade e o tamanho dos circuitos modernos. Com o aumento da procura por circuitos cada vez mais complexos, o custo de fabrico dos mesmos aumenta, levando à necessidade de criar ferramentas inovadoras que ajudem a automatizar as tarefas de projeto de circuitos integrados. Em um fluxo de projeto, a etapa de roteamento desempenha um papel importante, pois nesta etapa são estabelecidas conexões entre os pinos das redes de conexões de um circuito integrado por meio de camadas metálicas. À medida que os circuitos ficam cada vez mais complexos, contendo milhões de componentes em pequenas áreas, surge a necessidade de dividir a tarefa de roteamento em duas subetapas: roteamento global e roteamento detalhado.

O roteamento global tem como foco realizar o roteamento inicial das nets, buscando reduzir o comprimento do fio (wirelength) e congestionamento. Isto envolve realizar o roteamento para garantir que as rotas principais não gerem congestionamentos, o que poderia afetar o desempenho do circuito. Esse roteamento inicial é então usado como guia para o roteamento detalhado, responsável por encontrar as conexões finais de cada net. Nesta fase, as rotas são atribuídas a camadas metálicas específicas, a fim de atender a restrições específicas.

Aprendizagem de Máquina (ML) é uma área dentro da Inteligência Artificial que tem como objetivo desenvolver algoritmos e modelos que aprendam padrões a partir de um conjunto de dados e possam classificar sob um critério ou gerar novas informações (ex. predição), tudo sem intervenção humana. Algoritmos de aprendizado de máquina são amplamente utilizados em diversas áreas como robótica, processamento de linguagem

natural, reconhecimento de fala, visão computacional, entre outras.

Neste trabalho, propomos o NEATRouter como um roteador global bidimensional para nets de 2 pinos, cujo objetivo é gerar rotas de alta qualidade em termos de comprimento de fio e congestionamento. NEATRouter busca complementar ou substituir o algoritmo MazeRouter, que é usado em vários algoritmos de roteamento global de última geração. Utilizamos o algoritmo Neuroevolution of Augmenting Topologies (NEAT), que é uma variação de algoritmos genéticos capaz de otimizar redes neurais artificiais para problemas específicos.

Método de treinamento: Adaptamos o algoritmo NEAT para gerar redes neurais que otimizam o problema de roteamento de net de 2 pinos. Definimos como valores de entrada uma lista que inclui a quantidade de recursos de roteamento disponíveis, as distâncias dos vizinhos até a posição final e a quantidade máxima de valor histórico, o que ajuda a identificar quais posições estão congestionadas. Essas redes neurais determinam o conjunto de movimentos necessários para rotear uma net, buscando reduzir o comprimento dos fios e o congestionamento. A função de fitness do algoritmo NEAT se concentrará na conexão dos pinos de uma net, minimizando o uso de recursos de roteamento e comprimento de fio. Esta abordagem permite-nos melhorar a eficiência e a qualidade do roteamento, garantindo soluções ideais para nets complexas.

Roteamento global: Nosso método usará as redes neurais geradas para rotear cada net de 2 pinos de caso de teste. Primeiro, um roteamento inicial será realizado. Caso existam posições com congestionamento, nosso algoritmo utilizará o método Rip-Up and Reroute para redirecionar as nets, aumentando a cada iteração o custo de utilização das posições onde foi detectado congestionamento. Este método visa que as nets procurem caminhos alternativos e liberem áreas congestionadas. Em cada iteração Rip-Up e Reroute, nosso algoritmo roteará cada rede usando as redes neurais geradas. Nos casos em que o NEATRouter não consegue completar o roteamento de uma net, usamos o método MazeRouter para garantir as conexões de todas as nets. Esta estratégia combinada procura garantir um roteamento eficiente e eficaz, minimizando o comprimento do fio e o congestionamento.

Geramos o conjunto de testes para realizar nossos experimentos e comparamos nossos resultados com os do algoritmo MazeRouter. As métricas que usamos para comparação foram comprimento do fio, quantidade de recursos usados e tempo de execução. Estas métricas permitiram avaliar a eficiência e eficácia do NEATRouter em comparação com o método tradicional, fornecendo informações sobre o seu desempenho em termos de qual-

idade de roteamento e consumo de recursos.

Nossos resultados mostram que nosso algoritmo tem potencial para reduzir a quantidade de recursos de roteamento usados em 1% a 5% em comparação ao MazeRouter. Também encontramos melhorias no comprimento do fio em alguns testes, embora isso leve a um aumento no tempo de execução. A principal contribuição do nosso trabalho é reduzir o congestionamento, o que é fundamental para melhorar a qualidade do roteamento. Para que nosso algoritmo consiga lidar com um maior número de nets e tenha melhor desempenho em termos de tempo de execução, ele deve ser complementado com um algoritmo superior que divida o número de nets em conjuntos menores. Então nosso algoritmo pode rotear cada conjunto e no final realizar o roteamento global de todos os conjuntos das nets.

Como trabalho futuro, nosso estudo sugere que o uso de redes neurais pode melhorar a qualidade do roteamento na tarefa de roteamento global. Além disso, a aplicação de outras técnicas de aprendizado de máquina tem potencial para obter melhores resultados. Outro tema interessante para pesquisa é explorar diferentes formas de definir entradas para redes neurais, pois isso pode levar a uma melhor tomada de decisão na escolha do próximo movimento dependendo das restrições.