



**UNIVERSITÀ
DEGLI STUDI
DI TRIESTE**

UNIVERSITÀ DEGLI STUDI DI TRIESTE

**XXXVIII CICLO DEL DOTTORATO DI RICERCA IN
APPLIED DATA SCIENCE AND ARTIFICIAL
INTELLIGENCE**

SU FONDI DI INFINEON TECHNOLOGIES

Automating the Layout of Analog Circuits:

A Machine Learning-Based Approach

Settore scientifico-disciplinare: INF/01 INFORMATICA

DOTTORANDO
DAVIDE BASSO

COORDINATORE
PROF. FRANCESCO PAULI

SUPERVISORE DI TESI
PROF. LUCA BORTOLUSSI

CO-SUPERVISORE DI TESI
DR. MIRJANA VIDENOVIC-MISIC

CO-SUPERVISORE DI TESI
DR. HUSNI HABAL

ANNO ACCADEMICO 2024/2025



**UNIVERSITÀ
DEGLI STUDI
DI TRIESTE**



APPLIED DATA SCIENCE &
ARTIFICIAL INTELLIGENCE



UNIVERSITÀ DEGLI STUDI DI TRIESTE

INFINEON TECHNOLOGIES

Ph.D. in Applied Data Science & Artificial Intelligence

XXXVIII cycle

Automating the Layout of Analog Circuits: A Machine Learning-Based Approach

Candidate

Davide Basso

Supervisors

Prof. Luca Bortolussi

Dr. Mirjana Videnovic-Misic

Dr. Husni Habal

*To my family,
for your unconditioned love.*

Abstract

Processes are nowadays permeated by the usage of artificial intelligence (AI) techniques, aiming at boosting their efficiency and accessibility. Such extensive adoption has been fueled by remarkable progress in the chip manufacturing industry, particularly through the reduction of transistor and circuit component sizes. These advancements have made available an enormous amount of computational power, enabling the success of data hungry models, especially in fields such as natural language processing. Leveraging AI to accelerate and optimize chip design has, in turn, emerged as a critical research direction. In the digital domain, automated, learning-based frameworks have achieved significant progress. On the contrary, the automation of analog circuit design, especially for what concerns the layout phase, continues to lag behind. This gap arises from unique electrical and topological constraints that make analog layouts more complex to automate. Consequently, the proposed methods for streamlining analog layout remain limited in their industrial applicability.

This thesis presents a wide range of techniques for automatically generating the layout of analog integrated circuits, with a specific focus on reinforcement learning (RL) for floorplanning, along with pathfinding and deterministic approaches for efficient routing. We start with a dual development of a floorplanning engine, one combining RL and simulated annealing (SA) and another solely relying on RL mimicking the SA search process. An obstacle-avoiding rectilinear Steiner tree global router is integrated with the floorplanning engine into an existing procedural layout generation framework, enabling complete layout synthesis. We demonstrate the effectiveness of learning-based approaches in exploring a large solution space better than metaheuristic techniques, while also reducing runtimes compared to manually crafted layouts. To improve generalization and transferability, we devise a novel floorplanning solution that combines relational graph convolutional neural networks (R-GCNs) with RL to scale layout generation to more complex circuits. Devices are placed on a discretized grid, providing greater flexibility for the RL agent to optimize circuit area and proxy wirelength metrics. Additionally, we introduce a plug-and-play integration based on a beam search strategy to enhance the RL inference process, allowing for flexible objective weighting tailored to specific use cases and addressing congestion without policy finetuning. Lastly, we present a routing-aware version of the floorplanning engine that builds upon the R-GCN RL approach. This enhanced framework leverages a novel U-Net policy, dynamic routing resource estimation, and revised reward scheme for delivering routing-ready floorplans. A prototype A* rip-up and reroute routing engine completes the workflow, finally allowing for a completely automatic generation of layouts. Results from more than 20 use cases show that the proposed framework improves routability by 73.4% and overall placement quality over prior methods, reducing layout time by 67.3% compared to a manual approach.

Acknowledgments

With the completion of this thesis, I mark the conclusion of one of the most stimulating and significant journeys of my life. Embarking on a path of this kind was not straightforward, and, although it was rich in challenges, I look back with deep satisfaction on the growth I have undergone in these years, both academically and humanly. The fact that I had to question myself repeatedly allowed me to reflect on what truly matters and what does not, what contributes positively and what must, inevitably, be left behind. Ultimately, this process led to a profound introspective journey, which shaped some traits that I have always overlooked and that I am extremely proud of now.

I take the chance to first thank my supervisor, Luca Bortolussi, for the trust placed in me from the very beginning, for the freedom to experiment, and for the close collaboration that enabled us to achieve significant results. A heartfelt thank you also goes to my supervisors at Infineon, Mirjana Videnovic-Misic and Husni Habal, without your support none of this would have been possible.

My sincere thanks go to my fellow doctoral students, companions in the emotional highs and lows that characterize the PhD experience. Knowing that one is not alone in facing the inevitable challenges of such a journey has made everything more bearable. Many thanks to the Lago Duria office group and to Dr. Pong for our calls.

Thanks as well to the friends of a lifetime and to the new ones I have met along this journey. One of the most rewarding aspects of this experience has been the opportunity to meet people who have now become true friends. A heartfelt thought goes to those who have always been there, and for whom I will always be there in return.

Dulcis in fundo, I dedicate the fruits of this work to my family. You have always stood by me and supported me, helping me bring out the best in myself even when I struggled to see it. This is what being a family means, being there and giving unconditional love, above all else. I'm sure this will remain a constant in our lives, for which I am profoundly grateful, and I'll do whatever it is possible to keep it in that way.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Analog Layout Basics	2
1.1.1 Floorplanning	2
1.1.2 Routing	3
1.2 Motivation	4
1.3 Contributions	6
1.4 Structure of the Thesis	7
1.5 Publications	8
I Background	9
2 Analog Layout Synthesis	11
2.1 Layout of Individual Devices	11
2.1.1 Device Layout Automation	13
2.2 Floorplanning	14
2.2.1 Circuit Floorplanning Automation	15
2.3 Routing	19
2.3.1 Circuit Routing Automation	20
2.4 Post-Layout Verification	21
3 Optimization Techniques	23
3.1 Stochastic Methods	23
3.1.1 Simulated Annealing	24
3.1.2 Population-Based Approaches	25
3.2 Machine Learning	28
3.2.1 Deep Learning	30
3.2.2 Reinforcement Learning	35
4 Graph-Based Optimization for Routing	45
4.1 A* Algorithm	45
4.2 Steiner Trees	47

II	Machine Learning-Based Layout Automation	49
5	Fast ML-driven Analog Layout using Reinforcement Learning and Steiner Trees	51
5.1	Introduction	51
5.2	Problem Definition	53
5.2.1	Floorplanning	53
5.2.2	Routing	53
5.3	Automatic Layout Generation	53
5.3.1	Structures Recognition and Device Shapes Generation	53
5.3.2	<i>RL-SA</i> Floorplanning	54
5.3.3	<i>Annealed-RL</i> Floorplanning	55
5.3.4	RL Training Setup	55
5.3.5	Post-processing & Routing	55
5.4	Experimental Results	56
5.5	Discussion	58
5.6	Related Works on Fully Automated Analog Layout Frameworks	59
6	Effective Analog ICs Floorplanning with Relational Graph Neural Networks and Reinforcement Learning	61
6.1	Introduction	61
6.2	Methodology	62
6.2.1	Reinforcement Learning Formulation	62
6.2.2	R-GCN Circuit Representation Learning	63
6.2.3	<i>R-GCN RL</i> -based Floorplanning	65
6.2.4	Routing and Final Layout Generation	69
6.3	Experimental Results	69
6.3.1	RL Training Setup	69
6.3.2	Comparison Against Baselines	70
6.3.3	Evaluation of Complete Layouts	70
6.4	Discussion	73
7	Beam Search Enhanced Reinforcement Learning for Analog ICs Floorplanning	75
7.1	Introduction	75
7.2	Methodology	76
7.2.1	Beam Search	76
7.2.2	Congestion Handling	76
7.2.3	Beam Search and Reinforcement Learning: <i>BS-RL</i>	77
7.3	Experimental Results	78
7.3.1	Experiment Details	78
7.3.2	Comparison Against Baselines	80
7.4	Discussion	80

8	Advancing Routing-Awareness in Analog ICs Floorplanning	83
8.1	Introduction	83
8.2	Methodology	84
8.2.1	Pin-Enhanced Circuit Graph	84
8.2.2	A More Routing-Aware RL Agent	86
8.2.3	A* Routing Engine	90
8.3	Experimental Results	92
8.3.1	Floorplanning Metrics Assessment	93
8.3.2	Routing Metrics Assessment	94
8.4	Discussion	95
III	Closing Remarks	99
9	Conclusion	101
9.1	Limitations and Future Research	102
	List of Figures	105
	List of Tables	107
A	Supplementary Material for Chapter 8	123

Chapter 1

Introduction

The rapid advancements in artificial intelligence (AI), particularly in machine learning (ML) and deep learning (DL), have significantly transformed the semiconductor industry in recent years. These sophisticated technologies are driving an ever-increasing demand for high computational power to support their growing range of applications. In this context, the development of integrated circuits (ICs) plays a pivotal role in providing the robust and efficient hardware foundation necessary to harness the full potential of breakthroughs achieved in both academic research and industrial innovation. Analog ICs, including components such as analog-to-digital converters (ADCs), low-dropout (LDO) voltage regulators, and phase-locked loops (PLLs) for clock generation, are integral to this process. These circuits frequently coexist with large digital portions, resulting in mixed-signal systems that combine the precision and sensitivity of analog components with the computational capabilities of their digital counterparts.

The circuit design process faces several challenges, particularly when comparing digital and analog circuits. The discrete nature of digital signals, combined with characteristics such as low voltage operation and negligible static current consumption, has facilitated the development of reliable automated design tools for digital systems. Unfortunately, the same cannot be said for analog circuits. Analog circuit design is in fact inherently more complex due to the continuous nature of the signals they process and the increasing shrink in size of transistors. Several factors can significantly impact circuit performance, including the need to handle a wide range of current levels, from low to high voltage currents. Moreover, routing in analog circuits must often be uniquely sized for individual transistors to manage varying static currents, further complicating the automation process. These details make achieving full automation in analog circuit design a challenging and still largely unresolved task.

To address these difficulties, electrical design automation (EDA) [29] commercial tools, such as Cadence Virtuoso [17], are essential in assisting analog designers by alleviating bottlenecks in both the design and layout processes. The design stage, which involves defining the circuit's topology, sizing, and structure based on the intended application and performance requirements, has witnessed significant advancements through both academic and industrial efforts aimed at revolutionizing this process. For example, ANAGEN [136], a project run by Infineon Technologies, proposes a trailblazing shift from the traditional analog design workflow to a programmatic paradigm, enabling faster schematic generation as in Figure 1.1, and improved reuse of designs. Similarly, various evolutionary-based [169] and learning-based [102, 141] approaches have been proposed

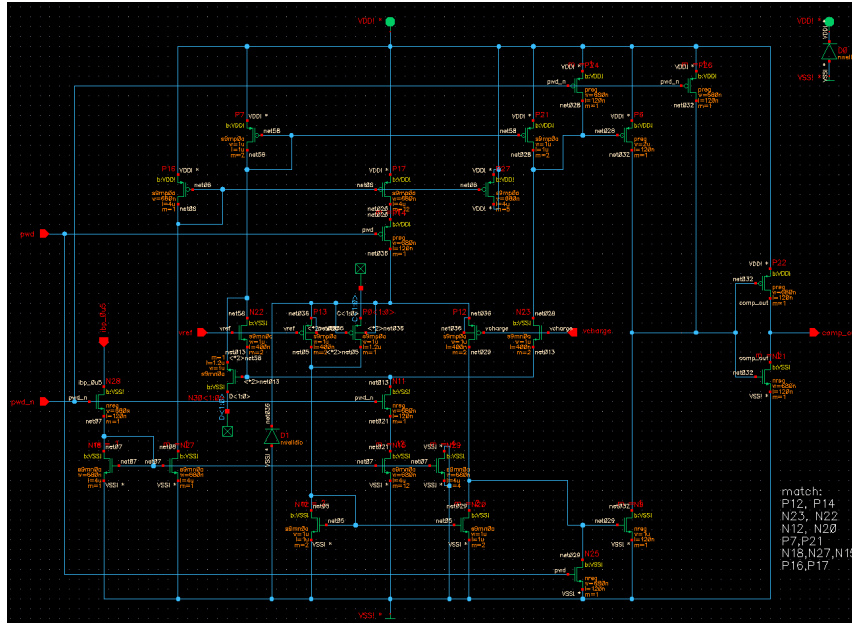


Figure 1.1: Example of an analog circuit schematic.

to enable precise sizing of circuit devices.

However, when it comes to the layout stage of analog circuit design, only a limited number of attempts have been made to streamline this process. But before delving into the main motivation behind this dissertation, let's give a broad overview of the layout of analog ICs.

1.1 Analog Layout Basics

Designing the layout of analog circuits is a crucial and complex task that requires significant expertise in order to be properly completed. As previously highlighted, analog circuits are particularly susceptible to noise and parasitics and are governed by strict topological requirements. These factors pose significant challenges for layout engineers, often necessitating multiple iterations to achieve an optimal layout that meets performance and design constraints.

The layout process can be broadly divided into two distinct yet closely entangled stages: *floorplanning* and *routing*. A visual representation of the results for both procedures can be found in Figure 1.2a and Figure 1.2b, respectively.

1.1.1 Floorplanning

The floorplanning procedure focuses on identifying the precise location of transistor, resistor and capacitor geometric shapes on the IC floorplan, with the primary objective of minimizing the area occupied while ensuring efficient routing.

While this goal may seem straightforward, achieving it is far from trivial. During floorplanning, several additional aspects have to be taken into account, such as soft modules handling and positional constraints satisfaction. Unlike hard modules, which have fixed width and height dimensions, soft modules require their shapes to be determined during

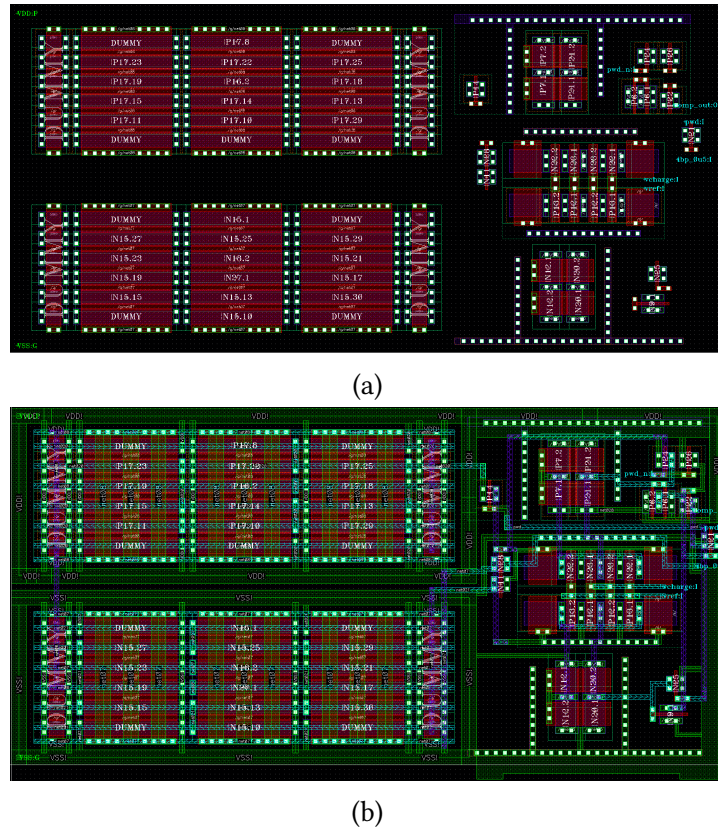


Figure 1.2: Examples of a plain (a) and routed (b) floorplan of an analog circuit.

the floorplanning process, contributing to increase the already high degree of complexity associated with analog floorplanning. Furthermore, as emphasized by Kahng in [81], chip manufacturers must often comply with strict specifications that result in fixed-outline floorplans. These constraints require the final floorplan to adhere to predefined aspect ratios (e.g., 1:1 or 1:2), leaving little room for deviation and further complicating the optimization process. From an electrical perspective, any automated floorplanning procedure must ensure aligned or symmetric placement of circuit devices [27] to minimize parasitic effects and guarantee matching whenever it is needed.

1.1.2 Routing

Once the circuit components have been placed, the routing process determines the paths for the circuit networks on the layout blueprint, ensuring that all required interconnections between pins, as defined in the schematic, are satisfied.

These routing paths must comply with several technology-dependent design rules to be considered successful. Among the most common rules are parallel run-length spacing, end-of-line spacing, and via spacing, all of which are verified using a design rule checking (DRC) tool. Again, symmetry between matching nets is essential to ensure proper electrical signal coupling. Other critical factors, such as electromigration and IR drop effects, must also be accounted for to maintain the electrical reliability and performance of the final circuit.

Long wires are typically discouraged due to their adverse impact on performances

and the increased risk of incurring into some of the aforementioned electrical issues. Consequently, efficient routing requires to minimize the total circuit wirelength. Unfortunately, this introduces additional complexity during the placement phase as well, as wirelength minimization depends on routing decisions that are not known apriori, i.e. during floorplanning. This interplay between placement and routing highlights just one aspect of the inherent complexity of the layout process, where multiple, often conflicting, objectives must be simultaneously optimized.

1.2 Motivation

Analog circuit layout, due to the reasons we just mentioned in Section 1.1, still requires nowadays years of expertise from engineers to produce solutions that are both constraint-compliant and highly efficient, hindering industries on the throughput that they aim to achieve. In fact, it is estimated that up to 40% of the total design time for analog circuits is spent on the layout phase, underscoring its role as a critical bottleneck in the overall design process. Addressing this inefficiency is essential to meet the growing demand for faster, more efficient design cycles in the semiconductor industry.

Unlike other fields that have readily embraced AI-based solutions to enhance automation and alleviate tedious tasks, the EDA community has historically exhibited a certain reluctance. This hesitation is, to some extent, rooted in the nature of the challenges faced in circuit design, where the complexity and specificity of problems often poses an obstacle to the direct application of general-purpose ML techniques. A notable example of this skepticism is the ongoing debate between two Google and Synopsys researchers [51, 119], which persisted for years. The discussion focuses on claims that a reinforcement learning (RL)-based solution for digital circuit placement from Google [126] was unfairly compared to more conventional stochastic-based approaches. Nevertheless, looking at what large language models (LLMs) have been capable to achieve in the very last few years in the natural language world, it is reasonable to consider that even a task as complex as this can be effectively addressed using advanced ML techniques.

Existing analog floorplanning automation methods mainly fall into three categories: procedural [39], template-based [142], or optimization-based methods [23, 41]. While these methods have primarily been confined to academic research, Pulsic [143] represents one of the few commercially available tools. However, its adoption remains scarce, and its impact on the broader field is still limited. The main reasons for such restricted application might be multiple. Optimization-based methods, which leverage metaheuristic techniques such as simulated annealing (SA) [88], genetic algorithms (GA) [70], and particle swarm optimization (PSO) [84], are capable of producing compact floorplans. Still, they often leave insufficient space for routing tracks, leading to layouts that are ultimately unusable. Furthermore, incorporating geometric constraints is complex and significantly increases the already lengthy optimization runtimes, particularly for large-scale or highly complex circuits. Template-based generators [122, 142] provide an alternative by utilizing fixed templates to determine optimal placements. However, similar to procedural approaches, they require extensive manual effort in creating and customizing appropriate templates, which goes in the opposite direction of a fully human-independent workflow automation.

For what concerns the routing, most of the advanced solutions in the literature either rely on pathfinding algorithms, typically constrained versions of A^* [47], obstacle-

avoiding rectilinear Steiner minimum tree (OARSMT) [106], integer linear programming (ILP) [174] or evolutionary algorithms (EA) [121]. While these methods have demonstrated potential, each comes with inherent drawbacks.

Sequential pathfinding algorithms, often employed for their high flexibility and optimality, introduce certain challenges due to their inherent sequential nature. Specifically, the routing of a single wire or net can interfere with the routing of subsequent ones, leading to the failure of a routing attempt. This issue becomes particularly problematic in analog circuits, which often involve a considerable number of nets to be routed. To address these challenges, non-systematic heuristics for net ordering, rip-up, and rerouting are employed to ensure valid and feasible routing solutions.

On the other hand, ILP-based routers are capable of routing all circuit networks simultaneously. However, this method suffers from severe scalability issues, as memory requirements increase significantly with the growing number of grid cells that must be handled for larger or more complex designs.

In contrast, EA-based routing engines are characterized by long runtimes and poor sampling efficiency, which limit their practicality for high-complexity circuits.

Lastly, procedural [62] and template-based [113] approaches have also been explored in the past. As observed in the floorplanning domain, these methods exhibit lower levels of automation compared to other techniques. This is primarily due to their reliance on manual intervention to define the absolute or relative locations of interconnects between cells. Moreover, layout changes are not well accepted, because the script that generated the routes may need to be re-coded or the templates themselves adapted. Despite these limitations, such methods offer useful ways to parametrize metal layers and layout features, providing a degree of flexibility for specific design needs.

Learning-based approaches to floorplanning and routing have a comparatively brief history in the research literature. The former problem, however, belongs to a family of challenges that has been extensively investigated within the ML domain: combinatorial optimization (CO) problems. More precisely, RL has gained significant attention for addressing this class of problems, as evidenced by recent surveys such as [123]. A primary advantage of employing such an approach, in contrast with traditional metaheuristic-based methods, lies in its capacity to leverage acquired knowledge from prior attempts to solve a single problem instance. This enables more efficient navigation of the solution space and facilitates a focused exploration of its most promising regions. Furthermore, given that floorplanning can be abstractly conceptualized as a sequential decision-making process, since it essentially consists in an iterative placement of various blocks onto a canvas, it lends itself well to being framed and solved effectively using RL.

Despite some initial efforts to automate routing with DL models [103], their practical adoption has been limited, mostly because of the lack of diverse, publicly accessible datasets required for effective training. Furthermore, while the routing problem can be conceptualized as a pathfinding task, allowing for the application of optimality-guaranteeing algorithms like A* for generating pin-to-pin connectivity, this methodology still presents inherent limitations. It is precisely in these areas that DL and RL solutions show significant promise. Specifically, they can introduce data-driven knowledge and adaptability for heuristic-dependent operations, such as net ordering, multi-pin network decomposition, and the selection and reordering of rip-up and reroute operations.

In this context, this thesis investigates learning-based automation for analog layout

with a primary focus on *constraint-aware floorplanning with routability-driven objectives*, complemented by the development of a first prototype of a pathfinding-based routing engine. The proposed approaches are assessed using quantitative metrics that capture both layout quality and automation efficiency, including half-perimeter wirelength (HPWL), area utilization, routing success rate, satisfaction of key analog constraints (symmetry, alignment), and the total time required to generate a complete layout. The overall objective is to achieve consistent improvements over conventional optimization-based baselines, such as metaheuristics and procedural generators, across these metrics, while meeting industrial quality standards and minimizing the amount of manual intervention required from layout engineers. Ideally, we aim to shrink the *human* effort of a process that typically spans several days, if not weeks, into a fully automated pipeline capable of producing layout candidates within seconds to minutes, depending on circuit complexity.

1.3 Contributions

This thesis initiates foundational work toward exploiting learning-based methods to streamline the analog ICs layout process, aiming ultimately for a full automation enabled by transfer and zero-shot capabilities. Multiple methodologies leveraging RL are presented for automated floorplanning. Furthermore, an OARSMT-based algorithm is devised to provide guidance during the routing phase. The complete pipeline is integrated into the Infineon ANAGEN procedural layout generation framework.

The novelty of this work includes the following contributions:

- We propose two RL-based automated floorplan generators for analog ICs that either collaborate with SA (*RL-SA*) or take inspiration from it to for the agent’s behavior and state representation (*Annealed-RL*). Target metrics, shared also with the following approaches, focus on area occupation and estimated wirelength optimization, while guaranteeing support for fundamental topological requirements such as device symmetry and alignment. In addition, they allow for the definition of specific aspect ratios of the final floorplan and variable shapes for each device. It is important to note that this last feature has not been previously explored in the existing literature.
- We train a relational graph convolutional neural network (R-GCN) [154] model to predict rewards for circuit placements. Once trained, it serves as an encoder of circuit, device, and geometric constraints for RL agent use to guarantee optimal generalization capabilities. The agent then combines graph and pixel-level (*R-GCN RL*) representations to comprehensively describe circuit and problem characteristics. The policy is designed to learn to select optimal shapes and positions for components while ensuring no overlaps and strict adherence to constraints such as symmetry and alignment.
- We validate our novel method on circuits of increasing complexity, demonstrating that our approaches outperform traditional metaheuristics in terms of area, proxy wirelength, speed, and cost metrics.
- An OARSMT method is devised for global routing, offering immediate guidance for ANAGEN to complete detailed inter-block connections. Moreover, an obstacle

avoiding A*-based routing algorithm is developed for defining precise pin-to-pin connectivity, abiding by fundamental DRC rules such as parallel run-length spacing, end-of-line spacing, and via spacing.

- We integrate floorplanning and global routing techniques into ANAGEN, providing engineers with an automated layout template pipeline. Industrial scenario tests demonstrate our method rivals the performance of manual layouts, leading to a 67.3% reduction in overall layout runtime and a 8.3% mean area reduction compared to manual layout.

In addition to the primary contributions, the following enhancements were developed to improve and extend the proposed methods:

- A Beam Search [114] enhancement of the *R-GCN RL* method with a new flexible objective function, allowing users to set trade-off weights between area and proxy wirelength metric optimization. The baseline *R-GCN RL* is improved by 5% up to 85% in all metrics, comparable to a fine-tuned approach, without the need of graphics processing units (GPUs).
- A routing-friendly version of *R-GCN RL* that incorporates device pin information. Based on the latter, precise spacing between blocks can be reserved, thereby accommodating all necessary routing wires in subsequent stages. To assess the effectiveness of this approach, we integrated it with the aforementioned A* algorithm, finally achieving a self-contained simulated environment for testing both floorplanning and routing. This novel setup yields significantly more routable floorplans, reducing the routing failure rate by an average of 73.4%.

1.4 Structure of the Thesis

The remainder of this dissertation is organized in three parts as follows:

- **Background:** This part delineates the foundational concepts that are crucial to understanding the topics and methodologies proposed in this thesis. In Chapter 2, a thorough overview of the details and challenges of the different steps that characterize the layout process is provided, along with a review of the automation techniques and works in the literature that have been proposed to streamline each phase. On the other hand, Chapter 3 describes the fundamental notions regarding the optimization techniques that have been employed in this work, with a specific focus on the background of stochastic, ML and RL algorithms. Finally, Chapter 4 discusses graph-based optimization algorithms as A* and Steiner Trees and how they can be used for solving circuit routing.
- **Machine Learning-Based Layout Automation:** This part represents the core of the thesis, where our contributions towards a learning-based paradigm for automated layout design are detailed. More specifically, Chapter 5, describes our first attempt to achieve such goal by presenting two novel floorplanning methodologies based on using RL alone and in cooperation with SA, along with an OARSMT-based

global routing technique. In Chapter 6, an enhanced floorplanning approach combining R-GCNs and RL is proposed, demonstrating how generalization capabilities and robustness can be increased by using more advanced methodologies. In Chapter 7, beam search is integrated into the previous chapter solution, aiming at improving the performance on unseen circuits at inference time. To conclude this part, Chapter 8 presents the latest version of our floorplanning algorithm, exploiting higher placement precision and a much increased routing-awareness. These improvements are validated using an internally developed A*-based routing system, demonstrating significant advancements over previous approaches.

- **Closing Remarks:** This part, contained within Chapter 9, reflects on the impact of the proposed methods and summarizes the overall findings of the dissertation and outlines the main limitations of such and avenues for future research.

1.5 Publications

- **Fast ML-driven Analog Circuit Layout using Reinforcement Learning and Steiner Trees**

Davide Basso, Luca Bortolussi, Mirjana Videnovic-Misic and Husni Habal.
20th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design - SMACD 2024 [8]

- **Effective Analog ICs Floorplanning with Relational Graph Neural Networks and Reinforcement Learning**

Davide Basso, Luca Bortolussi, Mirjana Videnovic-Misic and Husni Habal.
27th Design, Automation & Test in Europe Conference - DATE 2025 [10]

- **Enhancing Reinforcement Learning for the Floorplanning of Analog ICs with Beam Search**

Sandro Junior Della Rovere, Davide Basso, Luca Bortolussi, Mirjana Videnovic-Misic and Husni Habal.
21st International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design - SMACD 2025 [38]

- **Advancing Routing-Awareness in Analog ICs Floorplanning**

Davide Basso, Luca Bortolussi, Mirjana Videnovic-Misic and Husni Habal.
Workshop on Machine Learning for Systems - NeurIPS 2025 [9]

Part I

Background

Chapter 2

Analog Layout Synthesis

Layout, or physical design, is the process of generating geometric mask data for an IC using photolithography. This involves several key steps to produce a complete circuit layout. First, individual physical devices, such as metal-oxide-semiconductor (MOS) transistors, capacitors, and resistors, are laid out. Second, these components are placed on the chip canvas during the floorplanning stage. Finally, the routing process connects device terminals and circuit pins, ensuring electrical connectivity based on the circuit schematic. Once these steps are completed, electrical and geometric verification is performed to ensure the validity of the layout synthesis results.

In what follows, we provide a detailed description of each of the previously mentioned phases, emphasizing their influence on the layout outcome. Special attention is given to the unique characteristics and challenges that have impeded the advancement of fully automated solutions for the layout process and the automation attempts that have been explored in the literature.

2.1 Layout of Individual Devices

Analog circuits are composed of various electrical components, whose layout configurations are typically derived from the circuit schematic, an abstract graphical representation of an electrical circuit that consists of idealized electrical devices connected together by idealized wires. Devices can be split into smaller devices called *unit fingers*, which are then either connected in parallel or series. Transistors are characterized by parameters such as multiplicity, finger width, and finger length. Similarly, resistors are defined by their specified resistance, and capacitors are shaped based on their required capacitance. These parameters collectively determine the final shape and configuration of each component in the layout.

However, the circuit schematic does not provide all the information necessary for completing the layout. Certain aspects are left to the discretion of the layout engineer, allowing for a degree of freedom in determining additional parameters. These include decisions on gate, drain, and source routing metal layers, device orientation, pin positioning, and other geometric and connectivity details. This flexibility enables layouters to optimize the layout based on performance, while still being compliant to the layout rules imposed by the fabrication technology.

The layout of individual devices inherently gives rise to parasitic elements, such as

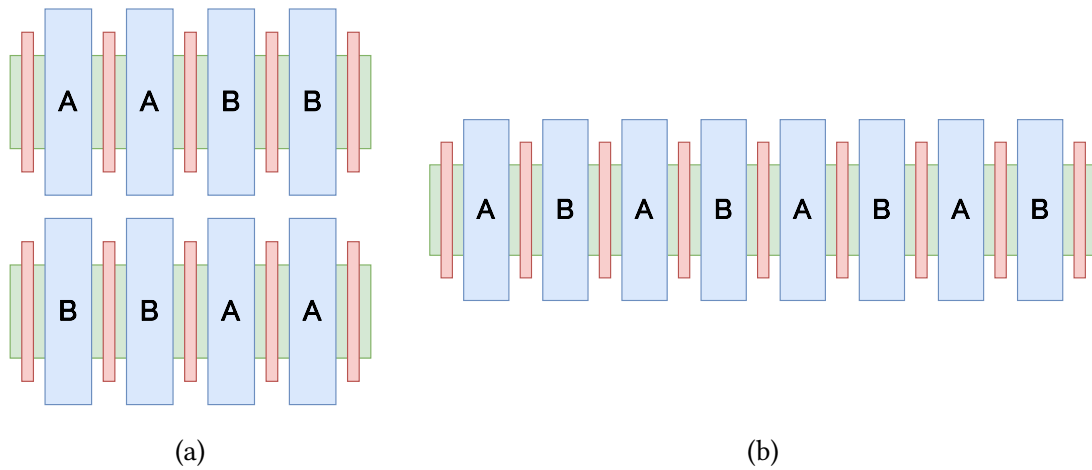


Figure 2.1: Example of common centroid (a) and interdigitated (b) placement patterns.

unwanted resistances, capacitances, and inductances, that can adversely affect circuit behavior. These parasitics originate from n-doped and p-doped regions within the silicon substrate. Layout engineers address these challenges through strategic device placement techniques (detailed in Section 2.2) and the utilization of parasitic extraction (PEX) tools. PEX tools, in particular, generate high-fidelity models crucial for detailed simulations, enabling engineers to be sure whether the extracted parasitics are within acceptable limits for the circuit's proper function.

Device parameters and behaviors can change due to processing and the environment in which those devices are placed. *Mismatch* is a local variation between two or more adjacent components on the same die that can degrade performance and in some cases can cause functional failure of silicon. This one differs from the global variation, which instead affects device parameters over a die or batch of die.

Matching [85, 139, 140] techniques are applied to a layout to reduce the mismatch by averaging the process effects across all matched devices. This procedure consists into placing two or more devices in a symmetric fashion to guarantee that they behave as identically as possible. This is typically required for structures such as differential pairs or current mirrors that ground their role on matching gate-to-source voltage and drain current, respectively. Among the many symmetric patterns that layout engineers can use to reduce mismatch, the most popular ones are the common centroid (CC) [83] and interdigitated one, which are depicted in Figure 2.1. With a CC pattern, the designer places all devices symmetrically around a central point. This arrangement minimizes the process random variations of each component along both the x and y axes, ensuring maximum matching. While highly effective, it typically leads to a slightly larger area occupation and more complex routing. On the other hand, an interdigitated pattern, which can be considered as a spacial case of CC, utilizes a single row to maximize the interleave between different devices, where the fingers of one transistor alternate with those of another.

To further reduce mismatch, another common practice is to include *dummy devices* at the devices array's boundaries. These dummies are non-functional transistors added solely to ensure the uniformity of poly geometries at the ends of the transistor array. This practice thereby minimizes variability in etch rates and provides effective isolation

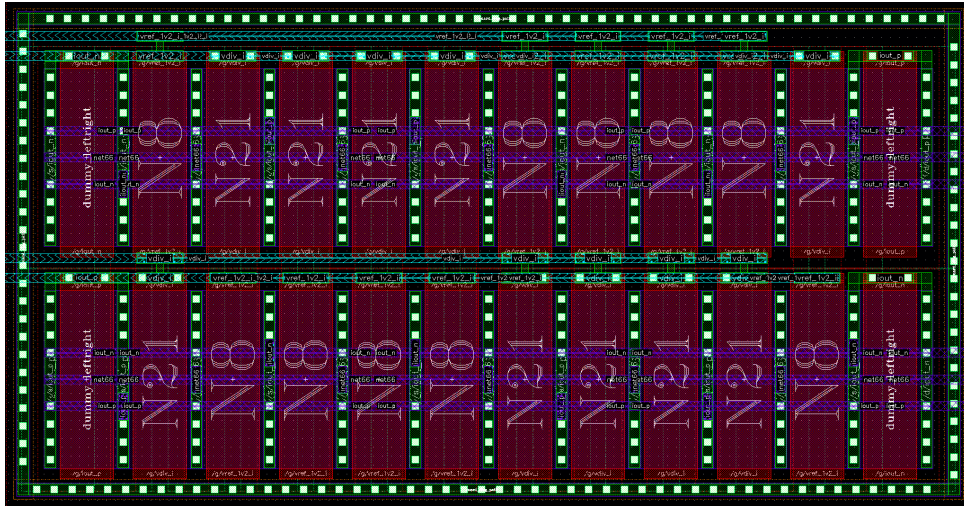


Figure 2.2: Differential pair with dummies on the side placed in CC fashion.

of the active devices from parasitic coupling with neighboring circuitry. An example of a CC placement for a differential pair, including side dummy devices, is given in Figure 2.2.

Layout-dependent effects (LDEs) have also become a critical issue in device layout due the significant decrease in size of transistors. The primary sources of LDEs can be categorized as: Well proximity effect (WPE), length of oxide diffusion (LOD), and oxide-to-oxide spacing effect (OSE) [135]. WPE arises from lateral non-uniformities in well doping, causing variations in threshold voltages and other electrical characteristics depending on a transistor's distance from the well edge. Conversely, LOD is a manufacturing effect which induces stress to the edges of the diffusion area during Shallow Trench Insulation (STI) formation. STI is a key fabrication technology that electrically isolates active device regions by etching trenches in the silicon and insulating them with dielectric material, typically silicon dioxide. This trench filling process and the resulting STI width, however, introduces mechanical stress into the silicon lattice surrounding the active areas. Both effects can be prevented by surrounding the circuit devices with dummies to ensure similar edge effects and enough distance from the guard ring, as depicted in Figure 2.3. OSE describes how the lateral spacing between diffusion areas can alter the magnitude of STI-induced stress. A way to address this issue is to increase STI width adjacent to active devices, which helps to relieve stress, thereby improving carrier mobility and overall circuit performance.

2.1.1 Device Layout Automation

With the advent of modern layout editors such as those provided by Cadence Design Systems or Synopsys, layout engineers can utilize parameterized cells (*Pcells*) to generate device layouts, as depicted in Figure 2.4. For example, when placing a MOS transistor Pcell, the layout engineer enters its width and length, typically via a graphical user interface (GUI). One of the key advantages of Pcells lies in their programmability, which allows their parameters to be defined programmatically, offering a seamless access point for automating layout generation through coding. However, as noted in [66, Chapter 3],

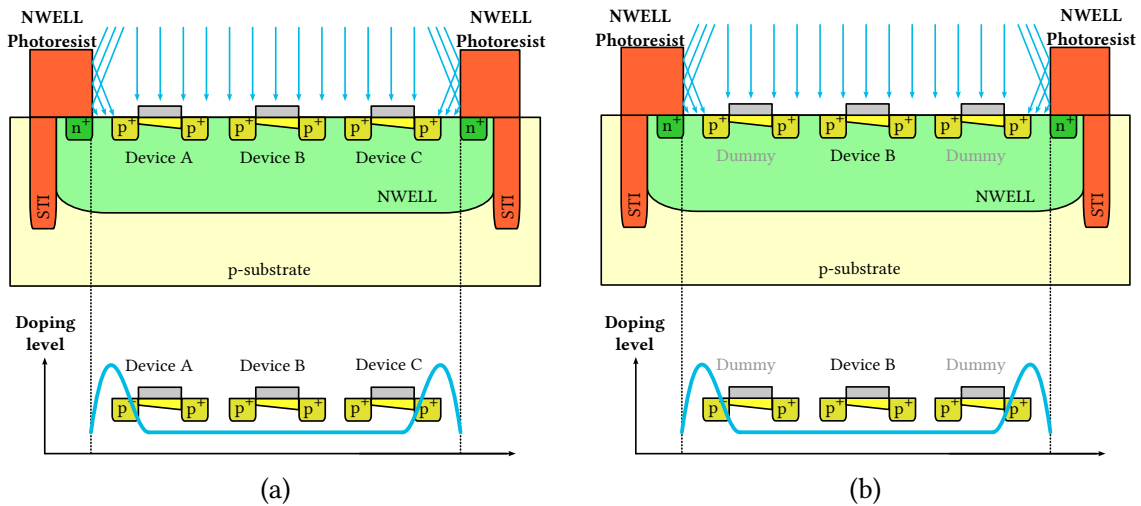


Figure 2.3: WPE effect in CMOS technology (a) and its prevention by adding dummies (b). Adapted from [161].

experienced layout designers may occasionally resort to custom layouts instead of Pcells. This is often due to challenges in synchronizing schematic parameters with layout parameters or concerns about performance degradation. Despite these limitations, a common best practice is to begin by instantiating a Pcell and then make adjustments or adopt custom layouts as needed.

2.2 Floorplanning

Floorplanning is the initial major step in physical design, where the layouted devices are placed on the chip canvas. Contrary to the digital floorplanning stage, the analog domain necessitates unique considerations and placement patterns to mitigate undesired electrical behaviors and meet stringent performance requirements.

The key objective for a layout engineer is typically the minimization of chip area occupation, as smaller layouts reduce silicon usage and, consequently, production costs. However, achieving this goal is constrained by circuit-specific geometric requirements, which may come from technology layout rules or customer specifications. These constraints often include fixed dimensions or aspect ratio requirements, also referred to as *fixed-outline* constraint.

By placing circuit components that are connected in the schematic closer together, the overall wirelength can be minimized, reducing parasitic effects and improving performance. However, this objective conflicts with the need to leave enough space between devices to ensure that all the necessary routing tracks can connect device terminals. Prioritizing compactness without considering routing requirements can lead to severe routing congestion, as highlighted in [158], creating further challenges for layout engineers.

Another critical aspect of analog floorplanning is the enforcement of symmetry and alignment constraints. As discussed in Section 2.1, matched devices require symmetric placement about an axis or a point to mitigate the effects of random and systematic variations, as well as changing operating conditions. For smaller devices with not so stringent accuracy requirements, a symmetry and proximity constraint alone can be

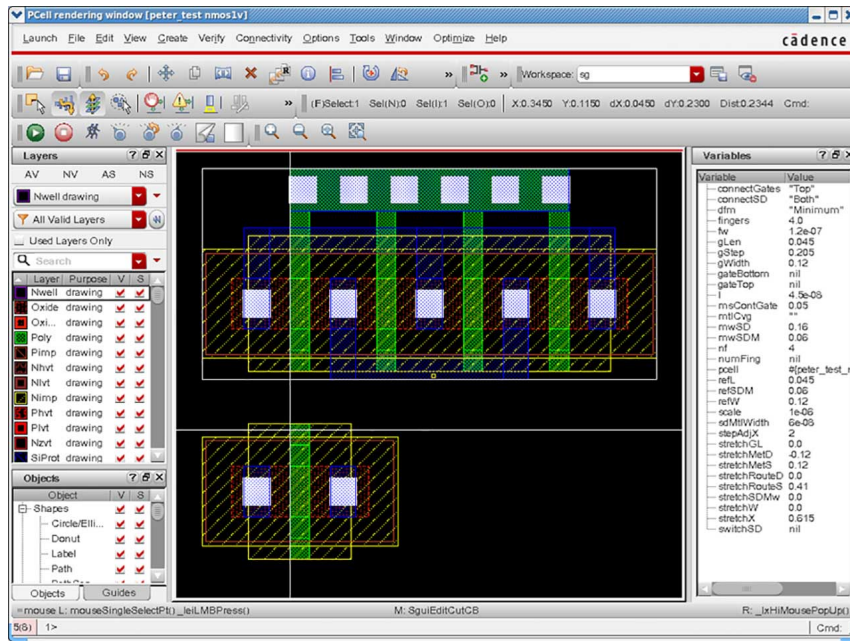


Figure 2.4: Example of a Pcell in Cadence Virtuoso, from [173].

sufficient to ensure proper circuit functionality, as reported in [183].

2.2.1 Circuit Floorplanning Automation

Floorplan Representation

The first step in automating the floorplanning procedure is to establish a suitable representation for describing the geometrical relationships between devices in the layout space.

A straightforward yet reasonable approach is to directly define the position of each circuit device using a pair of (x, y) planar coordinates. The primary benefit of this solution is the high degree of freedom it affords in arranging circuit components, which is particularly appealing given the intricate nature of analog IC layout and its numerous constraints. However, this approach also introduces a major drawback: the high dimensionality of the search space. This high dimensionality poses a challenge for deterministic, metaheuristic, and even learning-based solutions, as it can lead to increased computational time and difficulties in finding optimal solution.

Alternatively, topological representation models can be employed to describe the relative positions of blocks. These models inherently offer the significant advantage of preventing, by construction, overlaps between devices and limiting the arrangement possibilities to a finite set. Several such models exist, including the normalized polish expression [182], B*-tree [20], and sequence pair [128] (SP). As the SP representation will be utilized throughout this thesis, a brief description of the model is provided herein.

A SP, denoted as (Γ_+, Γ_-) , consists of two ordered sequences of module identifiers. These sequences encode the relative spatial arrangement of modules in the layout. The relationship between modules is determined as follows:

- **Left/Right Relation:** Module i is left to module j if i appears before j in both

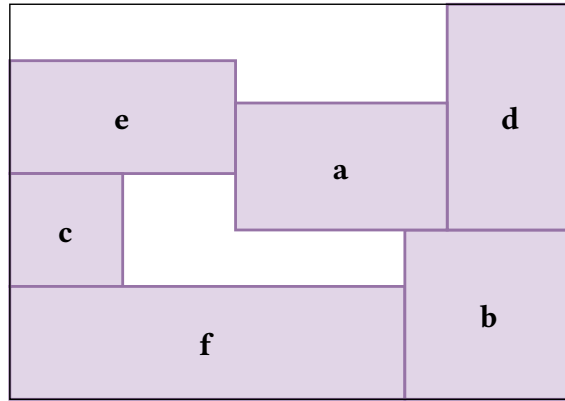


Figure 2.5: The compacted floorplan associated to $(\Gamma_+, \Gamma_-) = (ecadfb, fcbead)$.

Γ_+, Γ_- . Conversely, i is right to module j if j appears before i in both sequences.

- **Above/Below Relation:** Module i is below module j if j appears before i in Γ_+ and after i in Γ_- . Conversely, i is above j if i appears before j in Γ_+ and after j in Γ_- .

To illustrate, consider the SP $(\Gamma_+, \Gamma_-) = (ecadfb, fcbead)$ and its corresponding floorplan shown in Figure 2.5. For example, if we take module c , this one is positioned left to d because it comes before in both Γ_+, Γ_- and it is placed above f because it comes before in Γ_+ and after in Γ_- . Each unique permutation of Γ_+ and Γ_- defines a valid floorplan. For a layout with m modules, there are $m!$ possible permutations for each sequence, resulting in a solution space of size $\Theta(m!)^2$.

Automatic Constraints Extraction

It should be now clear that constraints play a fundamental role in floorplanning. Many of the floorplanning engines presented in the literature focus on complying with these constraints to replicate specific layout patterns. However, a key question arises: how are these constraints provided? Typically, they are annotated manually by layout design experts with extensive domain knowledge [153]. In contrast, *automatic constraints extraction* (ACE) techniques aim to identify constraints automatically for use in downstream tasks [194].

Traditional approaches for detecting matching constraints through circuit simulations have relied on sensitivity analysis [118]. Similarly, heuristics, such as those employed by Liu et al. [107], have explored symmetry constraint extraction based on mismatch analysis. Unfortunately, these approaches suffer from poor generalization capabilities and capability to grasp designers intent. To address this, more recent work by Liu et al. [108] has leveraged spectral and graph similarity methods to identify symmetric circuits at the system level. Furthermore, ML-assisted techniques employing graph neural networks (GNNs) have emerged in both supervised and unsupervised learning settings to estimate a similarity score for each pair of devices in the netlist [24, 27, 92].

Patel et al. [137] proposed a methodology that combines K-Means clustering [166] with GNNs to automatically recognize and label common functional blocks, such as current mirrors and differential pairs. This approach aims to assist layout engineers by

simplifying the process of defining appropriate constraints for these blocks. Building on this work, Pathak et al. [138] introduced a circuit recognition graph convolutional network (CR-GCN), which integrates rooted subgraphs [190] and a mixture of GNN models to identify more complex and diverse circuit structures. This extension enhances the capability of the original method to handle intricate hierarchies and variations in analog circuits.

Automatic Placement Approaches

Once the floorplan representation and constraints have been defined, the next step is the actual placement of circuit components on the layout blueprint. Over the years, various methodologies have been explored to achieve this goal, as discussed in Section 1.2.

Stochastic-based approaches have traditionally relied on SA as the main driver for floorplan generation. For instance, Balasa et al. in [6] utilized the SP model to encode the floorplan representation while supporting different types of symmetry. Similarly, Ma et al. [115] extended this approach to ensure the satisfaction of additional constraints, such as alignment, abutment, and CC layouts. The effectiveness of this SP and SA combination is further demonstrated in recent work by Li et al. in [101], where a customized GNN was trained to predict circuit performance and guide the SA search toward higher-performing regions in the search space. Additionally, SA has been integrated with other representations, such as the B*-tree, as shown in [28].

GA, which belong to the same family of stochastic approaches, have also been applied to floorplanning. Nakaya et al. [131] and Tseng et al. [175] combined GA with the SP model to generate and optimize floorplans, demonstrating its effectiveness as an alternative to SA.

ML has been reshaping various industries and opening new perspectives in analog EDA. A first attempt to automate the analog floorplanning procedure was proposed by Guerra et al. [55], where NNs were trained on a dataset of analog amplifiers containing thousands of placement solutions to directly predict (x, y) coordinates for each circuit component. This paper has been followed by several advancements coming from the same research group [56–58], culminating in the most advanced solution proposed by Gusmão et al. in [59], known as *AGraph2Seq*. In *AGraph2Seq*, an unsupervised attention encoder-decoder model was introduced, using R-GCNs to embed key topological constraints, such as symmetry, proximity, current flow. The embeddings were then decoded employing an attention-based Long-Short Term Memory (LSTM) [69] to generate circuit floorplans.

Building upon this methodology, Wang et al. [177] proposed a follow-up solution, *AIPlace*, which replaced the LSTM-based decoder with a multi-task neural network to predict the placed position of each device.

Despite these advancements, all these methods face some limitations, particularly the need for additional post-processing steps to validate and adjust the generated layouts. Issues such as device overlap and symmetry mismatches still require manual intervention, underscoring the challenges in achieving fully automated floorplanning.

To date, the only known attempt to employ RL for analog floorplanning has been proposed by Ahmadi et al. [2], where they trained an RL agent to place FinFET modules. After a first initialization phase, the agent has to decide how to move each module on a grid while minimizing symmetry and alignment errors, area and wirelength. This model, however, solely relies on a convolutional neural network (CNN) [97], omitting

circuit-specific metadata or device embedding that a GNN could provide. Additionally, the model’s assumption of fixed device shapes limits its flexibility to find placements that might ease routing or better conform to the floorplan design.

As a final note, although it extends beyond the scope of this chapter and thesis, which focus on analog IC floorplanning techniques, it is worth mentioning influential works in the digital domain that have significantly inspired the research presented in Chapter 6. The underlying concepts are in fact transferable to the analog domain. However, adapting these necessitates additional considerations to ensure the generated floorplans meet functional requirements and comply with the specific constraints outlined earlier in this section.

A notable work is the one from Mirhoseini et al. in *AlphaChip* [126], which introduced a novel approach to represent circuits as hypergraphs, where nodes correspond to modules and edges denote interconnections. In this framework, modules are placed onto a discrete grid by a RL agent. The agent leverages a GNN to generate expressive embeddings that enrich its state representation. The authors report that their method produces manufacturable digital chip layouts within 6 hours, achieving performance that matches or surpasses human experts and established tools across key metrics, including power, area, wirelength, and congestion. While the model was initially trained on a variety of chip designs, its generalization capability is further enhanced through fine-tuning via transfer learning. Eventually, Google’s latest generation of tensor processing units (TPUs) was developed using this algorithm without the need for manual refinement by human designers.

Lai et al. in *MaskPlace* [93] presented another RL approach that leverages pixel-level visual representations instead of traditional geometric encodings or more advanced graph representations. Chip modules are placed on a discrete 224×224 grid, with the use of three pixel-level masks to encode floorplan information. These masks are discussed further in Section 6.2.1. CNNs are employed to extract information about the current state of the RL agent from the placement grid and drive its action choices. A key feature of MaskPlace is that it guarantees no overlaps by construction, while AlphaPlace does not avoid it explicitly. Furthermore, MaskPlace considers a *dense reward function*, overcoming the sparsity of end of episodes reward of AlphaPlace, eventually enabling faster and more stable training.

Other relevant contributions from the digital IC domain are listed here after. *GoodFloorplan* [185] leverages GNNs and RL to generate efficient floorplans relying on the SP topological model representation. Amini et al. [4] instead replicate the same concept while using a corner block list (CBL) [71]. *ChiPFormer* [94] introduces a fully offline RL approach, leveraging a transformer-based decision model to learn global placement strategies directly from expert trajectories. *Miracle* [186] improves traditional RL techniques by incorporating a multi-action agent capable of predicting multiple relative positions between modules in a single decision step. Finally, *FlexPlanner* [192] extends RL-based methods to three-dimensional IC floorplanning, by modeling block positioning, aspect ratios, and cross-layer alignment using hybrid policies.

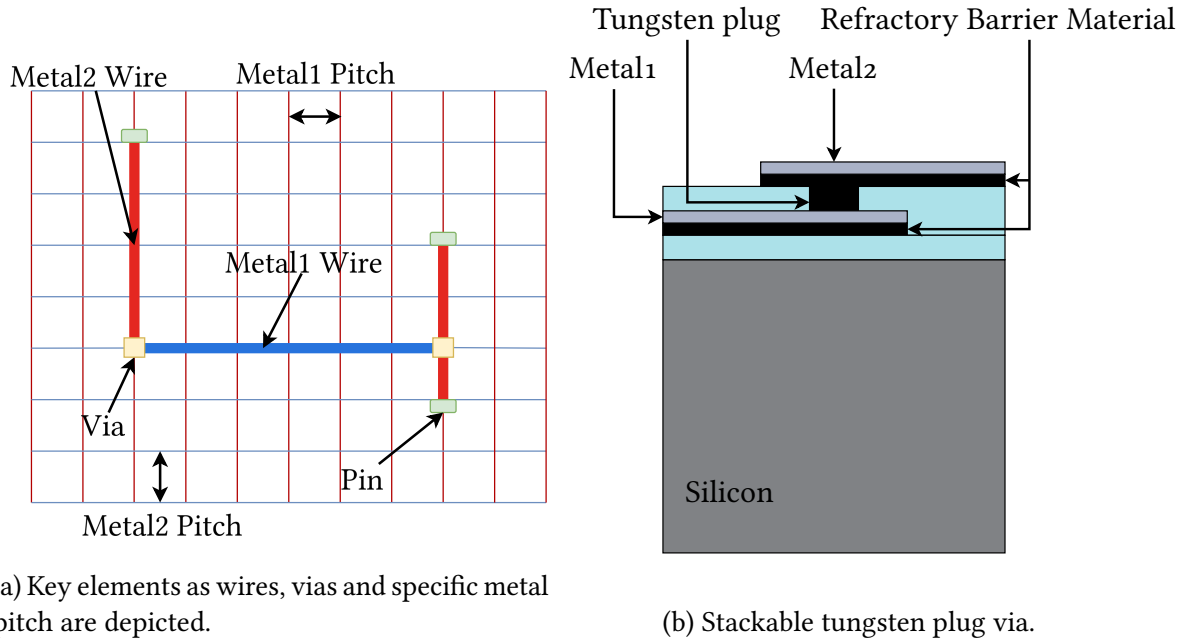


Figure 2.6: Examples of a routing grid (a) and a via (b).

2.3 Routing

After placement, the routing step determines the paths for the nets on a chip that connect different components or pins. Designs typically employ two or more metal layers to draw connectivity, and whenever a signal moves from one metal to the other, a via should be inserted. Routing wires are drawn on a grid with specified pitch, which equals to the distance from the center of one lead to the center of the next. Figures 2.6a and Figure 2.6b visually depict the aforementioned concepts.

Along the same lines of floorplanning, the key objective is to keep the length of the wires connecting all the components together as small as possible. Still, several aspects and technology dependent rules have to be taken into account.

Interconnect-level parasitics, including resistance, capacitance, and inductance, must be carefully considered as they can degrade circuit performance, particularly in the presence of long, parallel-running wires. To mitigate these effects, geometrical constraints are imposed by DRC rules. For instance, minimum spacing between wires and vias is required to limit coupling capacitance, while maximum wire length and minimum wire width are specified for each metal layer to reduce connection resistance and total load capacitance. Routing congestion, which happens when a routing grid exceeds its capacity due to too many routed wires, causing delays and inefficiencies in the routing process, also plays a crucial role in the success of a routing attempt [130].

Noise-sensitive signals also demand special attention during routing. Designers typically communicate information about these signals by annotating schematics, modifying signal names, or providing lists of affected signals. Proper handling of these signals is critical to maintaining circuit performance and reducing noise interference.

For symmetrically placed devices and differential signal paths, routing must ensure matched resistance and load capacitance, or, when possible, mirrored routing geometry. These measures are essential to maintaining signal integrity and minimizing mismatches

that could degrade the performance of differential circuits.

Additionally, routing must account for long-term reliability concerns, such as electromigration and *time-dependent dielectric breakdown* (TDDB). Electromigration refers to the movement of atoms caused by the flow of current, which can lead to open-circuits or short-circuit failures. TDDB, on the other hand, occurs when the dielectric material between the gate and the channel deteriorates over time due to high electric fields, eventually causing breakdowns. Both phenomena are exacerbated by high current densities and electric fields and require careful routing strategies to ensure the long-term reliability of the circuit.

Even though there are many challenges to face, some automation attempts have been explored in the past. We summarize them briefly in the following section.

2.3.1 Circuit Routing Automation

As analog ICs grow larger and incorporate increasing amounts of digital content, automated routing becomes increasingly attractive. In manual routing, a typical approach is channel routing [40], where signals are routed through dedicated channels placed between circuit blocks. However, automation of this methodology has not gained much traction, as it heavily relies on human expertise to define channel locations and configurations.

Modern automated routing methodologies aim to fully comply with DRC rules and geometrical constraints, as discussed earlier. Among the earliest approaches are Steiner tree (ST)-based methods, which are valued for their simplicity and effectiveness. In the context of analog layout, obstacle-avoiding rectilinear Steiner minimum trees (OARSMTs) have emerged as a critical solution to handle obstacle avoidance. Li et al. in [100] introduced a heuristic maze routing-based approach to solve the OARSMT problem, demonstrating its ability to handle large-scale problems and multi-pin nets. Liu et al. [105] improved upon this work with an $O(n \log(n))$ -time algorithm that guarantees theoretical optimality and better speed performance. They further extended their work in [106] by proposing a linear-space routing graph with efficient Steiner point candidates and adapting it for preferred routing directions across multiple metal layers. More recently, Chen et al. [30] proposed a RL-based approach for constructing OARSMTs, aiming to outperform traditional heuristics. However, most OARSMT studies focus on the methodology for constructing the tree rather than applying it directly to circuit routing, often overlooking critical design rules for routing.

EAs have also been applied to routing. Martins et al. [121] developed an EA-based routing engine in which an internal procedure cyclically evaluates each generation of solutions, optimizing all nets simultaneously without requiring rip-up, reroute, or net ordering procedures.

Another line of research explores ILP-based routers, such as the electromigration and parasitic-aware system proposed by Torabi et al. [174]. This method accounts for performance and mismatch constraints, making it suitable for analog circuits. However, as noted previously in Section 1.1.2, ILP-based approaches suffer from severe scalability issues, limiting their practicality for large designs.

Today, sequential pathfinding algorithms, particularly A^* , have become the de facto standard for state-of-the-art analog routing engines. Xu et al. presented MAGICAL [184], a fully automated analog IC layout automation system that employs A^* as backbone

of its routing engine. They subdivide the routing problem into two phases: global and detailed routing. During global routing, a coarser routing grid is generated and routing constraints are relaxed in order to provide guidance for the A* kernel during the detailed phase. The detailed routing step finalizes the interconnections between terminals and pins using the same A* kernel as before, assigning metal wire geometries to each net and attaining to the specified design rules. MAGICAL also incorporates handling for symmetric and self-symmetric nets.

Building on this, Zhu et al. [193] proposed an ML-guided routing paradigm that enhances the A* engine using a variational autoencoder (VAE) [86]. The VAE was trained to extract latent layout strategies from experienced engineers, guiding the A* algorithm toward more efficient solutions. Chen et al. in [22] further improved detailed routing by incorporating additional DRC rules, such as parallel run length, end-of-line spacing, minimum area, and minimum step rules, alongside electrical constraints including wider wires for critical nets and multi-cut vias for inter-layer connections. These improvements were achieved through pre-layout simulations, further enhancing routing reliability.

Zhang et al. [188, 189] also utilized A* as the core algorithm for detailed routing and extended prior solutions by introducing an ST-based wire-sizing procedure to prevent electromigration and IR drop effects, further improving long-term circuit reliability. Despite these advancements, all of the aforementioned approaches rely on a rip-up and reroute scheme to address violated nets, where the rip-up order and reroute decisions are typically deterministic.

As said before, RL has already been employed to improve routing automation. Chen et al. [25] presented an RL-guided methodology for detailed routing, where an RL agent was trained to guide the rip-up and reroute process by selecting suitable segment removal sequences, thereby enhancing the efficiency of the A* backbone. RL has also been applied for global routing automation. Liao et al. [103] developed an RL agent to solve global routing problems in a simulated environment, demonstrating performance competitive with, and in some cases surpassing, A* baselines. Furthermore, Qu et al. [144] introduced an asynchronous RL framework to optimize net ordering strategies, enabling the generation of high-performance routing sequences and producing overall higher-quality solutions.

Even though RL-based approaches have demonstrated significant potential, challenges remain in achieving fully automated routing solutions that can deal with such a large search and action space, while satisfying all DRC and performance constraints without extensive post-processing or manual intervention.

2.4 Post-Layout Verification

Post-layout verification marks the final stage of the physical design process, ensuring that the layout satisfies both geometrical and electrical requirements. This step typically includes DRC checking to enforce geometrical constraints, layout versus schematic (LVS) verification to confirm schematic consistency, and parasitic extraction both from floorplanning and routing to evaluate the impact of layout-dependent effects on circuit performance.

Commercial tool, such as Calibre [162], Cadence Spectre [16] and Synopsys PrimeSim [172] are widely used for post-layout simulations and provide highly effective solutions.

However, these workflows often rely on iterative feedback loops to address violations or performance issues, making them time-consuming and challenging to scale for large designs.

It's not a novelty to see post-layout verification inserted within the place and route pipeline [33, 101], ensuring compliance with design specifications straight during layout generation. However, given the iterative and prolonged nature of post-layout simulations that does not scale well with large designs either, this solution become prohibitively expensive in an iterative feedback loop [21, 180], further hindering the complete automation of the layout process.

An alternative direction involves using ML approaches to bypass these limitations by directly modeling layout effects and predicting post-layout simulation results. For instance, Martins et al. [120] introduced post-layout regressors built using linear and tree-based models, trained on reliable simulation data, to predict circuit performance metrics during placement optimization. This method eliminates the need for repeated parasitic extractions during optimization, significantly improving efficiency.

While promising, these emerging approaches cannot yet replace post-layout simulations, which remain indispensable for providing objective and precise circuit performance validation. Given their importance, research efforts should continue to focus on streamlining this process to reduce design iterations and improve efficiency.

Chapter 3

Optimization Techniques

Among the various techniques available, optimization-based approaches are the ones that have been used throughout this thesis in order to automate the floorplanning procedure. Section 3.1 explores the theoretical background of the stochastic methods used combined with learning-based techniques in the initial proposal for floorplanning automation, as well as those employed for baseline comparisons. Section 3.2 delves into the fundamentals of ML, highlighting the architectures and models that have been exploited. Particular emphasis is placed on the RL paradigm, which has been extensively leveraged as key actor in the proposed solutions.

3.1 Stochastic Methods

Problems of increasing academic and industrial interest are often characterized by high levels of uncertainty. This uncertainty can be modeled using random variables, for which a suitable objective function O or a set of constraints can be defined to find solutions. Based on whether O is defined over a continuous or discrete domain, an optimization problem can be classified as continuous or combinatorial. In the continuous case, O is a “smooth” function, often differentiable, allowing feasible if not optimal points to be reached by following the gradient. In contrast, combinatorial optimization is more challenging, as the variables are discrete, and a step in a certain direction does not necessarily guarantee improvement toward the optimum. Stochastic methods, which rely on probabilistic principles, are particularly effective for addressing these problems, especially when system noise, non-linear behaviors, or incomplete information make deterministic approaches unsuitable.

By definition, an algorithm is defined as *stochastic* when [167]:

- Random noise is present in the measurement of O ; and/or
- Random choices are made during the search process toward a solution.

The algorithms presented in the upcoming sections specifically rely on this second property. In certain cases, in fact, influence coming from random perturbations during the search process can be advantageous, as it helps to avoid early convergence to suboptimal solutions, increasing the chance to achieve a better result in the end, along with an increased speed toward a more optimized outcome.

When tackling a specific problem, O may be shaped such that the true *global* optimum is difficult to reach. This is often due to the presence of multiple *local* optima, which can cause the search process to prematurely converge to suboptimal solutions far from the best possible region. In general, there is no guarantee of finding a single optimum for an assigned problem either, since there might be multiple global solutions that an algorithm can converge to. In practice, a scientist or engineer might be satisfied with a solution that is close to any of the best result possible.

Despite all these challenges, as already stated, stochastic methods constitute a robust family of approaches that has been used for a wide variety of applications, including analog EDA. The main principles of the SA algorithm are described in Section 3.1.1, while a brief detour of other commonly employed population-based optimization approaches is provided in Section 3.1.2.

3.1.1 Simulated Annealing

Simulated annealing [88] is a metaheuristic global optimization technique that employs probabilistic perturbations to explore the solution space and search for the global optimum. The algorithm is inspired by the annealing process in solids, with its foundations rooted in statistical mechanics. In this analogy, the set of atomic positions $\{r_i\}$ of a material is weighted by the Boltzmann probability factor:

$$e^{\frac{-E(\{r_i\})}{k_B T}}, \quad (3.1)$$

where $E(\{r_i\})$ is the energy of the configuration, k_B is Boltzmann's constant, and T is temperature. During annealing, careful cooling is essential to ensure that the atomic structure reaches equilibrium at low temperatures, minimizing defects in the resulting crystal.

Along the same lines, SA adapts this concept to optimization problems using the Metropolis-Hastings algorithm [67, 125], to simulate the equilibrium of a collection of atoms at a given temperature. The energy quantity $E(\{r_i\})$ is replaced by the change in the cost function ΔC , which results from perturbing the current state solution s to generate a new one s' . Unlike purely greedy approaches, which only accept s' if it reduces the cost function, SA adopts a hill-climbing strategy that allows occasional acceptance of worse solutions to escape local optima. The probability of transitioning from s to s' is defined as:

$$p(s \rightarrow s') = \begin{cases} 1 & \text{if } \Delta C \leq 0, \\ e^{-\Delta C/T} & \text{otherwise.} \end{cases} \quad (3.2)$$

The acceptance of a new state is thus governed by a Boltzmann distribution, where the likelihood of accepting a solution depends on both the change in the cost function ΔC and the current temperature T . When $\Delta C \leq 0$, the new solution is always accepted, as it improves the objective. For $\Delta C > 0$, the solution may still be accepted with a probability that decreases as the magnitude of ΔC increases or as T decreases over the course of the algorithm.

The temperature T in SA follows a cooling schedule designed to balance exploration and exploitation. At high temperatures, the algorithm explores the solution space broadly, while at lower temperatures, it focuses on refining the search within the current region of

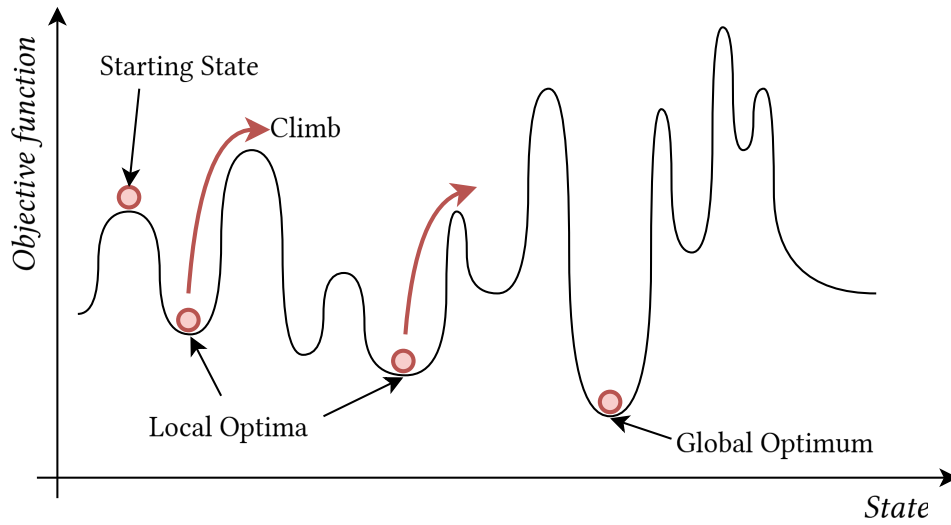


Figure 3.1: Simulated Annealing hill-climbing approach to escape local optima.

the solution space. This mechanism allows SA to escape local optima in the early stages and converge to a global optimum during later stages, as depicted in Figure 3.1. Several strategies such as linear, exponential or adaptive decay might be employed to define the cooling mechanism [132].

The search mechanism for a neighbor state is governed by either random perturbations or local search heuristics. A crucial aspect of this procedure is to devise a proper encoding to represent the solution, and specific techniques to manipulate it effectively. In the context of floorplanning automation, this involves defining a finite set of moves to perturb and lead to feasible configurations of the chosen topological representation model, as mentioned in Section 2.2. These moves typically include operations as permutations of nodes, identifiers or other structural elements of the representation.

SA has received widespread appreciation for its robustness, coming from its hill-climbing mechanism and temperature scheduling, as well as its inherent simplicity and adaptability to various scenarios. Nevertheless, a precise hyperparameter tuning must be carried out in order to reach the best performances possible. Moreover, SA's reliance on a single solution trajectory can restrict its effectiveness. In this regard, Section 3.1.2 details how population-based approaches, which simultaneously explore multiple solutions, can offer an enhanced alternative in certain cases.

3.1.2 Population-Based Approaches

Similar to SA, population-based techniques draw inspiration from biological processes, such as natural evolution and swarm intelligence. Key characteristics of these methods include the interaction between individuals within a population, cooperation, and the selection of the fittest individuals. For the purposes of this thesis, population-based techniques are used as baseline comparisons to evaluate the performance of the proposed methods in Chapter 5 and Chapter 6. Specifically, the following sections focus on genetic algorithms (GA) and particle swarm optimization (PSO).

Genetic Algorithms

GAs [70] fall in the family of the evolutionary algorithms (EAs), being their functionality tightly related to the natural selection process. Mimicking the Darwinian theory of evolution and the survival of the fittest, GAs maintain a population of candidate solutions that evolves over successive iterations. At each step of the optimization process, the population undergoes a selection procedure, followed by biologically inspired operators, such as crossover and mutation, to ensure sufficient diversity and exploration of the solution space.

A single solution is usually represented as a sequence of binary strings of fixed length, referred to as *chromosome*. It is important to distinguish between the genotype and phenotype representations of a solution, as certain operations are performed specifically on one or the other. The *genotype* is the encoded representation of a solution, such as binary strings or, in the floorplanning automation scenario, an SP. Genetic operators as mutation and crossover operations directly act at this level. The *phenotype*, on the other hand, represents the actual solution, e.g. the floorplan itself. It is on this specific object that the fitness is computed, in order to evaluate the quality of the proposed solution, and consequently guide the selection as well.

The procedure of GA involves the following steps:

- First, a population Y of n chromosomes is initialized randomly.
- Then, the fitness of each chromosome in Y is computed according to the defined evaluation function.
- Afterwards, a subset of chromosomes is selected for subsequent genetic operations. Several selection techniques can be employed, with *roulette wheel selection* and *tournament selection* being two of the most relevant ones [78]. The first defines the probability of an individual to be selected as proportional to its fitness. The latter instead extracts t individuals, where t is the tournament size, according to their fitness values from the population, and selects the one with the best fitness to be part of the next generation. This process is repeated with reinsertion until the desired number of individuals for the next generation is achieved. The choice of t directly influences the selection pressure, which determines the quality of individuals in the subsequent generation. Larger values of t increase selection pressure, favoring higher-fitness individuals but potentially reducing diversity in the population.
- After the *mating pool* is generated, pairs of individuals are selected as parents based on their fitness. The *crossover* operator swaps portions of the genotype from the two parents, with a certain probability, leading to a new offspring. Depending on the specific implementation, crossover can be performed using single-point, multi-point, or uniform techniques [90].
- Each offspring might be affected by a *mutation*, i.e. a random perturbation of the phenotype, following a certain small probability. One common example is the displacement mutation operator, which rearranges a substring of an individual by displacing it to a different position within itself [78].

- Once a stopping criterion is reached, typically upon achievement of a fixed number of generations, the best individual is selected and provided as output of the GA procedure.

A key advantage of GAs is their adaptability to multi-objective optimization, leading to multi-objective GA (MOGA). These algorithms generate a Pareto front of optimal solutions, allowing decision-makers to select a solution based on the relevance and preference of specific objectives. Additionally, GAs are well suited to be parallelized, given the independent nature of the fitness evaluation and the entire offspring generation cycle.

Unfortunately, one of the main drawbacks of using GAs is their high computational cost. This is influenced by several factors, including the expense of evaluating the fitness function, the number of iterations required, and the size of the initial population. Furthermore, careful parameter tuning is necessary to prevent premature convergence and to maintain sufficient diversity within the population. Finally, the algorithm outcome is highly influenced by the choices made for the mutation and crossover operators, meaning that a precise selection of these procedures must be operated as well.

Particle Swarm Optimization

PSO [84] has roots in both EAs and bird flocking, and fish schooling. Despite being first intended for simulation of social behavior, this method has been extended to perform optimization with high success. Leveraging the concept of swarm intelligence [19], for which collective behavior of an organized group of animals should strive for its survival, in the same way the population, in this case *swarm*, of individuals, i.e. *particles*, present in PSO collaborates together to reach the optimum of the defined objective function. The swarm is composed of n particles, where $x_i(t)$ and v_i^t represent the position and velocity of the i th particle of the swarm at time t , respectively. As for GAs, every particle is evaluated through a fitness function. At every step of the optimization procedure, the position of each particle is updated as $x_i(t+1) = x_i(t) + v_i(t+1)$. Their movement is driven by three crucial factors that influence the velocity update:

- *Inertia*, which discourages the particle to change direction immediately by using the inertia weight $w_{\text{inertia}} \in \mathbb{R}^+$.
- *Social attraction*, which influences the particle velocity to follow the best position g found so far in the swarm, i.e. the global best, by using the $c_{\text{soc}} \in \mathbb{R}^+$ social factor.
- *Cognitive attraction*, which regulates the particle behavior based on the best position b_i found by the current particle, i.e. the local best, by using the $c_{\text{cog}} \in \mathbb{R}^+$ cognitive factor.

Summing up, the velocity of a particle is updated as follows:

$$v_i(t+1) = w_{\text{inertia}} \cdot v_i(t) + c_{\text{soc}} \cdot r_1 \otimes (g - x_i(t)) + c_{\text{cog}} \cdot r_2 \otimes (b_i - x_i(t)), \quad (3.3)$$

being r_1, r_2 random vectors sampled from an uniform distribution $U(0, 1)$ and $\otimes$ the Hadamard product. After every position update, both p_i and g are updated, eventually returning the final global optimum once the termination criterion is met.

Analogously to GA, PSO has proven to be a highly efficient methodology for solving multi-objective problems. It can be easily parallelized, enabling faster convergence toward global optima with relatively few parameters to tune [48]. However, PSO is prone to premature convergence, especially in high-dimensional spaces, and its performance is sensitive to the initialization of control parameters, which can be challenging to configure effectively.

As observed, both GA and PSO share similar limitations. A common challenge across these stochastic methods, including SA, is the inability to leverage past experience, since they always begin the optimization procedure from scratch. This hinders their effectiveness in avoiding suboptimal regions of the search space and guaranteeing better robustness to initial conditions. ML-based techniques emerge as a promising alternative to address these issues, with the goal of improving exploration and adaptability in general.

3.2 Machine Learning

We live in a world defined by an immense and ever-growing volume of data. Failing to harness this invaluable resource and uncover the hidden insights it contains would represent a significant missed opportunity. Machine learning, a branch of artificial intelligence, seeks to extract knowledge by learning from data or past experience, in such a way a suitable mathematical model can be defined, without explicitly programming it [15].

Data play a central role in ensuring the effectiveness of a ML pipeline. The quality of the data and their source can significantly influence the performance and outcomes of the selected algorithm. More specifically, a dataset is a collection of numerical or categorical information used by a ML system to learn and solve predictive tasks. Datasets can be gathered systematically from existing examples, such as images or tables, or synthetically generated to ensure diversity and thoroughness.

Once the dataset is prepared and the ML model selected, the data are typically split into two portions: a *training* set and a *test* set. The training set is used to let the ML model learn the underlying statistical patterns, which it will leverage to make predictions during inference. The test set, on the other hand, evaluates the performance of the trained model using task-specific metrics that assess both the quality of its predictions and its generalization capabilities on unseen data. Talking about generalization, a common challenge in ML is the issue of *overfitting* [129]. This one occurs when the model learns not only the statistical properties in the training data but also its noise and variations. As a result, the extracted mathematical model becomes overly specific to the training data and fails to perform well on new ones. Conversely, when the model is unable to capture the data essential patterns, the phenomenon of *underfitting* occurs, resulting in poor predictive performance. In some cases, a third split, called the *validation* set, is introduced between the training and test sets. The validation set serves as a testbench for evaluating the model's performance during training. If the results are unsatisfactory, the model can be retrained with adjusted *hyperparameters* to improve its performance. Hyperparameters are parameters that govern either the training algorithm or any configurable part of the ML model, and they must be carefully defined and tuned to optimize the model's effectiveness. The primary goal of the training process is to find an optimal set of model parameters by minimizing a specific *loss* function. This loss function serves the same

purpose as the objective function introduced in Section 3.1.

ML can be broadly subdivided in different paradigms based on the properties of the input dataset and the training procedure.

- **Supervised learning:** In supervised learning, the model is trained on a dataset consisting of paired inputs, namely the feature and their corresponding label. Features can represent a wide variety of objects, spanning from numerical values, vectors, images to graphs. Labels, which are typically human-generated, can be dual, and based on their typology, we can distinguish two different families of problems. *Regression* problems occur if the label is a real-valued scalar. Conversely, if the label is categorical the problem is known as *classification*. The training in a supervised setting aims at minimizing the difference between the model's predicted values and the actual label associated with the input data.
- **Unsupervised learning:** In contrast, in unsupervised learning, the dataset fed to the algorithms consists only of features. The main goal of this paradigm is to group together data points based on similarities and structures that the model can extract from such unlabeled information. Applications of unsupervised learning involve clustering, visualization, dimensionality reduction, and anomaly detection. In this scenario, the loss function is designed to optimize the model's ability to encode meaningful patterns and structures from the unlabeled data.
- **Reinforcement learning:** RL differs from the previous paradigms. Here, the data is generated through the interaction of an agent with an environment. Based on the consequences of its actions within the environment, the agent receives a scalar reward, indicating the impact of its actions. The ultimate goal of RL is to maximize the cumulative reward over time, enabling the agent to perform its assigned tasks more effectively. RL has been successfully applied in fields such as control theory, robotics, and EDA. Further details on this paradigm are discussed in Section 3.2.2.

Several traditional ML methods, such as linear regression, logistic regression [64], decision trees [117], support vector machines (SVMs) [37], k-means [79] and principal component analysis (PCA) [116] among the others, have proven to be effective in solving regression, classification, and clustering problems. Artificial neural networks (ANNs), another member of this family of techniques, were first presented in 1951. However, it was only after Rosenbaltt's work on the perceptron [148] and the backpropagation mechanism proposed by Rumelhart et al. in [151] that they received the attention of the entire ML community. Inspired by biological information processing systems, ANNs are mathematical representations of interconnected processing units called artificial *neurons*. Like synapses in the brain, connections between neurons transmit signals, which are modified by weights that are iteratively adjusted during the training process. Signals are passed to subsequent neurons only if they exceed a certain threshold, determined by an *activation function*. Simple ANNs are typically organized into networks with an *input* layer that receives the data, and an *output* layer generates the final predictions [15, 77].

These classical ML algorithms are very effective, especially when working with well-structured data. However, when the ML field started to tackle problems of increasing complexity, both considering the type of input objects and their size, the need of more sophisticated solutions became more and more impelling. Tasks like speech and object

recognition in fact cannot be easily treated using the aforementioned methodologies. Deep learning was designed to exactly address such limitations.

3.2.1 Deep Learning

The key characteristic that distinguishes DL from traditional ML methods, especially from basic NNs, is the significant increase in the number of layers and neurons, resulting in deep neural networks (DNNs). This deeply nested architecture enables DNNs to learn highly complex functions in high-dimensional spaces [52]. By leveraging abundant data and a well-designed training schedule, DNNs can derive robust mathematical models capable of capturing intricate patterns in the data by automatically extracting the most important features that are needed for the corresponding learning task.

A DNN architecture aims to approximate a target function f^* by defining a mapping f such that:

$$y = f(\mathbf{x}; \theta), \quad (3.4)$$

where θ represents the set of trainable parameters. The training process involves learning the values of θ that yield the best approximation of f^* . NNs are referred to as networks because they can be described as a directed acyclic graph (DAG) $G(V, E)$, where nodes $v_i \in V$ correspond to neurons and edges $e_{ij} \in E$ represent the connections between neurons. The structure of these connections determines the layers of the network, and the output of each neuron depends on the weights and biases of its incoming edges. When all the neurons of one layer are connected to all the neurons of the following one, the DNN is called fully connected or a multilayer perceptron (MLP). For the i th neuron in the l th layer, the output $f_i^{(l)}$ is computed as follows:

$$f_i^{(l)}(\mathbf{x}) = \begin{cases} \sum_{j=1}^d w_{ij}^{(l)} \cdot x_j + b_i^{(l)} & \text{if } l = 1 \\ \sum_{j=1}^{n_{l-1}} w_{ij}^{(l)} \cdot \sigma^{(l)}(f_j^{(l-1)}(\mathbf{x})) + b_i^{(l)} & \text{if } l = 2, \dots, L + 1 \end{cases} \quad (3.5)$$

where d is the dimension of the input vector $\mathbf{x}$, n_l represents the total number of neurons in the l th layer, and L is the total number of hidden layers. The term $\sigma^{(l)} : \mathbb{R} \rightarrow \mathbb{R}$ denotes a nonlinear activation function, applied to the outputs of the previous layer to introduce nonlinearity. As described in Equation 3.5, the computation proceeds via *forward propagation*, moving layer by layer from the input to the output. The final output of the network is computed as:

$$f(\mathbf{x}; \theta) = f^{(L+1)}(\mathbf{x}), \quad (3.6)$$

where $f^{(L+1)}(\mathbf{x})$ corresponds to the output layer of the network. In Figure 3.2, a visual representation of an MLP is shown.

One of the key reasons for the widespread usage of MLPs and DNNs in general is their inherent modularity, which allows layers to be trained in parallel to optimize performance. As mentioned earlier, the set of trainable parameters θ includes the weights $\mathbf{W}$ and biases $\mathbf{b}$. By minimizing the loss function $\mathcal{L}(\mathbf{x}, \theta)$ during training and properly adjusting θ , the network can approximate the target function f^* . Indeed, $\mathcal{L}(\mathbf{x}, \theta)$ is typically a convex function of the difference between the observed and predicted values of the model. Gradient-based optimization techniques, such as gradient descent (GD) [150], are

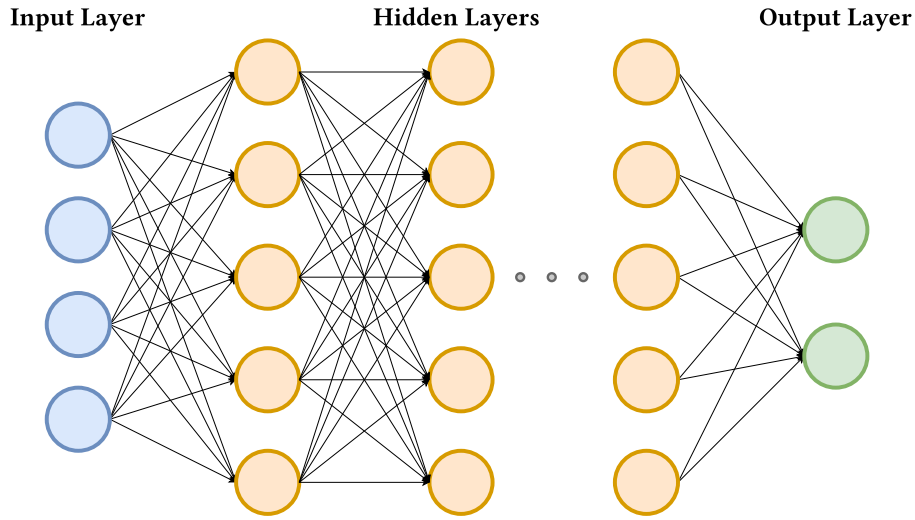


Figure 3.2: Architecture of an MLP with multiple hidden layers. Every neuron executes the function defined in Equation 3.5.

employed for iteratively updating θ , by leveraging the differentiability of σ . The update rule for θ at every iteration t is given by:

$$\theta_{t+1} = \theta_t - \eta_t \nabla \mathcal{L}(\theta_t), \quad (3.7)$$

where η_t denotes the *learning rate*, a hyperparameter that controls the magnitude of parameter updates at each step. The learning rate plays a crucial role since if η_t is too big, the algorithm might overlook local minima regions of the loss landscape. On the other hand, excessively small values can result in slow convergence.

While standard GD computes the gradient over the entire dataset, this approach does not scale well with the size of the input dataset and the noise inherently present in it. To address these issues, stochastic gradient descent (SGD) [146] has become the de facto standard for optimizing DNNs. By computing the gradient of the loss at each step on a random subset of the training points, known as *mini-batch*, this approach exploits a more reliable and faster convergence. Recalling the modularity of an MLP, partial derivatives of the loss in the l th layer w.r.t. weight w_{ij} can be computed. Using the backpropagation algorithm, these updates are propagated backward from the output layer to the input layer by leveraging the chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \frac{\partial \mathcal{L}}{\partial f_i^{(l)}} \cdot \frac{\partial f_i^{(l)}}{\partial f_j^{(l-1)}} \cdot \frac{\partial f_j^{(l-1)}}{\partial w_{ij}^{(l)}}. \quad (3.8)$$

Training a neural network involves alternating forward and backward passes. Once all training samples have completed a forward and backward pass, an *epoch* of training is completed. These computations, including gradient updates and loss evaluations, are typically managed by an optimizer [60].

Due to the deep and complex architecture of DNNs, they are prone to overfitting, particularly when trained on a limited amount of data. To mitigate this problem, regularization techniques are applied during the training phase. Among the various methods available [133], one widely used approach is *dropout*. Dropout randomly sets the output

of certain neurons to zero with a predefined probability during training. Doing so, an ensemble of smaller networks targeting to solve the same task is generated, whose results are averaged during inference.

DL models owe their success not only to the usage of wider and deeper networks, but also to the development of advanced operations between neurons. In MLPs, every neuron is fully connected to those in the subsequent layer. While effective for many tasks, this design becomes computationally expensive for high-dimensional data, such as images or sequences, due to the large number of parameters. For this reason, introducing an inductive bias into the model can make it more tailored to the given task and more efficient to train [11, 54]. For example, when processing data which shows a grid-like topology, e.g. images, exploiting local relationships between neighboring cells is often more important than considering distant ones. Convolutional neural networks (CNNs) are precisely devised to leverage such inductive bias by using convolution and pooling operations. Similarly, for graph-structured data, leveraging the inherent neighborhood relationships between nodes, as well as the symmetries and equivariance of the graph structure, can significantly improve performance. Graph neural networks (GNNs) are designed to natively handle these properties. In the following sections, we will explore the details of each of the aforementioned methods, since they will be employed in Chapter 6.

For the sake of completion, another important DL model for dealing with the sequential nature of its input data are the Recurrent neural networks (RNNs) [156], which have been widely applied to time-series, text, and speech. RNNs leverage recurrent connections between neurons at successive time steps, enabling the network to learn from past inputs and incorporate that information into its current processing.

Convolutional Neural Networks

CNNs [98] are a specific type of DNN inspired by the receptive field mechanism in biology, i.e. the specific region of sensory space (in the case of sight, a portion of the visual field) that, when stimulated, can affect the activity of a neuron.

The key component of CNNs is the *convolutional layer*, which extracts features from local regions of the input data. A convolutional layer consists of multiple convolution kernels, each acting as a neuron. These kernels have three dimensions: width, height, and depth. The size of the kernel, given by the product between width and length, determines the size of the neuron's receptive field, i.e. the number of cells that will be used from the previous layer on which it will be performed a convolution. The depth of the kernel corresponds to the number of feature maps output by the layer. Increasing the number of *channels*, i.e. depth, enhances the network's ability to extract complex features but comes at the cost of higher computational overhead. The convolution operation involves sliding kernel (or filter) over the input image, performing element-wise multiplication between the kernel and the corresponding image region, and then summing the results to produce an output known as the feature map. Mathematically, it is defined as:

$$O_{\text{map}}(i, j) = \sigma \left(\sum_{m=0}^{k_H-1} \sum_{n=0}^{k_W-1} K(m, n) \cdot I(i \cdot s + m, j \cdot s + n) + b \right), \quad (3.9)$$

where I is the input matrix of size $H \times W$, K is the convolution kernel of size $k_W \times k_H$, O_{map} is the output feature map, i, j represent the rows and columns of the output, m, n

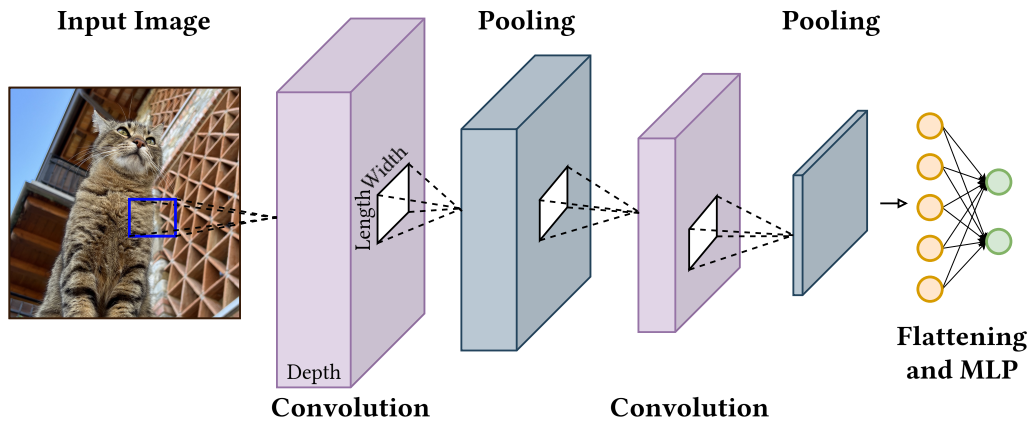


Figure 3.3: Visual representation of a CNN architecture. The different convolutional and pooling layers are highlighted, along with the kernel width, length and depth (# of channels). A final MLP is placed as example for using learned image representation for subsequent tasks, e.g. classification.

are the kernel indices, b is a bias term, s is the *stride*, i.e. the step size by which the kernel slides over the input from left to right and top to bottom, starting from the top left of the input matrix, and σ an activation function.

To preserve the spatial dimensions of the data, *padding* is often applied to the input. Padding involves adding rows and columns of pixels to the edges of the input matrix, typically using zero-padding or replicating the boundary pixels. This prevents the feature map from shrinking as the convolutional layers stack deeper into the network.

The feature map represents the extracted features of the input data. By stacking multiple convolutional layers, CNNs can detect higher-order and more abstract features. While deeper networks often increase computational costs, CNNs mitigate this through weight sharing. Indeed, since the same set of weights across an entire convolutional layer is shared (because the same kernel slides over the entire input matrix), this permits an efficient local feature extraction and significantly reduces the number of trainable parameters compared to MLPs or fully connected networks.

Another important characteristic of CNNs is their downsampling capability, which enables the extraction of informative low dimensional embeddings from high dimensional data. This is achieved by progressively reducing the kernel size in subsequent convolutional layers and employing *pooling* techniques. Pooling layers operate on non-overlapping regions of the input feature map and generalize these regions by outputting a single value, typically the maximum or average value. Combined with classical regularization methods, such as batch normalization and dropout, pooling contributes to prevent the overfitting phenomenon. To summarize, Figure 3.3 presents a visual overview of all the mentioned CNN's components.

CNNs have proven to be highly successful in solving several tasks related to the computer vision field [191], such as image classification, segmentation, medical image analysis, and object detection. Among the most popular architectures we can name AlexNet [91], ResNet [68] and U-Net [147]. Some of these models can be used as policies of RL algorithms to enhance the agent's capability to extract meaningful features from the state representation. More on this aspect will be described in Section 3.2.2.

Graph Neural Networks

GNNs were first introduced by Gori et al. [53] as a framework to enable NNs to natively operate on graph-structured data. The development of such architectures is motivated by the fact that many real-world entities, such as images, text, social networks, and molecules, can naturally be represented as graphs [152]. Before delving into the details of GNNs, we first summarize the fundamental concepts necessary for understanding graph structures.

A graph is described through a tuple $G = (V, E)$, with V indicating the set of n nodes and E the set of m edges. $\mathcal{N}(v)$ represents the neighborhood of node v s.t. $\mathcal{N}(v) = \{u \in V | (u, v) \in E\}$. An adjacency matrix $\mathbf{A}$ is a commonly used graph representation, defined as $\mathbf{A}_{u,v} = 1$ if there is an edge $e_{u,v}$ between node u and node v , and $\mathbf{A}_{u,v} = 0$ otherwise. Node features $\mathbf{X}$ for nodes $v_i \in V$ and edge features $\mathbf{X}^e$ for edges $e_i \in E$ have dimensions $\mathbb{R}^{n \times p}$ and $\mathbb{R}^{m \times q}$, respectively, with p and q as feature vector sizes. Graphs are categorized into different classes based on the properties of edges and nodes. A graph is defined to be *directed* if the direction of the edges matters; otherwise, the graph is defined as *undirected*. In the case the edges are assigned with a certain value, the graph is said to be *weighted*, turning $\mathbf{A} \in \mathbb{R}$. Furthermore, if a graph contains multiple types of nodes or edges, we call it *heterogeneous*; if all nodes and edges are of one type, the graph is instead *homogeneous* [61, 112].

GNNs main goal is to learn a continuous representation vector, called *embedding*, per node in the input graph. Based on these learned embeddings, it is possible to use them for solving different tasks: graph-level, e.g. to predict the property of the entire graph; node-level, e.g. to predict the role or identity of each node in the graph; edge-level, e.g. to predict which relationship an edge stands for or the presence of such.

Node embeddings are obtained using a message passing process [50] involving the exchange of vector-based messages among adjacent nodes over a set number of iterations. At each iterative step t , a node $v \in V$ aggregates information from its direct neighbors $\mathcal{N}(v)$, progressively enriching its own state $h_v^{(t)}$ with the neighborhood context message $m_v^{(t)}$, as defined by the following equations:

$$m_v^{(t+1)} = \sum_{u \in \mathcal{N}(v)} M_t(h_v^{(t)}, h_u^{(t)}) \quad \text{and} \quad h_v^{(t+1)} = U_t(h_v^{(t)}, m_v^{(t+1)}), \quad (3.10)$$

where M_t and U_t are the message and update functions, respectively. These are arbitrary, non-linear, and differentiable functions. After the iterative message passing process, a readout function, typically implemented using a NN, is applied to produce the final representation. This vectorized output can correspond to a node, edge, or the entire graph, depending on the downstream task.

The most basic version of GNNs are the graph convolutional neural networks (GCNs) [87], which generalize the concept of CNNs convolution to graph data by aggregating features from one node neighborhood as the following:

$$h_v^{(l+1)} = \sigma \left(\sum_{u \in \mathcal{N}(v)} \frac{h_u^{(l)} W^{(l)}}{c_v} \right), \quad (3.11)$$

being $h_v^{(l+1)}$ the updated node feature vector, σ a nonlinear differentiable function, l the

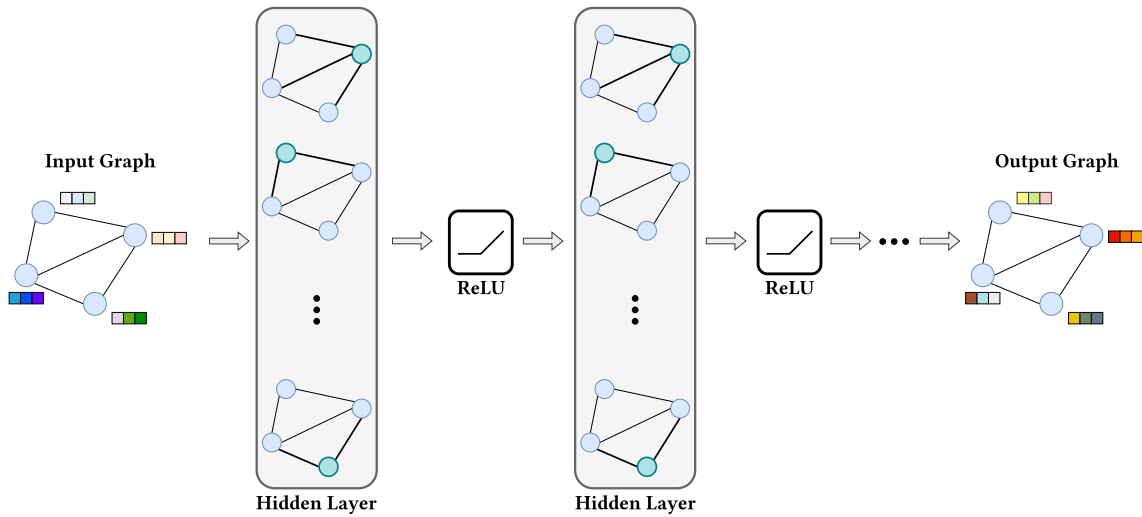


Figure 3.4: Overview of the GCN architecture.

l th GCN layer, $h_u^{(l)}$ the neighboring nodes feature vector, $W^{(l)} \in \mathbb{R}^{a \times a}$ a learnable weight matrix and c_v a normalization factor. Figure 3.4 depicts the basic GCN architecture.

In order to extend results obtained by the GCN model to heterogeneous graphs, relational graph convolutional neural networks (R-GCNs) [154] have been considered to generalize the aggregation method and include different relationships between nodes. The new formulation of hidden node representations is reformulated as follows:

$$h_v^{(l+1)} = \sigma \left(W_0^{(l)} \cdot h_v^{(l)} + \sum_{r \in R} \sum_{u \in \mathcal{N}^r(v)} \frac{W_r^{(l)} \cdot h_u^{(l)}}{c_{v,r}} \right), \quad (3.12)$$

where $\mathcal{N}^r(v)$ corresponds to the set of neighbors of node v under relation $r \in R$, $W_0^{(l)} \in \mathbb{R}$ is a learnable weight vector for each layer associated with a node's self-connection, and finally $c_{v,r}$ is again a normalization constant that can vary based on the R-GCN task.

3.2.2 Reinforcement Learning

The application of ML algorithms in prior research has consistently relied on the incorporation of domain knowledge to enhance the performance of their outputs. This knowledge is typically embedded to introduce strategies that can be exploited during the training process, ultimately leading to more robust and systematically optimized models. In contrast, the fundamental principle underlying RL [170] is to use an agent to autonomously explore its environment through a trial-and-error process. By iteratively interacting with the latter, the agent incrementally learns to optimize a predefined objective, even in the absence of explicit instructions or prior knowledge about how to achieve it.

Due to its inherent flexibility, RL has been successfully applied to address tasks that have proven intractable using supervised or unsupervised learning techniques. Notably, RL agents have demonstrated superhuman performance in complex domains such as Atari video games [127], chess [165], and Go [164]. These achievements were enabled through mechanisms like self-play and the integration of NN-based RL systems to identify optimal

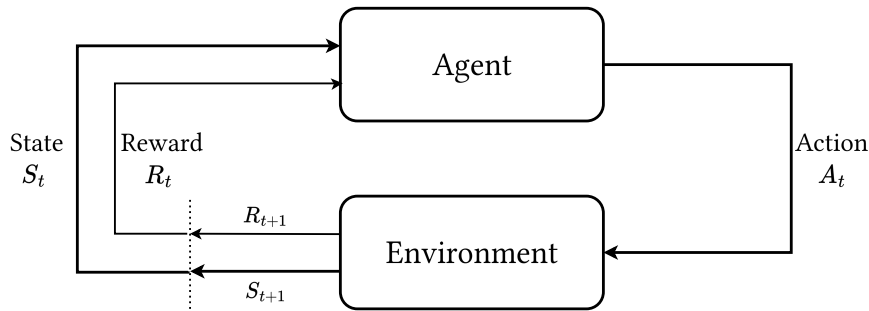


Figure 3.5: Interaction between agent and environment in Reinforcement Learning.

strategies. Beyond gaming, RL has been extensively utilized in robotics and control theory, with the goal of letting robots to navigate unstructured and unknown environments with precision, facilitating the design of sophisticated behaviors that are otherwise difficult to manually engineer [89]. Additionally, as outlined in Section 1.2, RL has also been applied to tackle combinatorial optimization (CO) problems. In order to leverage RL in CO, the problem must first be reformulated within the framework of a Markov decision process (MDP). Before exploring specific RL algorithms and learning paradigms, the following section provides an overview of fundamental MDP and RL concepts.

Reinforcement Learning Fundamentals

The family of problems that are tackled by RL belongs to the domain of *sequential decision-making*, which involves making a series of interdependent decisions over time while trying to optimize for a certain objective. This setup can be extended to many real-world problems, including analog IC layout optimization, where system dynamics are often unknown and must be inferred. As depicted in Figure 3.5, the core components of an RL setting are the *agent* and the *environment*. The agent serves as the learner and decision-maker, while the environment encompasses all external elements with which the agent interacts through its *actions*. These ones are processed by the environment to eventually return a *reward*, which is a scalar signal quantifying the quality of agent's actions, and an *observation*, representing the resulting state of the environment after the action is executed. The agent and the environment interact cyclically, with the agent's goal to maximize the cumulative reward over time by selecting optimal actions.

The mathematical formulation for modeling such framework is provided by MDPs [12], which are defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

- $\mathcal{S}$ is a finite set of states of the environment;
- $\mathcal{A}$ is a finite set of actions;
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the state transition probability matrix, with transition probability:

$$p(s'|s, a) = \mathbb{P}(S_t = s' | S_{t-1} = s, A_{t-1} = a), \quad (3.13)$$

where t is the current time step in the interaction between the agent and environment;

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, where $r(s, a)$ denotes agent's contribution to the desired outcome.
- $\gamma \in [0, 1]$ is the discount factor that properly weights the importance in time of future rewards.

Contextualizing the MDP concepts to the RL framework, at each discrete time step t the agent finds itself certain state $s_t \in \mathcal{S}$. Based on the information it can get from s_t , it operates an action $a_t \in \mathcal{A}$ to interact with the environment, receiving a reward $r_{t+1} \in \mathcal{R}$ as response and landing in a new state s_{t+1} following p , which is the function that governs the dynamics of the MDP. As shown in Equation 3.13, the probability of transitioning to a new state depends solely on the preceding state-action pair. This system characteristic is referred to as *Markov property*, which asserts that the current state s_t encapsulates all relevant information from the past necessary to predict the system's future behavior.

As said earlier, the reward function $\mathcal{R}$ plays a pivotal role in the agent's learning procedure, as its primary goal is to maximize the discounted sum of rewards, or discounted return. The *return* G_t is defined as:

$$G_t = \sum_{k=0}^T \gamma^k \cdot R_{t+k+1}, \quad (3.14)$$

where T represents the final time step, also known as the *horizon*, and R_{t+k+1} the reward value at $t + k + 1$ time step. The value assumed by T determines the length of an *episode*, which can be either finite or infinite. This motivates the presence of a discount factor γ^k in Equation 3.14. In fact, a value close to 0 prioritizes immediate rewards, while a value closer to 1 assigns greater importance to long-term rewards. Incorporating a discount factor enables the representation of uncertainty regarding future states and reflects a natural human tendency to favor immediate rewards over delayed ones. However, while this preference for immediate rewards aligns with human intuition, it is not always the optimal approach in problem-solving contexts.

Building on this point, the agent determines which action to take based on a probability distribution over actions given a particular state called *policy*:

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s), \quad (3.15)$$

where $a \in \mathcal{A}(s)$, i.e. the set of possible actions that can be taken from state s . Being π stochastic, the agent action selection is highly flexible, allowing it to adapt dynamically to the diverse situations it may encounter during interactions with the environment. Through repeated exploration and evaluation of different actions, the agent seeks to converge to an optimal policy, π^* , that provides the best action for each possible state configuration. To determine the optimal policy, an RL algorithm must estimate how good it is for the agent to be in a given state or, similarly, how good it is to perform a given action in a given state. These, almost equivalent, informations are provided respectively by the *state-value function* (3.16), also called value function, and the *action-value function* (3.17):

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \quad (3.16)$$

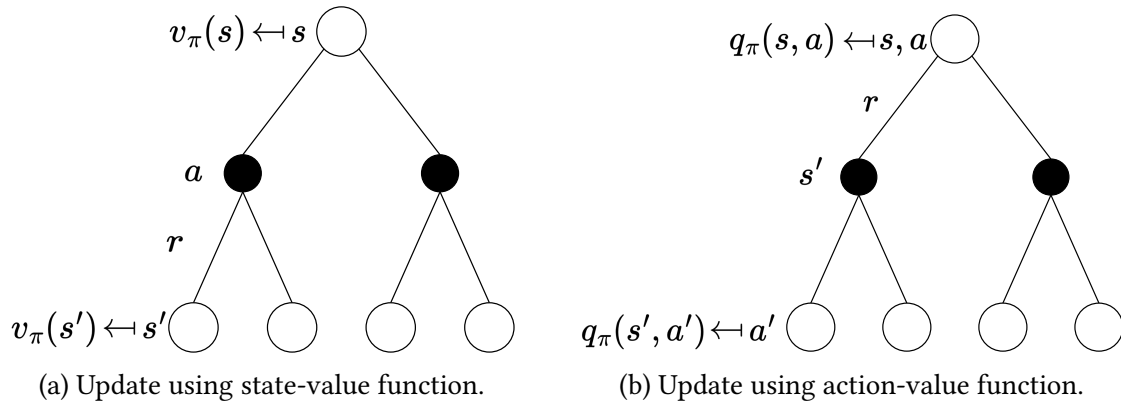


Figure 3.6: Bellman equation visualization. Each open circle represents a state and each solid circle represents a state-action pair. Starting from state s the agent could take any of some set of actions. From each of these, the environment could respond with one of several next states s' , along with a reward r . The Bellman equation averages over all the possibilities, weighting each by its probability of occurring. Figures from [163].

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]. \quad (3.17)$$

The goodness of being in a certain state is described by $v_{\pi}(s)$ in terms of expected return $\mathbb{E}_{\pi}$ when starting in a state s and following policy π , while the same concept is expressed by the $q_{\pi}(s, a)$ as the expected return starting from s , taking the action a , and thereafter following π . One essential characteristic shared by both of these functions is their recursive representation through the *Bellman equation* [12], which underlines the relationship between the value of a state (or state-action pair) and the values of its successor states. These recursive formulations are given by:

$$v_{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s], \quad (3.18)$$

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]. \quad (3.19)$$

Figure 3.6 shows the idea behind the Bellman equation and the way the update from one state to another one is performed.

It is important to emphasize that every policy corresponds to a unique state-value or action-value function. A state(action)-value function is said to be optimal if it provides the maximum state(action)-value function over all policies as follows:

$$v^*(s) = \max_{\pi} v_{\pi}(s), \forall s \in \mathcal{S}, \quad (3.20)$$

$$q^*(s, a) = \max_{\pi} q_{\pi}(s, a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \quad (3.21)$$

Since value functions define a partial ordering over policies, one policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. Consequently, it can be concluded that for every MDP there exists at least one optimal policy π^* that is better than or equal than all the other ones, i.e. $\pi^* \geq \pi \forall \pi$, and that all optimal policies achieve the same optimal state(action)-value function. An

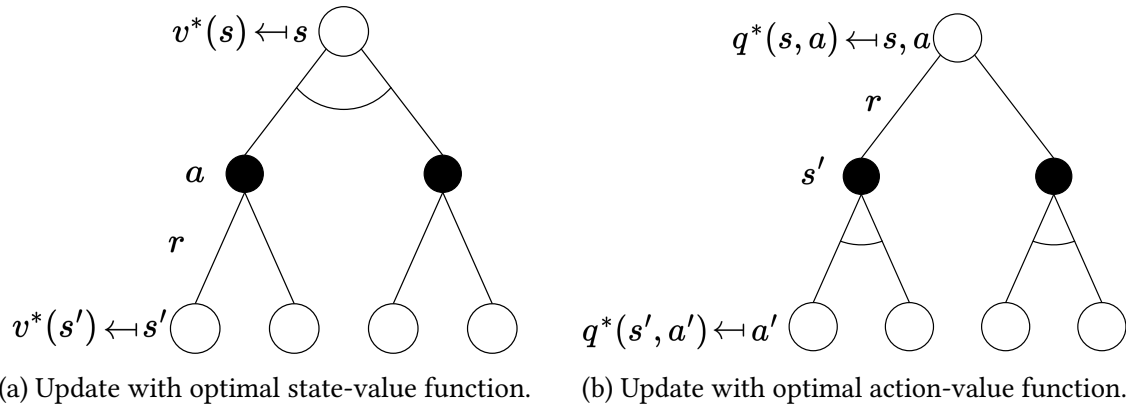


Figure 3.7: Same as fig. 3.6 but in this case arcs have been added at the agent's choice points to represent that the maximum over that choice is taken rather than the expected value given some policy. Figures from [163].

optimal policy π^* can be derived by greedily selecting the action that maximizes $q^*(s, a)$:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg \max_{a \in \mathcal{A}} q^*(s, a), \\ 0 & \text{otherwise.} \end{cases} \quad (3.22)$$

The optimal value functions $v^*(s)$ and $q^*(s, a)$ can be computed using the recursive Bellman optimality equations (3.18, 3.19), which leads to the formulation of the *Bellman optimality equation* (Figure 3.7 shows its visual representation):

$$v^*(s) = \max_{a \in \mathcal{A}} q^*(s, a) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \cdot v^*(s') \right], \quad (3.23)$$

$$q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \cdot v^*(s') = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \cdot \max_{a' \in \mathcal{A}} q^*(s', a'). \quad (3.24)$$

However, solving the Bellman optimality equation explicitly is computationally infeasible in most real-world scenarios. This is due to practical limitations, such as the need for precise knowledge of the environment dynamics and the prohibitive computational cost of enumerating all possible states and actions, particularly in large or continuous state spaces. Therefore, RL algorithms often use approximations to learn optimal value functions and derive optimal policies. These approximations, which are typically provided using DNNs, enable RL to scale to complex environments without relying on exact solutions, as explained in the following section.

RL and Deep RL Algorithms

A high level partition between the different RL algorithms can be done based on the level of knowledge of the model, i.e. the environment. *Model-based methods* assume access to the environment's dynamics, such as transition probabilities and rewards, which can either be explicitly provided or learned. On the other hand, *model-free methods* solely leverage the agent's interactions with the environment to learn an optimal policy. Another

distinction that can be made between RL algorithms is whether they are value-based or policy-based.

Value-based techniques focus on finding an optimal policy through the approximation of $v(s)$ or $q(s, a)$. In this way, the policy is implicit and can be derived directly from the value function using a greedy or ϵ -greedy approach. The most basic formulation of such approach is the *value iteration* algorithm, which aims at providing an estimate of the optimal value function by iteratively applying the Bellman optimality equation 3.20 as update rule:

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \cdot v_k(s') \right]. \quad (3.25)$$

For any arbitrary initial condition v_0 , it is proved that the sequence of updates converges to v^* . Despite theoretical guarantee, value iteration comes to be impractical when trying to solve large scale problems, given the need of storing very large lookup tables containing state or state-action pairs. Furthermore, value iteration is a model-based approach which requires complete knowledge of the system dynamics, which is pretty hard to get given the uncertainty that characterizes real-world scenarios. To address this limitation, *Q-Learning* [179] was introduced as a model-free, *off-policy* algorithm. In fact, Q-Learning operates using a *behavioral policy* which leverages collected data from states, actions and rewards, to optimize the action-value function to determine the optimal policy independently. The algorithm employs the concept of *temporal difference* (TD) learning, enabling incremental updates to the state-action value function based on observed transitions. The update rule for Q-Learning is defined as:

$$q_k(s_t, a_t) = q_{k-1}(s_t, a_t) + \alpha \left[r(s_{t+1}, a_{t+1}) + \gamma \cdot \max_a q_{k-1}(s_{t+1}, a) - q_{k-1}(s_t, a_t) \right], \quad (3.26)$$

where k indicates the current iteration, α is the learning rate, and the term in brackets represents the *TD error*, i.e. the difference between the predicted and actual values. Over successive iterations, the algorithm minimizes the TD error, eventually leading $q(s, a)$ toward an optimal state-action value function $q^*(s, a)$. Even though Q-Learning provides a way to be independent w.r.t. system dynamics, it does not address the scalability issue inherent to value-based techniques. To solve such problem, function approximators as NNs can be used to provide a parametric estimate of the value functions. Mnih et al. in [127] presented a deep Q-Network (DQN) that approximates $q(s, a)$ using a NN, denoted as q_θ , which is trained by minimizing a mean squared error loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(q_\theta(s, a) - r - \gamma \max_{a'} q_\theta(s', a') \right)^2 \right] \quad (3.27)$$

where $\mathcal{D}$ is a replay buffer that stores past experiences from interactions with the environment.

Policy-based methodologies on the other hand directly model the policy instead of relying on approximation provided by the value function. The policy is typically represented as a parametric function:

$$\pi_\theta(s, a) = \mathbb{P}(a|s, \theta) \quad (3.28)$$

that is optimized in order to find the best set of parameters θ that leads to an optimal policy π_θ^* . This optimization is driven by maximizing a scalar performance measure,

$\mathcal{J}(\theta)$, with respect to the policy parameter, which varies depending on the nature of the problem. Among the key advantages of such approaches, there are better convergence properties, higher effectiveness in high-dimensional or continuous action spaces and the fact that the learned policy is stochastic. Overall, policy-based methods aim to maximize $\mathcal{J}(\theta)$ such that their updates approximate gradient ascent in $\mathcal{J}$:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla \mathcal{J}(\theta_t)}, \quad (3.29)$$

where $\widehat{\nabla \mathcal{J}(\theta_t)} \in \mathbb{R}^d$ is a stochastic estimate of the gradient of the performance measure with respect to θ_t , and α is a step-size parameter. Algorithms that follow this approach are referred to as *policy-gradient methods*. The gradient of $\mathcal{J}(\theta)$ can be computed using the *policy gradient theorem* [171], which provides an analytic expression for the gradient:

$$\nabla_{\theta} \mathcal{J}(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) \cdot q_{\pi_{\theta}}(s, a)]. \quad (3.30)$$

This expression allows the computation of updates to optimize Equation 3.29 directly. The first policy-gradient algorithm, REINFORCE [181], leverages a Monte Carlo approach to obtain samples such that the expectation of the sample gradient is proportional to the actual gradient of $\mathcal{J}$. This means that the action taken at time t are used to update the policy gradient. In order to overcome the high variance of such naive approach, a baseline term b can be introduced to perform a comparison with the action value and eventually reduce the variance of the gradient estimate without any inducted bias, as long as b is independent of the action taken. The adjusted policy gradient is given by:

$$\nabla_{\theta} \mathcal{J}(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \cdot (q_{\pi_{\theta}}(s_t, a_t) - b(s_t))]. \quad (3.31)$$

A natural choice for b is an estimate of the state-value function $\hat{v}(S_t, \mathbf{w})$. In this case, $q(s_t, a_t) - v(s_t)$ is referred to as the *advantage function*, which quantifies the additional reward obtained by choosing a particular action compared to the expected reward for the state. *Actor-critic* [7] extend basic policy-gradient approaches by incorporating two components:

- The *actor* estimates the policy $\pi_{\theta}(a|s)$, i.e. outputs the conditional probability of taking each action a when in a state s .
- The *critic* estimates the value function $\hat{v}(S_t, \mathbf{w})$, which evaluates the long-term expected reward for a given state. In this way the critic can inform the actor about the goodness of the action that it took and how to change its behavior accordingly.

Nowadays, actor-critic frameworks have become one of the most established approaches in RL for tackling different problems, owing to their improved sample efficiency, faster convergence and versatility across discrete and continuous action spaces. Among these, the most popular approach, that has also been used in the following methodology chapters of this thesis, is *proximal policy optimization* (PPO) [157]. PPO alternates SGD to optimize a clipped objective function with sampling data from the environment it interacts with. At the core of PPO is a new mechanism to overcome destructive policy updates. Traditional policy-gradient methods can suffer from instability due to excessively large updates to the policy, which can lead to performance degradation. PPO mitigates this issue by

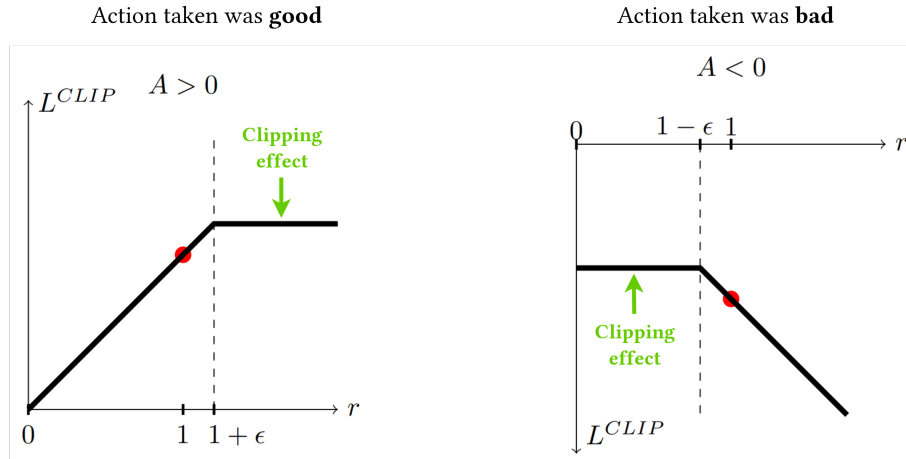


Figure 3.8: Plots of the “surrogate” objective L^{CLIP} as a function of the probability ratio $r(\theta)$ showing the clipping effect on positive (left) and negative (right) surrogates. Adapted from [157].

constraining the policy update so that the new policy does not deviate significantly from the previous one. This is achieved by introducing a clipped “surrogate” objective function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)], \quad (3.32)$$

where $\hat{\mathbb{E}}_t$ is the empirical average over a finite batch of samples at timestep t , $\hat{A}_t$ is an estimator of the advantage function, $r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}(a_t|s_t)}}$ is the probability ratio between current and previous policy, and ϵ is a hyperparameter to regulate the clipping procedure. The probability ratio, $r(\theta)$, takes values greater than 1 if the specific action a_t has a higher likelihood for the current policy than the prior one, and values between 0 and 1 when the action is less likely for the current policy. The main distinction of PPO lies in the second term within the “surrogate” objective, where the probability ratio gets truncated with a clipping operation that eliminates extreme policy updates exceeding the interval $[1 - \epsilon, 1 + \epsilon]$. The main benefits of using clipping reflect on multiple factors:

- The combination of the minimum operation and clipping leads to distinct behaviors depending on whether the advantage estimate is positive or negative. For positive advantages, the loss function flattens out when $r(\theta)$ gets too high and so it is clipped to $1 + \epsilon$. Similarly, for negative advantages, the loss function flattens out as $r(\theta)$ approaches 0, implying that a certain action is considerably less probable on the current policy compared to the previous one. In this way it is possible to avoid greedy behaviors both for positive and negative advantages, as displayed in Figure 3.8.
- The clipping mechanism also enables recovery when gradient steps lead to undesired outcomes. As shown in Figure 3.9, a $r(\theta)$ growth in the right region of the right plot due to an overly aggressive gradient step can negatively affect the policy. If that were the case, L^{CLIP} ensures that the gradient moves in the opposite direction, as the loss function becomes negative, effectively reversing the step. These “undo” regions, corresponding to the unclipped term $r_t(\theta)\hat{A}_t$, help correct mistakes from prior updates.

Finally, PPO runs the policy by having multiple parallel actors collecting data and then sampling mini-batches of this data to train for K epochs using the clipped “surrogate”

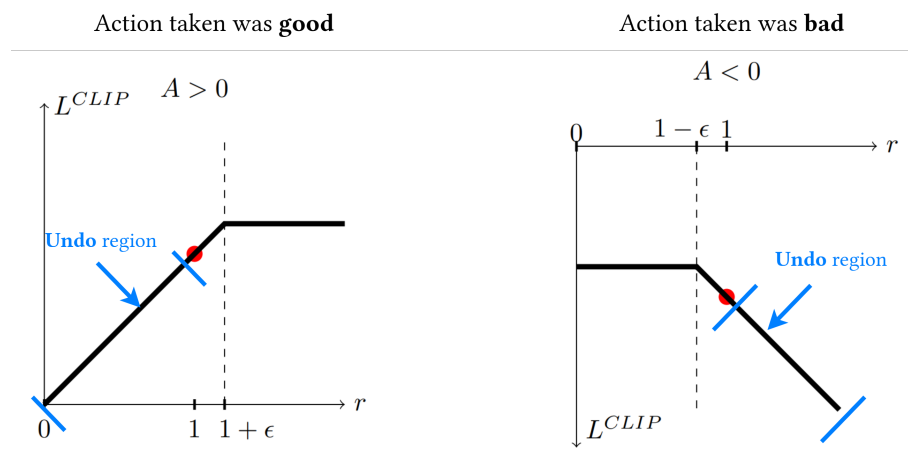


Figure 3.9: Plots of the “surrogate” objective L^{CLIP} as a function of the probability ratio $r(\theta)$ showing the undo regions for positive (left) and negative (right) surrogates. Adapted from [157].

objective function. This allows to extract more information from the same set of samples compared to traditional methods, leading to better convergence and more efficient learning.

Chapter 4

Graph-Based Optimization for Routing

The routing problem in analog IC design involves establishing connections between multiple pins while minimizing the overall circuit wirelength. Additionally, routing must adhere to stringent constraints imposed by both technology and electrical considerations, such as parasitics, layout density, and design rules. The relationships between devices and terminals in the circuit can be effectively represented as a graph, where nodes correspond to pins and edges represent the possible interconnects between them.

Over the years, many algorithms have been developed to efficiently generate near-optimal paths for satisfying all required interconnections. For the purposes of this thesis, this chapter will provide a detailed background description of two widely used approaches. Specifically, Section 4.1 depicts the A* algorithm, a heuristic pathfinding method widely used across various fields in computer science for its completeness, optimality, and computational efficiency in finding the least costly paths between two nodes. On the other hand, Section 4.2 introduces the Steiner tree problem, which provides a global optimization framework for interconnecting multiple terminals, and the heuristics used to solve it.

4.1 A* Algorithm

The A* algorithm was first introduced by Hart et al. in 1968 [65] as an improvement over the classical *breadth-first search* (BFS) [36, Chapter 22.2] and *Dijkstra's* algorithm [42]. BFS and Dijkstra's algorithm operate as wave-based search methods on a graph $G(V, E)$, starting from a source node s . Although these algorithms can be applied to graphs of any nature, they are typically used with grid-based representations in routing problems, particularly for routing problems in EDA [41, 184]. Other fields of application of A* involve video games [95], navigation systems [176] and robotics [43].

A key concept in such search procedures is the *frontier*, which represents the set of nodes to be expanded during the search. BFS is suited for cases where all edges have equal cost, as it explores nodes level-by-level, ensuring the shortest path in terms of the number of edges. When the edges have varying costs, i.e. we have a weighted graph, Dijkstra's algorithm is applied, as it prioritizes nodes based on the cumulative cost to reach them. Both algorithms expand the frontier by examining the neighboring nodes of

a node p and selecting the one with the lowest cumulative cost $g(q)$ to reach a neighbor q . In order to achieve such goal, a priority queue has to be used to maintain and periodically reorder nodes based on their path costs. In the case a node q has already been visited and inserted into the priority queue, but a newly discovered path offers a lower reaching cost, then $g(q)$ has to be updated accordingly.

While BFS and Dijkstra's algorithm perform an uninformed search by expanding the frontier in any direction, which is reasonable when looking for a path to all graph locations or several of them, this can be inefficient when only a single target node t is of interest. A* improves upon this by using an *informed* search approach by integrating an heuristic to properly guide the search towards t . The cost function for A* is defined as:

$$f(n) = g(n) + h(n), \quad (4.1)$$

where $g(n)$ is, as before, the cumulative cost to reach node n , and $h(n)$ is a heuristic estimate of the cost to reach the target node t from n . The updated cost function $f(n)$ is used to determine the order in which nodes will be traversed in a priority queue. Common choices for $h(n)$ involve the Euclidean, Manhattan, diagonal distances, depending on the nature of the grid and problem constraints. As long as the heuristic $h(n)$ does not overestimate the true distance to the target, making it *admissible*, A* guarantees finding an optimal path, similar to Dijkstra's algorithm. Moreover, on finite graphs with non-negative edge weights, A* is guaranteed to terminate and is complete, i.e. it will always find a solution if one exists. Finally, it is straightforward to see that both BFS and Dijkstra's algorithms are a specific case of A* in which $h(n) = 0$.

The grid on which the search is performed can include obstacles or constraints that must be satisfied. In analog IC layout, these obstacles often correspond to already placed devices, high-voltage areas, or other regions where routing is prohibited or restricted. Incorporating obstacle awareness and constraint compliance can be done essentially in two manners. One approach is to assign large penalty costs to nodes that correspond to blockages or constraint violations, making these nodes highly undesirable for the pathfinding algorithm and ensuring they are avoided unless no feasible alternative exists. Alternatively, nodes corresponding to obstacles can be entirely excluded from the set of possible neighbors during the graph construction phase, guaranteeing that these regions are never considered during the search process.

Despite being optimal and complete, A* can incur in lengthy runtimes for finding the optimal path, depending on the grid size and the selected heuristic function. To address this challenge and accelerate the search process, variations of A* have been developed, such as *weighted* A* and *bidirectional* A*. Weighted A* introduces a weight $\epsilon > 1 \in \mathbb{R}^+$ to the heuristic function $h(n)$ in Equation 4.1 as follows:

$$f(n) = g(n) + \epsilon \cdot h(n), \quad (4.2)$$

eventually biasing the search towards nodes that are closer to the goal. However, this speedup comes at the cost of losing strict optimality, as the paths found may no longer be the absolute shortest. In this sense, ϵ acts as a control parameter, with larger values favoring faster but less optimal searches.

Bidirectional A* further improves efficiency by performing concurrent searches from s and t . The forward search proceeds from s , while the backward search starts from t , with each search expanding nodes iteratively until they meet. During execution, the algorithm

alternates between the two searches, with the forward search temporarily advancing to explore the best forward candidate, followed by the backward search progressing toward that candidate. This alternating process continues until the two search frontiers converge on the same node, identifying the path.

While the A* algorithm excels at finding the shortest paths between two points in a graph, its primary limitation lies in its focus on single-source, single-target pathfinding. However, routing problems in analog IC design often require connecting multiple pins simultaneously. One common approach to address this is to decompose the multi-pin network into a series of two-pin connections, such as by constructing a minimum spanning tree (MST). Nevertheless, MST-based routing results would depend on the decomposition and often lead to only suboptimal solutions. Instead, as discussed in the next section, a Steiner tree-based approach is preferred due to its ability to natively and optimally handle multi-terminal networks.

4.2 Steiner Trees

The Steiner tree problem (STP) is one of the most extensively studied and prominent problems in the CO literature. Its applications span a wide range of fields, from network design and integrated circuit layout to location problems, and more recently, to emerging areas such as machine learning, systems biology, and bioinformatics [110].

Formally, given an undirected weighted graph $G(V, E)$ with edge weights $w : E \rightarrow \mathbb{R}^+$ and terminal nodes $T_n \subseteq V$, the STP aims to find a subgraph $G_s = (V_s, E_s) \subseteq G$ such that G_s connects T_n and minimizes the total weight $w(G_s) := \sum_{e \in E_s} w_e$. The additional nodes used to construct G_s , called *Steiner points*, belong to the set $S = V_s \setminus T_n$. Steiner points enable the reduction of total interconnection costs, making the solution more efficient than simply spanning the terminal nodes. The edge weights of the graph can represent a variety of metrics, but in real-world scenarios, they are typically derived from the Euclidean or Manhattan distances between nodes.

In particular, the rectilinear Steiner minimum tree (RSMT) is of primary interest in the EDA field, where interconnections between nodes are represented as a sequence of vertical and horizontal segments. The number of potential Steiner points for constructing an RSMT is theoretically infinite. However, Hanan in [63] proved that for a given set of P pins, there exists an optimal RSMT where all Steiner points lie on the *Hanan grid*, which is constructed by extending vertical and horizontal lines through every pin in P , as illustrated in Figure 4.1b.

An interesting extension of the RSMT problem involves the introduction of obstacles, resulting in the formulation of the obstacle-avoiding RSMT (OARSMT). This variation gains particular importance in circuit physical design, where, as already mentioned in Section 4.1, obstacles play a crucial part during the routing stage [34, 73, 74, 100, 105, 106]. Incorporating obstacles into the Hanan grid consists in simply adding further Steiner points in correspondence of each intersection between the Hanan grid lines and the projection from the corners of every blockage, as shown in Figure 4.1c.

While exact algorithms such as ILP [5] or Branch and Bound [46] can generate optimal Steiner trees for small problem instances, the STP is NP-hard, making these methods impractical for larger graphs or real-world applications. Consequently, a variety of heuristics and approximation techniques have been developed to tackle the problem more

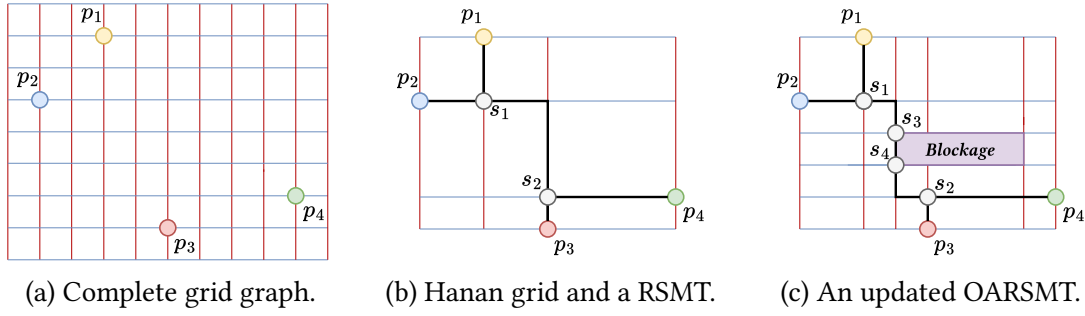


Figure 4.1: Comparison between a complete grid graph and the Hanan grid generated from a set of four pins. The construction of the RSMT and the OARSMT is shown, with Steiner points highlighted in both cases.

efficiently while maintaining sufficient accuracy. Heuristic approaches often rely on the MST as a starting point. For instance, MST-based methods provide a $3/2$ -approximation of the Steiner tree [76], making them a robust foundation for practical solutions. Kahng and Robins [80] proposed the iterated 1-Steiner heuristic, which begins with an MST and iteratively selects Steiner points that reduce the total wirelength. The process continues until no further improvements can be made, producing a near-optimal solution that balances efficiency and accuracy.

More advanced techniques have been developed to handle larger and more complex instances. FLUTE [35], a widely used and efficient RSMT algorithm, precomputes optimal solutions for nets with up to nine pins and stores them in a lookup table. For larger nets, FLUTE recursively breaks them into smaller subnets until they are small enough to be solved using the precomputed table. This trade-off between precomputation and recursion allows FLUTE to achieve high processing speeds while maintaining result quality.

Building upon these traditional approaches, more recent advancements witness the employment of modern hardware and ML techniques. Fallin et al. in [45], propose a GPU-accelerated heuristics for constructing RSMTs, significantly improving runtime performance while still relying on MST-based methods for initial tree construction. Khang et al. in [82] leveraged a neural network to boost performances of a previous dynamic programming (DP)-based RSMT technique, using four different NN models to compute encodings of DP subproblems instances and assess Steiner point selection quality. Moreover, Lin et al. in [104] proposed a RL-based OARSMT algorithm that is capable to learn and generate heuristics for solving such problem by using a GCN for feature extraction, allowing the model to effectively adapt to the structure of the Steiner tree graph without relying on manually designed methodologies.

Part II

Machine Learning-Based Layout Automation

Chapter 5

Fast ML-driven Analog Layout using Reinforcement Learning and Steiner Trees

This chapter presents an AI-driven methodology to reduce the bottleneck often encountered in the analog ICs layout phase. We frame the floorplanning problem as a MDP and leverage RL for automatic placement generation under established topological constraints. Consequently, we introduce Steiner tree-based methods for the global routing step and generate guiding paths to be used to connect every circuit block. Finally, by integrating these solutions into a procedural generation framework, we present a unified pipeline that bridges the divide between circuit design and verification steps. Experimental results demonstrate the efficacy in generating complete layouts, eventually reducing runtimes by 1.5% compared to manual efforts.

In Section 5.2, a brief overview of floorplanning and routing problem objectives is given. Then, in Section 5.3, the RL-based floorplan generation, OARSMT global routing and their addition to the ANAGEN flow is detailed. Furthermore, Section 5.4 presents the results obtained on an Infineon developed OTA circuit, and in Section 5.5 the final considerations regarding the proposed solutions and limitations are addressed. Finally, Section 5.6 provides a concise overview of the state-of-the-art fully automated layout engines, along with a brief overview of the paradigm followed by ANAGEN. The content of this chapter has been published in [8].

5.1 Introduction

The layout of analog ICs, traditionally dependent on manual expertise, has been slow to adopt AI advancements that have transformed digital counterpart through EDA especially owing to high susceptibility to noise, variations in process, voltage, and temperature. These properties, in fact, as mentioned in Section 2.2, translate into several topological requirements that must be met to produce robust layouts, tackling possible parasitics and routability issues.

Analog floorplanning has traditionally required a significant amount of expert knowledge and involved a high degree of repetitive work. Metaheuristics such as SA, PSO, GA presented in Section 3.1 have been employed to streamline this process, however, they

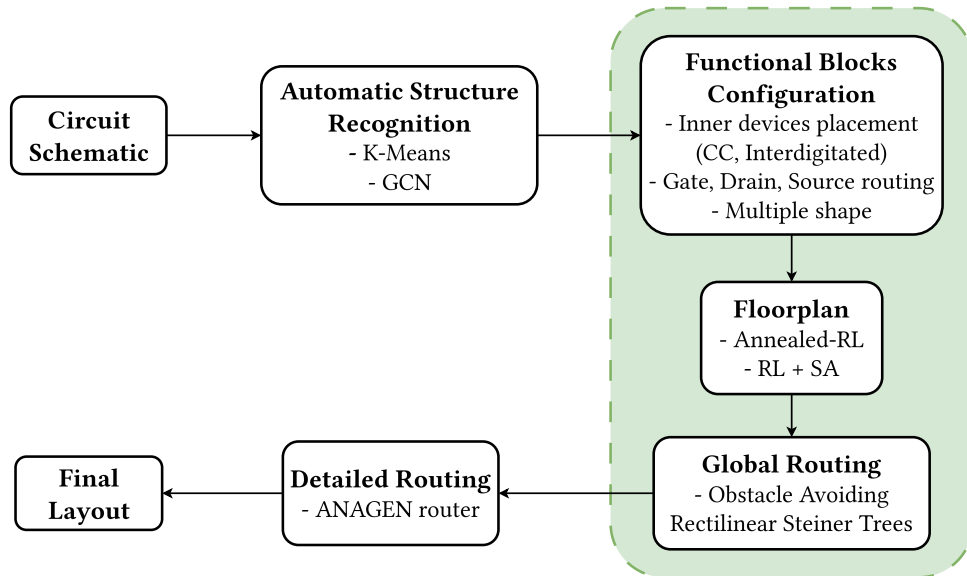


Figure 5.1: High-level schematic of the AI-powered automated layout pipeline with this chapter contributions highlighted in green.

cannot utilize past or external knowledge to enhance exploration of the solution space, as each problem instance is optimized anew. Recently, learning-based techniques as RL have gained traction thanks to their effectiveness on solving combinatorial problems [123], to which floorplanning belongs. Very few attempts have been made following this direction, especially in the analog setting [59] and, to our knowledge, the use of RL is still to be explored. Efforts to automate routing with DL models [103] have seen limited real-world application as well, largely due to the scarcity of diverse, public datasets.

We therefore propose two novel approaches for optimal floorplan generation, framing the problem as a Markov decision process and eventually employing RL techniques both independently and in conjunction with SA. The mathematical representation selected for the floorplan is the sequence pair topological model. Both approaches are designed to minimize circuit area occupation and estimated wirelength. In addition to this, the proposed methods guarantee the support of fundamental geometric requirements such as device symmetry, alignment and optimal routability, along with the possibility to define specific aspect ratios of the final floorplan and a variable shape for each device. These features aim to achieve a level of flexibility comparable to manual layout design, meeting the high standards typically required in analog layout synthesis. Finally, our models are trained on synthetic floorplans for optimal topological constraint handling, ensuring they generalize well to new, real-world circuits.

An OARSMT-based algorithm, inspired by [106], is proposed to generate global routing guiding paths, enabling optimal net and metal layer parameter selection. This algorithm provides immediate guidance for ANAGEN [39, 136] to efficiently complete detailed inter-block connections.

To conclude, an overview of the complete pipeline, integrating both automatic floorplan and routing strategies with a preliminary circuit functional blocks recognition [137], is illustrated in Figure 5.1.

5.2 Problem Definition

5.2.1 Floorplanning

In this study, during the floorplanning procedure, we focus on pure area minimization of a floorplan F (5.2) or its combination with half-perimeter wirelength (HPWL) within a fixed-outline constraint (5.3). HPWL, a widely used approximation for true wirelength, is computed as the half-perimeter of all nets' bounding boxes:

$$\text{HPWL} = \sum_{i=1}^n \max(x_i) - \min(x_i) + \max(y_i) - \min(y_i), \quad (5.1)$$

where x_i, y_i are the endpoints of net i , and n is the total count of nets in the netlist. Coming back to the floorplanning cost definition, the two possible ones are:

$$\text{cost} = F_{\text{area}} \quad (5.2)$$

$$\text{cost} = \alpha \frac{F_{\text{area}}}{\sum_{i=1}^m A_i} + \beta \frac{\text{HPWL}}{\text{HPWL}_{\text{avg}}} + (1 - \alpha - \beta)(\text{Ar}^* - \text{Ar})^2. \quad (5.3)$$

Here, α and β are weights in $[0, 1]$ balancing area and wirelength terms importance. A_i represents the area of the i^{th} device and HPWL_{avg} the average HPWL from the last 100 simulations. Both elements serve for standardization purposes, guaranteeing a comparable scale of the two different optimization goals. Finally, Ar^* and Ar are respectively the target and current floorplan aspect ratios.

5.2.2 Routing

The global routing phase aims to optimize the allocation of on-chip routing resources, commonly discretizing the layout into a grid, to interconnect circuit components given their placement. As per ANAGEN's rules, we treat each circuit block not belonging to the network of interest as an obstacle. This motivates the usage of an obstacle avoiding technique to connect all the desired pins together. For this reason, a minimal wirelength routing tree is needed to ensure correctness and perfect compliance with the industrial tools we have to interface with. After having constructed such rectilinear tree, the retrieved paths can be used to form the basis for the detailed routing stage, which precisely defines physical interconnections of the final layout.

5.3 Automatic Layout Generation

Following the overview provided in Figure 5.1, the following sections cover the methodological details that have been employed to automate the layout process.

5.3.1 Structures Recognition and Device Shapes Generation

Given an input analog circuit schematic, we search for an optimal placement of each device minimizing one of the defined cost functions (5.2, 5.3). Utilizing Infineon's structure recognition tool, we apply clustering and GCN to detect functional blocks within

the schematic. Each functional block consists of one or more devices present in the circuit schematic, such as differential pairs, current mirror banks, etc. Although manual identification of these blocks might appear straightforward, it is a critical step in the automation pipeline, as it provides an essential foundation for the subsequent automated steps.

Subsequently, various block shape configurations are generated. This is possible by ensuring compliance to a fixed total device width:

$$W_{\text{tot}} = w_f \times N_f \times M, \quad (5.4)$$

respectively being finger width, number of fingers, and device multiplicity. Just by sweeping across admissible ranges of such parameter values, which are originated from experienced designer specifications defined in the circuit schematic, it is possible in fact to produce several variants of the same recognized structure while also taking into account design rule checks.

Moreover, leveraging ANAGEN's capabilities, we tailored intra-block properties to the type of structures identified. For instance, when a differential pair is detected, devices are arranged in an interdigitated or common centroid layout to reduce mismatch, with the latter approach inspired by [159]. Finally, through a reverse engineering process, conventional routing patterns such as metal layer selection, gate, drain and source connections, dummy transistors integration, among other enhancements have been adopted.

5.3.2 RL-SA Floorplanning

In our first floorplanner setting, an RL agent generates a starting point to be refined by SA. The agent in fact perturbs few times the SP representing the floorplan, as well as doing rotation or altering blocks' shape, thus modifying their parameter setup. During this stage, the agent receives intermediate rewards defined as:

$$r_t = \text{cost}(s_t) - \text{cost}(s_{t-1}), \quad (5.5)$$

with s_{t-1} and s_t denoting the states previous and after agent's action at time step t .

SA then iterates on this early configuration to eventually find an optimal layout. The agent is informed of the goodness of its initialization through a global reward:

$$r_{\text{final}} = \text{cost}(s_T) - \text{cost}(s_I), \quad (5.6)$$

with s_T and s_I respectively being the states after the RL initialization and the SA refinement. This strategy leverages RL's capacity for broad search space exploration, compensating for SA's susceptibility to poor initialization that may result in suboptimal layouts. Moreover, we adopted a cyclical RL-SA collaboration during training, where environment's state is not reset between episodes, i.e. a floorplanning run, allowing the agent to start a new episode upon the SA optimized floorplan state rather than from scratch. This technique, inspired by Qingpeng et al. [18], proved to enhance agent's exploration efficiency, leading to optimized solutions with smaller costs in shorter run-times. This is because r_{final} acts as an estimate for the value function of a technically infinite episode, i.e. with no terminal state, since the agent's action space consists of perturbations.

5.3.3 Annealed-RL Floorplanning

An RL agent is now in charge of finding optimal searching techniques to generate a floorplan entirely on its own. The search phase entails sampling a neighboring state using the same set of actions described for SA. The agent has then to decide whether to accept or not the new supplied sample. In order to better direct the agent through its exploration and somehow mimic the SA search behavior, the environment state is augmented with the following additional metrics:

- *Cost related:* The agent keeps track of the current, minimum, average, and neighboring state costs to make better informed decisions; for example, it will likely reject states if their cost substantially exceeds the observed minimum.
- *Search phase related:* A scalar, analogous to the temperature parameter in SA, represents the current optimization step, informing at state-level the RL agent on its phase in the optimization process.

Being solely dependent on the number of circuit devices, this design enhances model capability to learn from disparate floorplan configuration and ensures its reusability across layouts with diverse user-defined constraints during inference.

5.3.4 RL Training Setup

In our study, PPO state-of-the-art actor-critic method is employed for agent training. To promote generalization, robustness, and following empirical trials, each model is trained on a diverse set of synthetic floorplans featuring varying topological constraints and aspect ratios. We train one model for each circuit category, differentiated by the count of devices, spanning from 5 to 20.

The RL-SA framework is trained using 128 RL steps and 10 epochs, with SA performing 2000 steps starting at a temperature of 15. On the other hand, the Annealed-RL setup undertakes a longer training with 5000 steps and 50 epochs for thorough learning. Both frameworks' neural networks use 3 and 2 hidden layers with 128 neurons for the actor and critic, respectively. Average training times for RL-SA is 5 minutes, while for pure RL it is 10 hours.

5.3.5 Post-processing & Routing

The floorplan generated by previous algorithms is refined using a congestion estimation technique [158]. This step involves grid-based analysis of wire capacities to produce a congestion map, which guides the redistribution of circuit blocks for a practical and routable layout.

At this point, we build the OARSMT for each net specified in the netlist while minimizing wirelength and avoiding blockages present in the floorplan. The resulting routing tree is decomposed into horizontal and vertical segments and bundled into conduits where feasible, i.e. when there is enough space to group all together vertical and horizontal lines that are not interleaved by any obstacle. These conduits, containing details on the connected devices, associated net, chosen metal layer, are processed by ANAGEN detailed router to finally connect all circuit's devices.

Table 5.1: Metaheuristics hyperparameters.

Method	SA		GA			PSO			
Hyperparameter	Temperature	Steps	Mutation	Crossover	Size	Inertia	Cognitive	Social	Size
Value	15	5000	0.1	0.9	200	0.8	1.49	1.49	200

Table 5.2: Runtime, floorplan empty space, and HPWL comparison between RL-based algorithms and traditional metaheuristics.

Circuit	# Devices	Algorithm	Runtime (s)		Empty space (%)		HPWL (μm)	
			mean	std	mean	std	mean	std
OTA-1	5	Annealed-RL	25.70	0.31	12.14	3.85	73.57	7.31
		RL-SA	3.60	2.54	14.02	5.24	72.44	10.16
		SA	2.69	3.28	16.58	5.08	75.60	8.93
		GA	4.54	0.20	26.94	9.81	87.60	16.76
		PSO	4.06	0.03	12.85	3.28	69.72	5.78
OTA-2	8	Annealed-RL	35.86	0.38	10.19	4.18	135.06	24.71
		RL-SA	3.42	1.12	13.61	5.53	127.16	17.32
		SA	1.76	1.63	14.38	7.14	136.09	15.05
		GA	5.14	0.05	22.43	8.42	164.80	45.51
		PSO	4.82	0.04	11.75	3.54	137.74	13.37
Bias	11	Annealed-RL	28.74	0.35	14.30	5.19	220.50	31.84
		RL-SA	2.77	2.35	14.90	5.34	249.24	29.43
		SA	5.86	12.62	14.97	4.87	236.44	33.85
		GA	5.48	0.26	28.12	11.30	320.26	69.27
		PSO	5.83	0.11	18.51	4.72	311.33	35.40

5.4 Experimental Results

Our automated layout frameworks are developed using Python 3.9 and use the stable-baselines3 library [145] to define the RL environment and PPO model.

To validate the RL-based floorplanning algorithms, performance metrics on 3 different circuits from Infineon Commercial Technology (130nm) are benchmarked against established metaheuristics, including SA, PSO and GA, whose hyperparameters can be seen in Table 5.1. Table 5.2 data clearly demonstrate the RL methods’ superiority in producing more compacted and wirelength-optimized floorplans. While the full RL approach does incur slightly longer runtimes anyway achieving best packing results, its hybridization with SA strikes a balance between optimal metrics and metaheuristic comparable speed.

Our methodology is applied to an 11-device operational transconductance amplifier (OTA), featuring a differential pair, current mirror, and cascode structure, showcased in Fig. 5.2 for practical evaluation. Considering the identified functional blocks, vertical and horizontal alignment constraints are imposed to promote layout regularity. The proposed initial template, shown in Figs. 5.3a and 5.3b, is further refined, specifically by shifting certain devices to the left, as depicted in Fig. 5.4.

A quantitative comparison regarding layout generation times, spatial efficiency and routing wirelength is detailed in Table 5.3. The data underscore the efficacy of the automated process, significantly reducing the time needed to produce fully optimized

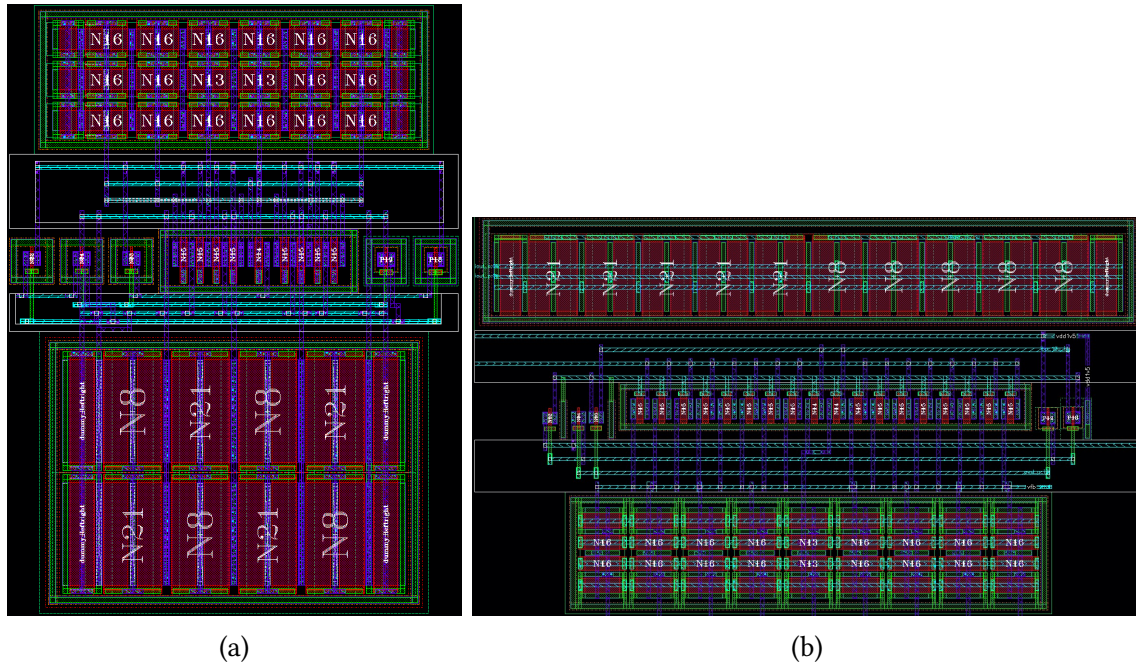


Figure 5.4: Comparison of OTA layouts: (a) generated automatically via our pipeline and (b) designed manually by a physical design engineer.

layout w.r.t. manual methods from 24 hours to ~ 21 minutes. Moreover, the produced layout is 13.8% more compact, without compromising routing quality.

5.5 Discussion

In this chapter, we addressed the complex challenge of analog ICs layout with an AI-driven approach. First, we detailed two RL-based floorplanning algorithms capable to deal with topological constraints, accommodating variable internal device and functional block configurations while minimizing area and HPWL objectives.

Additionally, we developed an OARSMT global router, offering valuable support to physical designers during this phase. Integrated into the ANAGEN framework, our methods have yielded complete layouts in a consistently reduced timeframe, from several hours to just few minutes, shortening time-to-market for analog ICs while adhering to stringent industry quality standards.

Despite these advancements, manual refinements of the automated layouts were still required, primarily due to the ongoing development of ANAGEN’s router, which has yet to reach its full potential for seamless integration. Moreover, the approaches presented in this chapter are capable to well generalize on problem instances containing the same number of structures to be placed. However, this limitation implies that a separate model must be trained for each unique circuit block count, adding a layer of complexity to an ideal workflow.

Nonetheless, the proposed automated floorplan generation, combined with strategic routing guidance for ANAGEN’s trackline placement, significantly alleviates the workload of engineers, providing a robust foundation for future improvements in analog layout automation.

5.6 Related Works on Fully Automated Analog Layout Frameworks

In recent years, the field of automated analog layout design has witnessed considerable advancement with the introduction of numerous open-source frameworks. At the moment, state-of-the-art techniques for layout automation are represented by [23, 41], which fall in the category of optimization-based layout synthesizers.

Chen et al. in [23] present a complete pipeline that processes unannotated schematic netlists to extract symmetry constraints and creates primitive devices using templates. Placement is initially carried out through an analytical procedure that simultaneously optimizes multiple objective and gets later legalized with linear programming (LP) algorithms, while routing is handled using the A* algorithm. Notably, this pipeline does not employ learning-based strategies for generating primitive devices, preventing to improve layout metrics over time by reusing past experience or tailor placement with circuit-specific information.

Very similarly, Dhar et al. in [41] propose an automatic analog circuits layout generation flow in which GNNs and graph traversal methods are employed for netlist auto-annotation, with subsequent circuit hierarchical representation, and constraint generation. The floorplanning algorithm is the one in [115], which employs SP and SA and suffers from the very same problems that have been already mentioned in Section 5.1.

In [39], Demiri et al. introduced the ANAGEN framework, a procedural generator for automating analog ICs layout. Distinct from previous approaches, ANAGEN constructs layouts by composing together basic devices and structures using modular generators. This contrasts with optimization-based methods, as it relies on predefined rule sets or templates to form layout patterns instead of specific objective functions minimization. While scripting automates much of the layout process, selecting the best parameters for each instance still requires substantial manual effort.

Chapter 6

Effective Analog ICs Floorplanning with Relational Graph Neural Networks and Reinforcement Learning

In this chapter, we present an improved automatic floorplanning algorithm with respect to the ones presented in Chapter 5, again based on RL. This new version of our approach is augmented by an R-GCN model for encoding circuit features and positional constraints. The combination of these two ML methods enables knowledge transfer across different circuit designs with distinct topologies and constraints, increasing the *generalization ability* of the solution. Applied to 6 industrial circuits, the proposed pipeline surpassed established floorplanning techniques in terms of speed, area and half-perimeter wire length. When integrated into a *procedural generator* for layout completion, overall layout time was reduced by 67.3% with a 8.3% mean area reduction compared to manual layout.

The RL problem formulation and our R-GCN and RL-based floorplanning system are detailed in Section 6.2. Experimental results are provided in Section 6.3, while final remarks are drawn in Section 6.4. The content of this chapter has been published in [10].

6.1 Introduction

In Chapter 5, an initial attempt to develop a learning-based approach for the automated generation of complete analog IC layouts was presented. As discussed in Section 5.5, one of the key limitations of the Annealed-RL and RL-SA methodologies lies in the fact that a distinct model must be trained for each specific circuit component count. In contrast, recent advancements, such as those presented in [101], provide a compelling foundation for addressing this limitation. Specifically, GNN-based techniques demonstrate great potential in enhancing generalization by producing optimal floorplans across diverse circuit topologies.

Furthermore, the previous approach relied on topological models to capture the geometric relationships among circuit components. While effective in some scenarios, this representation inherently constrained the placement of components within the chip canvas, limiting design flexibility and complicating the implementation of geometric

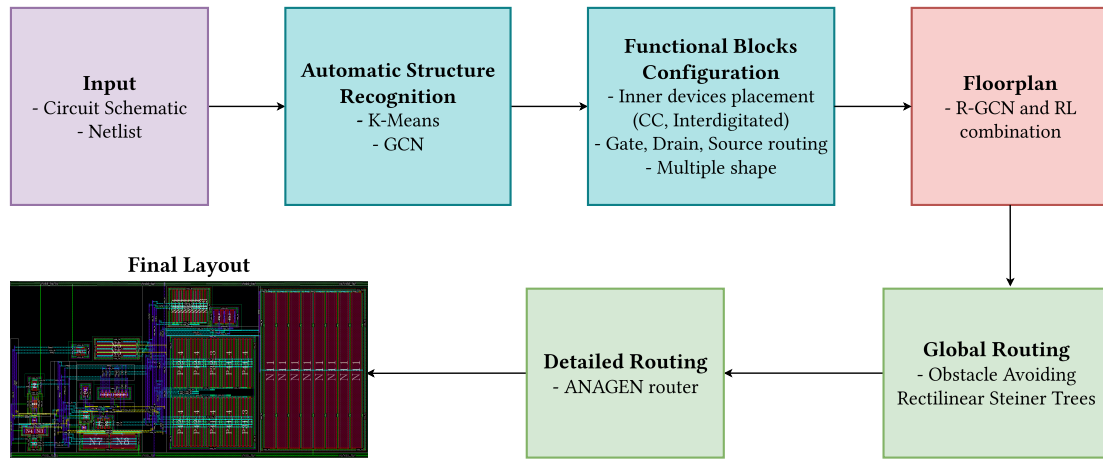


Figure 6.1: Overview of the automatic layout pipeline enhanced with the R-GCN RL floorplanning algorithm.

constraints, especially for more complex circuits. To address these challenges, adopting an absolute placement strategy offers a more suitable alternative for the development of an ideal floorplanning engine.

Finally, an increasing number of RL techniques have led to state-of-the-art outcomes in digital layout applications [4, 31, 93, 94, 186], confirming our intuition to use such methodology and calling for further exploration in the *analog* layout domain.

In particular, we train an R-GCN model to predict rewards for circuit placements. Once trained, it serves as an encoder of circuit, device, and geometric constraints for RL agent use to guarantee optimal generalization capabilities. The detailed circuit data information is then combined by the RL agent with spatial, pixel-level, encodings from a CNN to comprehensively describe circuit and problem characteristics. The agent policy is designed to learn to select *optimal shapes and positions* for components while ensuring *no overlaps and adherence to constraints* such as symmetry and alignment. To our knowledge, this is the first time such approach is proposed.

We validate our novel method on circuits of increasing complexity, showing that our approach outperforms traditional metaheuristics and existing RL-based methods in terms of area, proxy wirelength, speed and cost metrics.

Eventually, we integrate this methodology into ANAGEN in the same way as in Chapter 5. Our enhanced procedural layout generation pipeline proves to consistently shorten design timeframes while matching and even surpassing the quality of manually designed layouts. The workflow overview is depicted in Figure 6.1.

6.2 Methodology

6.2.1 Reinforcement Learning Formulation

To address the limitations of the floorplanning algorithms discussed in Section 5.3.2 and Section 5.3.3, we propose a novel approach devoted at developing a unified agent capable of generating floorplans for circuit designs of any input size, topology, or type. To achieve this, we integrate an R-GCN model for chip representation learning with an RL agent.

The R-GCN model encodes the structural and topological information of the circuit and structures contained in it, providing the RL agent with an enriched state representation. The agent then selects shapes for circuit components and places them onto a discretized grid, which serves as a representation of the layout space.

The floorplanning MDP is defined as follows:

- **States:** The state s_t combines detailed information encoded by the R-GCN model, including the circuit graph g , current instance n_k 32-dimensional embeddings, and local feature maps extracted at the pixel level by a CNN. The latter consist of a 32×32 grid representation

$$f_g \in \{0, 1\}^{32 \times 32},$$

and two reward-related masks,

$$f_{ds}, f_w \in [0, 1]^{32 \times 32},$$

showing increases in the placement empty space and wirelength proxy metric, similar to [93]. Additionally, three positional masks

$$f_p \in \{0, 1\}^{3 \times 32 \times 32},$$

also used for action masking, delineating the admissible placement cells for the next block, given the three possible shapes and adherence to non-overlapping and optionally defined spatial constraints.

- **Actions:** The action a_t at time step t consists in selecting one of three possible shapes for the current block b_t , which have been precomputed using the technique described in Section 5.3.1, and determining the grid cell to place its lower left corner.
- **Rewards:** We define a partial reward r_t as the negative increase of proxy wirelength and empty space in the floorplan after a_t . The end of episode reward is instead equal to the negative weighted sum of floorplan's area, half-perimeter wirelength and, if specified, discrepancy w.r.t. the target aspect ratio.

6.2.2 R-GCN Circuit Representation Learning

An optimal floorplanning algorithm should be capable to generalize its performances across various circuit configurations and constraints. R-GCNs emerge as a natural choice for this, thanks to their permutation invariance that allows them to handle graphs of varying dimensions and structures. Additionally, their proven effectiveness at distilling connectivity between nodes into low-dimensional vector embeddings make them well-suited for capturing information encapsulated in diverse circuit designs.

Works as [4, 96, 126] proposed to pre-train a GNN model on the supervised task of predicting rewards of input circuit graphs. By aligning training tasks with the RL agent's goals, circuit embeddings produced for subsequent stages capture meaningful signals, enhancing RL agent's decision-making with augmented generalization capabilities.

In our setting, shown in Figure 6.2, circuits are represented as undirected graphs where each node v_i corresponds to a functional block or single device. The edges e_i represent relationships between nodes (u, v) , which can be connectivity (if they are connected

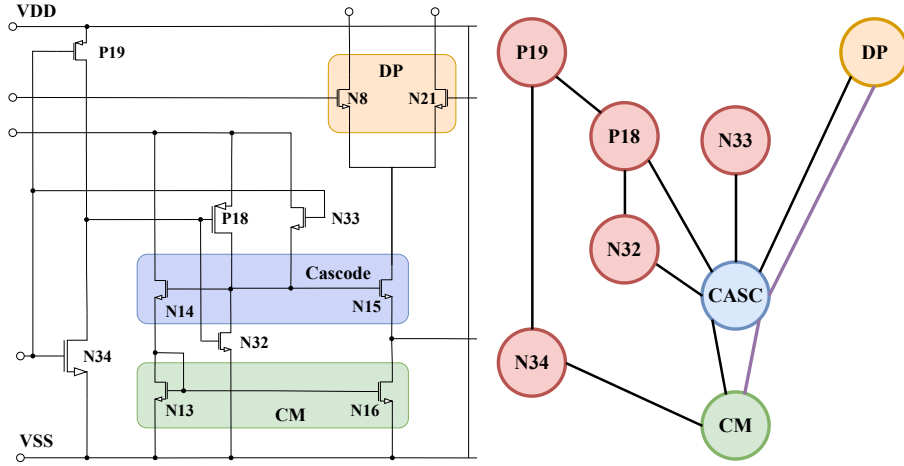


Figure 6.2: 8-structure OTA circuit schematic with its graph representation. Violet edges are for vertical alignment and black for connectivity. Nodes are colored according to the functional block type.

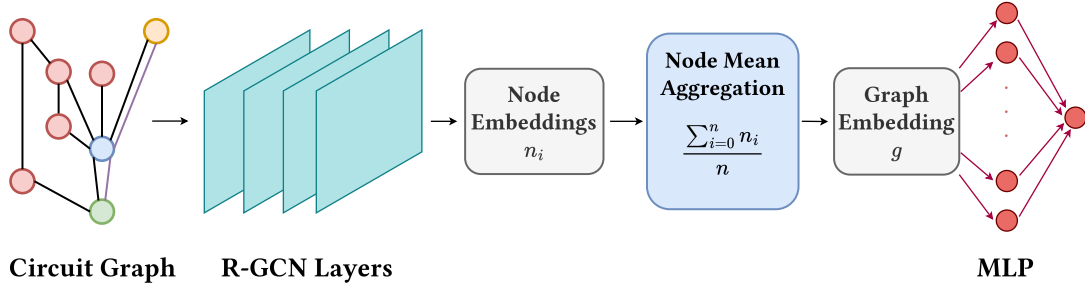


Figure 6.3: R-GCN architecture for circuit reward prediction.

in the netlist), horizontal or vertical alignment, or horizontal or vertical symmetry. A node feature vector $x_u \in X$ includes the block area, internal parameters like transistor or resistance stripe width, terminals routing direction, pin counts, and a 28-dimensional one-hot encoding of the block's functional structure (e.g., current mirror, differential pair, cascode, etc.).

R-GCN Pre-Training Setup

Figure 6.3 shows the R-GCN model architecture, which consists of 4 R-GCN layers followed by a node mean aggregation block to produce the whole graph embedding. This is then fed into 5 FC layers to predict the reward value.

The R-GCN training dataset comprises 21600 floorplans and corresponding reward values, generated in 40 hours by optimizing placement w.r.t. area and proxy wirelength metrics using a mixture of SA, GA, and PSO. The circuits of interest vary in size and complexity, including OTAs, bias circuits, drivers, level shifters, clock synchronizers, comparators, and oscillators. Moreover, we ensured a balance between constrained and unconstrained floorplans.

Finally, the supervised model is trained to minimize the mean squared error between the ground truth and predicted reward associated to the input circuit graph. Other R-GCN and training hyperparameters are listed in Table 6.1. The overall training time took 4

Table 6.1: R-GCN and RL models hyperparameters.

R-GCN Model		RL Model	
R-GCN units	32	Epochs	9
Learning rate	7.9e-4	Learning rate	1.2e-4
Dropout	0.047	Batch size	32
Optimizer	ADAM	Vectorized environments	16
Weight Decay	7.6e-4	Steps per update	256
β_1, β_2	0.9, 0.999	Optimizer	ADAM
FC Neural Network		PPO Loss	
Input normalization	tanh	Clip ϵ	0.2
Activation	ReLU	Discount factor γ	0.9995
Hidden layer neurons	[32,16,8,4,1]	Value Coeff.	0.5
Loss	MSE	Entropy Coeff.	1.3e-6

minutes on a single Nvidia A30 GPU.

6.2.3 R-GCN RL-based Floorplanning

Given the R-GCN trained model, we can employ it as circuit encoder for the RL agent and eventually enhance transfer learning capabilities of the whole methodology. To do so, it is sufficient to remove the final FC layers and then use the remaining part of the model as encoder of the policy network. Figure 6.4 provides an overview of the RL architecture. Eventually, we decided to train the RL agent using a masked version of PPO in order to properly apply the positional mask.

Designing Action Space and Masking

The layout canvas on which a physical engineer would place circuit components can be transformed from a finely detailed to a more concise grid representation. To emulate the flexibility of human design and provide the RL agent with similar degrees of freedom, we let the agent to not only select in which cell to place the lower left corner of a block, but also to decide its shape.

However, large action spaces can hinder RL convergence towards an optimal policy. Therefore, we discretize the layout canvas into a 32×32 grid f_g , effectively balancing performance and accuracy by containing the action space while ensuring constraints satisfaction. The grid height H and width W are computed as:

$$W = H = \sqrt{\frac{\sum_{i=1}^m A_i}{Ar_{\max}}}, \quad (6.1)$$

being A_i the area of the i^{th} circuit device and $Ar_{\max} = 11$ the maximum empirically derived aspect ratio for a floorplan. Since both H and W depend on the size of each functional block, this design choice accommodates any complex circuit placement. As mentioned, the agent can choose from 3 candidate shapes for a circuit structure. Combined with selecting the cell for placing the lower-left corner of a block, this results in an

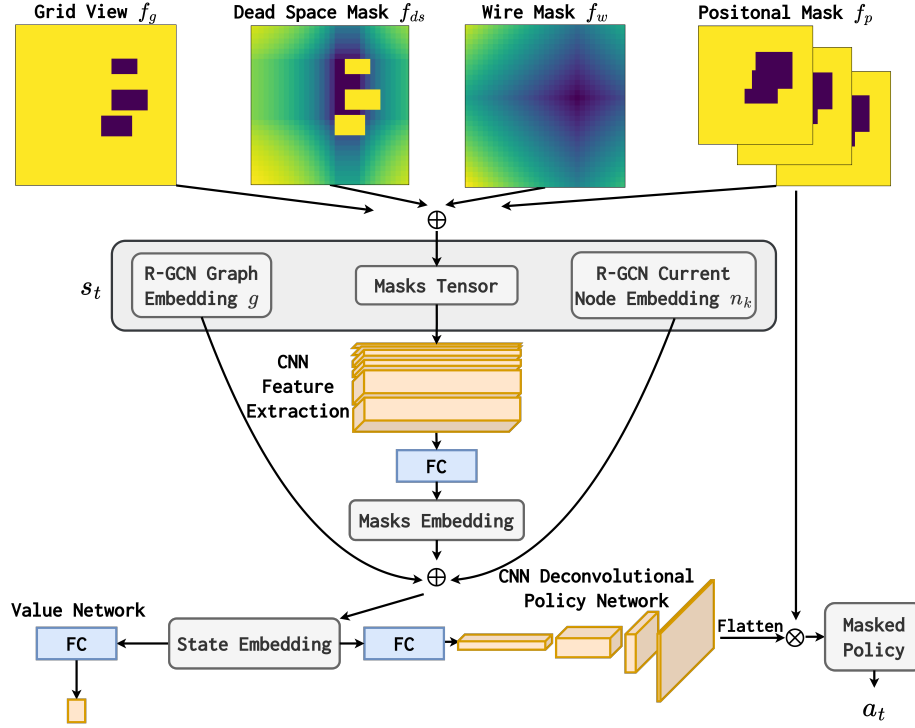


Figure 6.4: Overview of the RL model, enriched with CNN-based feature extractor and policy network.

action space $\mathcal{A}$ of size $3 \times 32 \times 32 = 3072$. To prevent further escalation of action space dimensionality, we use a heuristic inspired by [126], which arranges block placement in order of decreasing size.

Another crucial aspect for our floorplanning methodology is the capability to properly handle fundamental positional requirements such as symmetry, alignment, and guarantee the absence of device overlap. Works as [72] prove that policy gradient algorithms can take advantage of masking procedures to avoid selecting invalid actions based on state information. Therefore, at each episode step, we generate three *positional masks* f_p , one for each possible candidate shape. These masks are obtained by combining two binary matrices: one representing partial placement and the other symmetry or alignment constraint masks. In the first one, a value of 1 indicates an available cell, while 0 signifies an already occupied one. The second one designates with 1s where the corresponding constraint would be satisfied (based on the placement of blocks belonging to a constraint group and the corresponding symmetry or alignment axis) and 0s where it would not. We map the real sizes of each circuit instance (w, h) without approximation, where:

$$w_g = \left\lceil \frac{w \times 32}{W} \right\rceil, \quad h_g = \left\lceil \frac{h \times 32}{H} \right\rceil \quad (6.2)$$

are the respective scaled width and height on the grid.

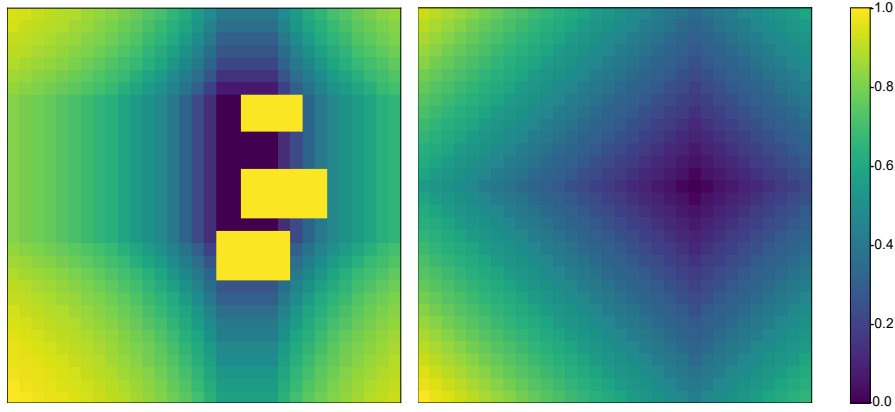


Figure 6.5: Dead space (left) and wire (right) masks. Darker areas highlight higher rewards regions in the case a block is placed there.

State Design

Given that the R-GCN model does not supply detailed location information for specific circuit instances during an episode, we enrich the agent’s state representation. As mentioned in Section 6.2.1, this is achieved by combining the current node, i.e. block b_t , and graph, i.e. circuit, embeddings n_k and g with 6 additional grid-based masks: f_g , f_w , f_{ds} and f_p , which reflect partial placements, wire and dead space, i.e. empty space, increases, and valid positions. This augmented state aids the RL agent in identifying the optimal placement to optimize rewards.

Lai et al. [93] first proposed this approach, using f_p , f_g and computing f_w as the increase in proxy wirelength when placing b_t in a specific position. Our approach extends the positional mask f_p to account for multiple device shapes and introduces the *dead space mask* f_{ds} , a normalized, continuous matrix indicating the increase in empty space if b_t is placed in a certain location. To construct f_{ds} , we iterate over all available cells on the placement grid, compute the resulting dead space from placing b_t , and subtract the previous dead space value. Already occupied locations are set to the maximum increment 1 to mask invalid positions. Figure 6.5 provides a visual example of f_w and f_{ds} .

Policy Design

We create a CNN-based encoder network that extracts features from the grid masks in order to combine them with the vectorized R-GCN embeddings. Treating grid masks as analogous to images, as suggested by [31], is a natural choice to exploit CNNs effectiveness in converting their representation into informative embeddings.

For this reason, we concatenate the masks forming a single tensor of dimension $6 \times 32 \times 32$ and feed them to a CNN. The convolution layers use a 3×3 kernel size with stride of 1, padding of 1 and 16, 32, 32, 64, 64 filter channels, followed by one FC layer to produce a 512-dimensional embedding vector. Finally, after concatenating the R-GCN embedding and CNN outputs, this compact state representation is fed into both value and policy network. The policy includes a single FC layer that converts the input vector to a 512-dimensional one, 3 deconvolution layers with a kernel size of 4×4 , a stride of 2, a padding of 1 and 32, 16, and 8 filter channels. The policy then generates a probability

distribution over actions, allowing the agent to jointly decide the shape and location of the new device to be placed.

Reward Shaping

As delineated in Section 6.2.1, the agent’s goal is to minimize two primary metrics: area occupation and HPWL. HPWL is defined in the same way as shown in Equation 5.1, while the dead space DS in a floorplan F is computed as:

$$DS = 1 - \frac{\sum_{i=1}^m A_i}{F_{area}}. \quad (6.3)$$

To better guide the agent during an episode rollout, we provide intermediate rewards r_t based on the increase in partial floorplan dead space and HPWL, computed from the currently placed instances after possibly taking action a_t . The intermediate reward is defined as:

$$r_t = -(\Delta_{ds} + \Delta_{HPWL}), \quad (6.4)$$

where $\Delta_{ds} = DS_t - DS_{t-1}$ and $\Delta_{HPWL} = HPWL_t - HPWL_{t-1}$.

Given the optional constraint of a fixed aspect ratio for the final floorplan, we define the agent’s end of episode reward as at the final time step T :

$$r_T = -\left(\alpha \frac{F_{area}}{\sum_{i=1}^m A_i} + \beta \frac{HPWL}{HPWL_{min}} + \gamma (Ar^* - Ar)^2\right). \quad (6.5)$$

Here, α , β and γ are empirically set weights (1, 5, and 5, respectively) determined through extensive experimentation. These weights balance area, wirelength and fixed outline error terms in terms of their magnitude and impact on the final floorplan quality. $HPWL_{min}$ is the minimum HPWL value estimated through a metaheuristic-based simulation for standardization while Ar^* and Ar denote the target and current floorplan aspect ratios, respectively, as defined in Section 5.2.

To ensure adherence to predefined design constraints, a penalty mechanism is incorporated into the reward function. Specifically, whenever the generated floorplan violates any constraint, the agent is penalized with a reward of -50 . This heavy penalty is designed to strongly discourage the agent from repeating the same actions that led to any kind of violations, thereby guiding the agent toward more viable behaviors in subsequent iterations.

RL Training Schedule

Our methodology is designed to develop a single RL agent capable of generating optimal floorplans for a wide array of circuit types and constraints. To achieve this, we apply a hybrid curriculum learning (HCL) approach, described in [187], which incrementally presents the agent circuits of growing complexity. Starting with smaller circuits, we interleave them with random sampling of new circuit instances and constraints. This method maintains the agent’s exposure to complex scenarios and prevents the loss of previously acquired knowledge, thereby enhancing transferability to new, unseen instances. The circuits used to train the RL are 3 operational transconductance amplifiers (OTAs) and 2 bias ones, respectively encompassing 3, 5, 8, 3 and 9 blocks, to ensure enough diversity in the data. Each of them is part of an Infineon 130nm Commercial Technology suite.

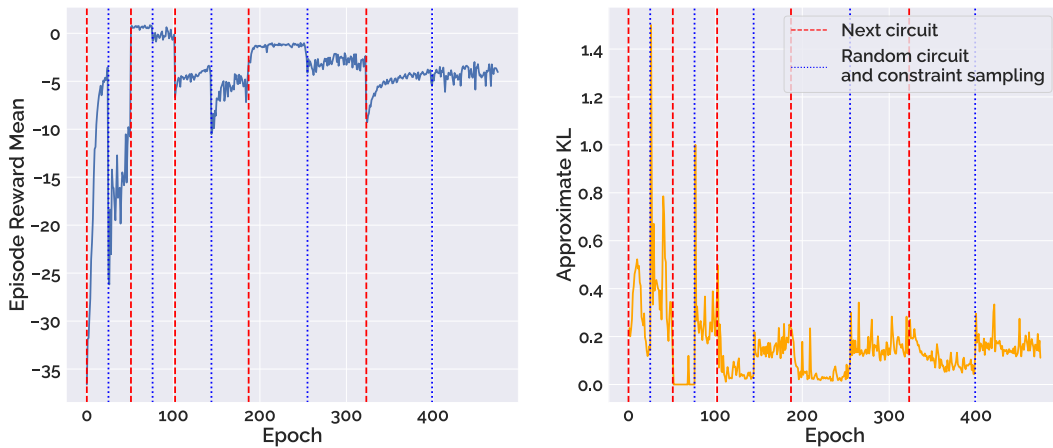


Figure 6.6: Episode reward mean and approximate Kullback-Leibler divergence during RL agent’s HCL training schedule.

6.2.4 Routing and Final Layout Generation

Once a feasible floorplan is generated, we construct an OARSMT for each net to minimize wirelength and avoid obstacles, in the same way as presented in Section 5.2.2. Nevertheless, unlike the previous approach, which required congestion estimation to reserve space for routing channels, this new method yields routing ready floorplans. The global routing tree is segmented into conduits, detailing connections and layers, guiding ANAGEN’s router to finalize circuit connections.

This improvement enhances ANAGEN’s capabilities and potentially makes it competitive with state-of-the-art techniques [23, 41] by allowing the RL placement engine to focus on higher-level device arrangement, while ANAGEN handles low-level place and route tasks.

6.3 Experimental Results

Our floorplanning pipeline is built in Python 3.9, using DGL [178] and stable-baselines3 libraries for implementing respectively the R-GCN and RL models.

6.3.1 RL Training Setup

In order to foster policy’s robustness and its convergence time we use 16 parallel environments to gather multiple experiences. Using Optuna optimization framework [3], we fine-tuned our RL agent’s parameters, the results of which are summarized in Table 6.1.

In line with the HCL approach presented in Section 6.2.3, our strategy initially focuses on allowing the agent to achieve convergence for a given circuit instance. To do so, we avoid to introduce new constraints or circuits during the first half of the total episodes for optimizing the current circuit. Once this first phase is over, we start to sample a new circuit instance and a new constraint respectively with probability $p_{\text{circuit}} = 0.5$ and $p_{\text{constraint}} = 0.3$.

Figure 6.6 illustrates the benefits of the HCL procedure for convergence towards a stable policy. By engaging with this method, the agent acquires strong generalization

abilities, allowing it to recover optimal rewards on unseen circuits and gaining robustness during random circuit sampling. Moreover, the low displayed approximate KL values further confirm agent’s consistent converge towards an optimal policy. Training the RL agent on a single Nvidia A30 GPU took 12 hours and 42 minutes.

6.3.2 Comparison Against Baselines

We evaluate our approach against established metaheuristics such as SA, the same methodology used by state-of-the-art automatic layout generator [41], PSO, GA as well as RL-SA and Annealed-RL methods presented in Chapter 5. Unfortunately, the circuits of interest are incompatible with other fully automated layout generation engines [23, 41] due to technology constraints.

Performances are measured on 3 industrial designs from the RL training set and 3 unseen ones, aiming to validate transferability with *zero-shot* and *few-shot* fine-tuning on novel circuits. To better clarify, few-shot learning involves refining a pre-trained RL agent by continuing its training on a specific problem or circuit instance, optimizing it for that specific context. Targeting a fair comparison, congestion-aware device spacing is applied to all other approaches to allocate sufficient room for routing channels, as our methodology provides routing-ready floorplans. No constraints are imposed on any circuit.

Table 6.2 lists the inter-quartile mean (IQM) and standard deviation values for algorithm computational runtime, floorplan HPWL, dead space, and associated reward. We emphasize the best results in bold, and highlight the second and third best rewards in italic underlined and underlined only, respectively. Unseen circuits are represented as gray rows.

As shown, our novel approach surpasses the baselines in terms of floorplan reward in all scenarios with proper fine-tuning, and in 4 out of 6 cases at zero-shot while significantly improving runtime. Moreover, our methodology demonstrates remarkable transfer capability across unseen and more complex designs, even without additional fine-tuning. On new circuits, HPWL and dead space percentage are reduced by 38.7% and 66.8%, respectively, compared to past techniques. Moreover, few-shot fine-tuning improves results compared to the zero-shot model for the same number of iterations. While training time is significant for few-shot solutions, once fine-tuned, the model doesn’t require retraining, making single floorplan generation runtimes comparable to zero-shot ones and outperforming other approaches.

6.3.3 Evaluation of Complete Layouts

We plug our novel floorplanning algorithm into the pipeline proposed in the framework proposed in Chapter 5 and validate its effectiveness by comparing the completed layouts of a 3-block OTA, 9-block bias, and 17-block driver from [39] against their manually designed counterparts. Notably, the manual Bias layout was crafted without using ANAGEN, unlike the other two. The metrics of interest involve floorplan’s area, dead space, and time required to produce a DRC and LVS clean layout.

Results in Table 6.3 underscore algorithm’s capability to quickly generate valid floorplans, facilitating faster design iteration. The OTA and driver circuits exhibit improved area and dead space metrics, although the bias circuit has higher dead space percentages.

Table 6.2: Comparative analysis of R-GCN and RL method, across various fine-tuned setups, versus previous techniques.

Circuit	# Struct.	Metric	R-GCN RL o-shot	R-GCN RL 1-shot	R-GCN RL 100-shot	R-GCN RL 1000-shot	SA	GA	PSO	RL-SA	Annealed-RL
OTA-1	5	Runtime (s)	0.06 $\pm$ 0.24	7.97 $\pm$ 0.31	15.88 $\pm$ 0.3	174.13 $\pm$ 3.55	0.91 $\pm$ 0.01	4.58 $\pm$ 0.01	6.86 $\pm$ 0.03	1.03 $\pm$ 0.01	53.08 $\pm$ 0.66
		Dead space (%)	57.5 $\pm$ 1.25	47.19 $\pm$ 1.7	47.38 $\pm$ 4.32	43.93 $\pm$ 1.82	49.79 $\pm$ 4.28	53.95 $\pm$ 5.74	46.75 $\pm$ 3.16	51.19 $\pm$ 4.2	49.05 $\pm$ 3.93
		HPWL (μm)	150.62 $\pm$ 9.0	196.89 $\pm$ 11.03	175.94 $\pm$ 18.61	131.34 $\pm$ 23.76	166.03 $\pm$ 18.72	175.05 $\pm$ 25.85	164.32 $\pm$ 11.05	164.17 $\pm$ 17.81	166.96 $\pm$ 19.75
		Reward	-2.37 $\pm$ 0.42	-3.35 $\pm$ 0.46	-2.61 $\pm$ 0.87	-0.21 $\pm$ 0.91	-2.04 $\pm$ 0.68	-2.46 $\pm$ 1.13	<u>-1.86</u> $\pm$ <u>0.47</u>	<u>-1.97</u> $\pm$ <u>0.67</u>	-2.05 $\pm$ 0.82
OTA-2	8	Runtime (s)	0.14 $\pm$ 0.01	9.2 $\pm$ 0.09	28.0 $\pm$ 0.2	287.03 $\pm$ 0.88	1.09 $\pm$ 0.03	4.98 $\pm$ 0.02	7.30 $\pm$ 0.03	1.3 $\pm$ 0.1	94.64 $\pm$ 1.8
		Dead space (%)	43.77 $\pm$ 5.27	33.18 $\pm$ 1.95	33.18 $\pm$ 2.34	35.57 $\pm$ 1.62	57.5 $\pm$ 4.08	57.3 $\pm$ 6.77	55.3 $\pm$ 4.66	55.68 $\pm$ 4.48	54.17 $\pm$ 5.4
		HPWL (μm)	202.19 $\pm$ 22.89	154.27 $\pm$ 5.79	164.14 $\pm$ 9.37	168.29 $\pm$ 7.52	244.42 $\pm$ 41.68	237.3 $\pm$ 30.17	226.27 $\pm$ 31.32	229.17 $\pm$ 23.26	234.95 $\pm$ 37.39
		Reward	-2.52 $\pm$ 0.89	-0.68 $\pm$ 0.23	<u>-0.96</u> $\pm$ <u>0.31</u>	-1.16 $\pm$ 0.28	-3.97 $\pm$ 1.54	-3.68 $\pm$ 1.33	-3.22 $\pm$ 1.17	-3.51 $\pm$ 0.85	-3.62 $\pm$ 1.38
Bias-1	9	Runtime (s)	0.16 $\pm$ 0.15	8.96 $\pm$ 0.51	26.17 $\pm$ 0.22	303.62 $\pm$ 3.13	1.23 $\pm$ 0.01	5.4 $\pm$ 0.37	7.74 $\pm$ 0.48	1.57 $\pm$ 0.12	72.06 $\pm$ 1.27
		Dead space (%)	53.16 $\pm$ 3.97	56.85 $\pm$ 1.54	59.93 $\pm$ 5.64	45.52 $\pm$ 7.22	67.79 $\pm$ 4.74	73.28 $\pm$ 5.06	68.22 $\pm$ 4.77	67.57 $\pm$ 4.29	67.58 $\pm$ 4.68
		HPWL (μm)	271.51 $\pm$ 39.03	313.63 $\pm$ 23.50	321.76 $\pm$ 24.11	191.91 $\pm$ 64.07	288.97 $\pm$ 40.05	269.7 $\pm$ 47.05	329.14 $\pm$ 55.57	285.19 $\pm$ 39.65	288.46 $\pm$ 39.74
		Reward	<u>-5.68</u> $\pm$ <u>1.51</u>	-7.40 $\pm$ 0.87	-7.70 $\pm$ 0.83	-2.53 $\pm$ 2.60	-6.61 $\pm$ 1.7	-6.29 $\pm$ 2.25	-8.07 $\pm$ 2.29	-6.5 $\pm$ 1.69	-6.65 $\pm$ 1.65
RS Latch	7	Runtime (s)	0.11 $\pm$ 0.0	6.82 $\pm$ 1.09	17.65 $\pm$ 0.05	166.22 $\pm$ 0.75	0.99 $\pm$ 0.01	4.8 $\pm$ 0.01	6.71 $\pm$ 0.03	1.16 $\pm$ 0.03	42.96 $\pm$ 0.6
		Dead space (%)	57.85 $\pm$ 2.54	53.17 $\pm$ 7.73	59.87 $\pm$ 2.25	33.76 $\pm$ 4.41	65.62 $\pm$ 4.32	69.99 $\pm$ 5.56	65.57 $\pm$ 4.27	63.84 $\pm$ 4.57	61.56 $\pm$ 3.63
		HPWL (μm)	108.0 $\pm$ 3.22	118.06 $\pm$ 13.34	128.63 $\pm$ 16.53	97.55 $\pm$ 7.58	127.99 $\pm$ 17.01	128.26 $\pm$ 15.95	127.92 $\pm$ 16.62	124.11 $\pm$ 15.17	120.22 $\pm$ 15.1
		Reward	<u>-4.04</u> $\pm$ <u>0.31</u>	<u>-4.63</u> $\pm$ <u>1.27</u>	-5.55 $\pm$ 1.26	-2.34 $\pm$ 0.62	-5.44 $\pm$ 1.58	-5.58 $\pm$ 1.59	-5.39 $\pm$ 1.56	-5.03 $\pm$ 1.44	-4.69 $\pm$ 1.39
Driver	17	Runtime (s)	0.25 $\pm$ 0.0	11.6 $\pm$ 0.1	79.64 $\pm$ 0.81	814.81 $\pm$ 7.76	1.85 $\pm$ 0.01	6.63 $\pm$ 0.06	10.4 $\pm$ 0.06	2.24 $\pm$ 0.07	155.08 $\pm$ 3.53
		Dead space (%)	63.5 $\pm$ 4.79	62.49 $\pm$ 1.73	59.71 $\pm$ 3.35	48.17 $\pm$ 4.11	71.44 $\pm$ 7.22	73.3 $\pm$ 7.16	70.64 $\pm$ 6.24	70.8 $\pm$ 7.82	69.36 $\pm$ 8.42
		HPWL (μm)	1794.5 $\pm$ 173.3	1811.61 $\pm$ 163.9	1811.04 $\pm$ 133.7	1419.57 $\pm$ 61.88	1981.07 $\pm$ 319.51	2192.15 $\pm$ 448.45	2152.62 $\pm$ 294.24	1862.99 $\pm$ 372.89	1941.85 $\pm$ 516.42
		Reward	-7.43 $\pm$ 1.29	-7.43 $\pm$ 1.05	<u>-7.26</u> $\pm$ <u>0.84</u>	-4.43 $\pm$ 0.53	-8.55 $\pm$ 2.41	-10.23 $\pm$ 3.23	-9.66 $\pm$ 2.23	-7.69 $\pm$ 2.72	-8.31 $\pm$ 3.73
Bias-2	19	Runtime (s)	0.34 $\pm$ 0.04	12.74 $\pm$ 1.4	88.73 $\pm$ 1.01	849.86 $\pm$ 3.56	2.07 $\pm$ 0.01	6.91 $\pm$ 0.02	11.6 $\pm$ 0.14	2.42 $\pm$ 0.02	244.62 $\pm$ 3.27
		Dead space (%)	68.49 $\pm$ 6.81	56.01 $\pm$ 4.39	57.36 $\pm$ 3.22	45.12 $\pm$ 2.66	73.68 $\pm$ 4.2	70.36 $\pm$ 4.71	69.32 $\pm$ 5.75	74.89 $\pm$ 4.76	70.88 $\pm$ 6.37
		HPWL (μm)	3375.84 $\pm$ 235.86	2967.92 $\pm$ 174.56	2780.59 $\pm$ 225.7	2141.84 $\pm$ 150.64	2896.52 $\pm$ 177.68	3501.24 $\pm$ 490.11	3346.66 $\pm$ 498.34	2872.94 $\pm$ 369.75	2854.06 $\pm$ 366.79
		Reward	-5.95 $\pm$ 0.67	<u>-4.34</u> $\pm$ <u>0.7</u>	<u>-3.65</u> $\pm$ <u>0.57</u>	-1.43 $\pm$ 0.47	-5.17 $\pm$ 0.68	-6.26 $\pm$ 1.65	-5.74 $\pm$ 1.67	-5.08 $\pm$ 1.3	-4.36 $\pm$ 1.34

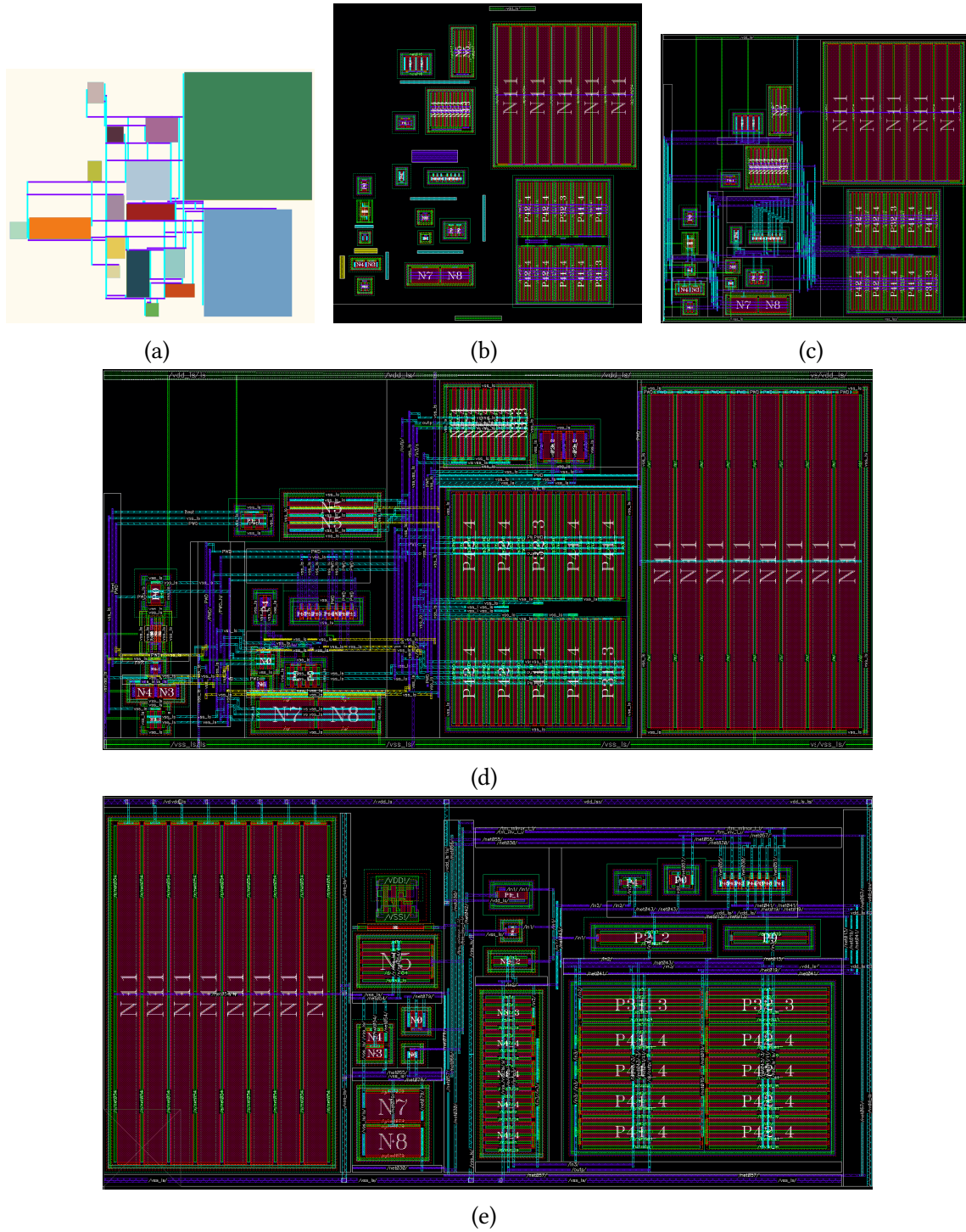


Figure 6.7: (a) RL-generated placement and OARSMT global routing, (b) post-adjustment floorplan and channels definitions, (c) resulting layout, (d) post-manual refinement optimized layout, and (e) full manual design.

Table 6.3: Comparison of area, dead space, and layout generation time between our automated method and manual design.

Circuit	Method	Area (μm^2)	Dead space (%)	Template generation time (s)	Manual improvement time (h)	Final layout generation time (h)
OTA	Ours	228.6 (-14.1%)	30.01 (-5.98%)	111.0	0.17	0.20 (-97.5%)
	Manual	266.0	31.92	-	-	8
Bias-1	Ours	515.6 (+52.1%)	54.01 (+8.68%)	127.1	1	1.04 (-87.0%)
	Manual	247.1	49.32	-	-	8
Driver	Ours	3584.7 (-2.43%)	38.78 (-3.82%)	456.3	20	20.13 (-37.1%)
	Manual	3674.0	40.32	-	-	32

This latter layout in fact benefits from the absence of ANAGEN’s routing constraints, resulting in smaller area occupation.

Finally, Figure 6.7 shows a direct comparison between the automatically generated driver layout from our methodology and the manually designed one. Figures 6.7a and 6.7b illustrate the outcome of our floorplanning and global routing algorithm, providing clear guidance for physical design engineers on the expected wireflow. In more complex layouts, manual refinement of routing channels guided by the OARSMT is still necessary to accommodate ANAGEN’s routing sensitivities. However, for simpler layouts like the OTA example, routing channel generation was fully automated. Improving this aspect remains a focus for future work.

6.4 Discussion

In this chapter, we presented a novel methodology for analog IC floorplanning that combines R-GCNs with RL. The proposed approach ensures compliance with fundamental layout constraints, such as alignment, symmetry, and overlap avoidance, while maintaining flexibility and adaptability. A key strength of our methodology is its ability to generalize and transfer knowledge across different circuit types, including previously unseen designs. The generated floorplans not only surpass the performance of established baselines but, when integrated into a procedural layout generation pipeline, yield complete layouts of comparable quality to those crafted by human designers, all within significantly reduced runtimes.

Future work will focus on refining the selection of reward weights by employing advanced optimization techniques to better balance competing objectives. Additionally, we aim to enhance the floorplanning algorithm by incorporating detailed routing information, enabling the placement process to be further conditioned for routing efficiency. This integration will guide the algorithm towards producing even more optimized layouts that minimize routing complexity while adhering to design constraints.

Chapter 7

Beam Search Enhanced Reinforcement Learning for Analog ICs Floorplanning

In this chapter, we present a hybrid method that combines the approach presented in Chapter 6 with a beam search (BS) strategy. The BS algorithm enhances the agent’s inference process, allowing for the generation of flexible floorplans by accommodating various objective weightings, and addressing congestion without the need for policy retraining or fine-tuning. Moreover, the RL agent’s generalization ability stays intact, along with its efficient handling of circuit features and constraints. Experimental results show $\sim 5 - 85\%$ improvement in area, dead space and HPWL compared to a standard RL application, along with higher rewards for the agent. Moreover, performance and efficiency align closely with those of existing state-of-the-art techniques.

A brief summary regarding the BS algorithm and the new *BS-RL* solution are depicted in Section 7.2, while a comparison with previous works is presented in Section 7.3. Eventually, a discussion regarding the achieved results and future directions is provided in Section 7.4. The content of this chapter has been published in [38, 149].

7.1 Introduction

As seen in previous chapters, RL methods proved to be highly effective in generating well compact and routing optimized floorplans. Nevertheless, although optimal results can be achieved with thorough training, policy fine-tuning is typically required to achieve the best results, as reported in Section 6.3, leading to the demand for more data and expensive computational resources.

For this reason, we propose an enhancement of the inference process that builds a search tree to represent the state space of the RL agent from the previous chapter, then periodically prunes it using the BS algorithm [114] to balance exploration and exploitation. Eventually, following such approach leads to better final results than a 0-shot strategy, without requiring the use of GPUs or a policy update, while scaling comparably.

Some previous works used the BS algorithm to solve the bin-packing problem, which is a generalization of block placement in IC floorplanning. These relied on heuristics and not ML [13, 14]. Similar approaches to ours have been proposed in [32, 49], however,

these exploit BS to perform rollouts and policy updates in order to enhance the agent’s learning ability. Hybrid BS and RL solutions have been applied to enhance each other’s performance [44, 75], but there have not been many applications in EDA, especially in the case of analog layout.

Our solution is instead plug-and-play and allows us to boost the agent’s inference process by leaving untouched its underlying structure. A new objective function is also introduced, allowing users to set the trade-off weights between area and HPWL optimization to adjust the output based on their needs. Moreover, a new congestion management procedure, inspired by [93], ensures a more routing-friendly output and makes the method more suitable for industrial applications.

After a final evaluation of the proposed method in the same testing environment as in Section 6.3, we noticed that the baseline is improved by 5% up to 85% in all metrics comparable to a state-of-the-art fine-tuning approach, and without the use of GPUs.

7.2 Methodology

7.2.1 Beam Search

Beam Search is a heuristic search algorithm that explores a graph in a Breadth-First Search fashion, starting from a source node, and expanding a set composed by the Λ (an integer called *beam width*) most promising successors, for further exploration.

This approach reduces memory and computational requirements, compared to a full search, albeit at the cost of making the search incomplete. Despite this, the algorithm empirically shows good results, providing sub-optimal solutions that are better than the ones obtained with a greedy approach.

7.2.2 Congestion Handling

A key characteristic of high-quality layouts, particularly in analog electronics, is the need to strike a balance between compactness and low congestion. While minimizing layout area by placing modules close to one another is essential, adequate spacing must be maintained to ensure efficient routing. This spacing allows for the optimal placement of wires during the routing phase, minimizing total wire length, alleviating congestion, and ultimately enhancing the overall quality of the layout.

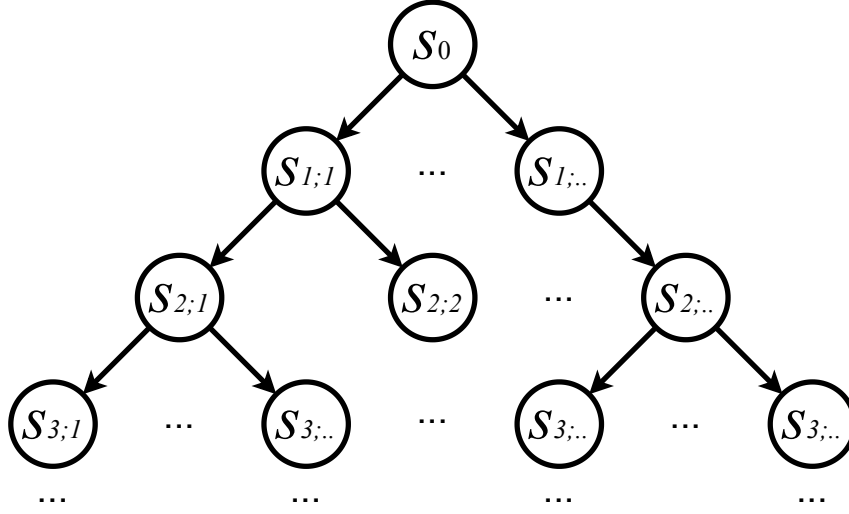
Before detailing how BS has been leveraged to enhance the performance of the RL agent, it is important to introduce the congestion handling mechanism adopted in the following sections.

RUDY, a routing demand estimation method originally proposed in [168], is defined as:

$$\text{RUDY} = \sum_{n \in N} d_n \cdot \mathbb{1}_n(x, y), \quad (7.1)$$

where N denotes the circuit netlist, and $d_n = \frac{WA_n}{NA_n}$ is the wire density of net n , defined as the ratio between the wire area WA_n and the net area NA_n . The function $\mathbb{1}_n$ is an indicator over the x - y plane for each net, defined as:

Figure 7.1: Example state tree space for an RL agent. The root S_0 represents the initial state, at time-step 0. Nodes $S_{t;a}$ represent the states at time-step t , after taking action a .



$$\mathbb{1}_n(x, y) = \begin{cases} 1 & \text{if } 0 \leq x - x_n \leq w_n \wedge 0 \leq y - y_n \leq h_n, \\ 0 & \text{otherwise,} \end{cases}$$

where (x_n, y_n) are the lower-left coordinates of the bounding box containing all the terminals belonging to net n , and w_n, h_n represent the corresponding bounding box width and height, respectively.

RUDY has become one of the most used congestion estimation metrics employed in literature due to its fast and reasonably accurate results while not relying on any specific routing model, making it applicable to a wide variety of use cases.

7.2.3 Beam Search and Reinforcement Learning: *BS-RL*

To increase the state space exploration of the o-shot RL agent from Chapter 6, a search tree is built. As an MDP is inherently stochastic, multiple runs of the algorithm on the same input can yield different sequences of decisions, trajectories, and outcomes.

This process naturally gives rise to a tree-structured solution space, as visualized in Figure 7.1: nodes represent the RL agent's states, and edges denote possible actions taken at each step. Internal nodes correspond to partial floorplanning solutions, while leaf nodes represent complete floorplans. The tree arity q is user-defined and directly determines the expansion rate of the solution space, which grows exponentially with the tree height.

Given the intermediate and end-of-episode reward definitions outlined in Equation 6.4 and Equation 6.5, respectively, every node in the tree is associated with a value ρ defined as follows:

$$\rho = \begin{cases} -\omega_1 \cdot \Delta_t^{ds} - \omega_2 \cdot \Delta_t^{\text{HPWL}} & \text{for internal nodes,} \\ -\omega_1 \cdot \alpha \cdot \frac{1}{1 - \text{DS}_T} - \omega_2 \cdot \beta \cdot \frac{\text{HPWL}_T}{\text{HPWL}_{\min}} & \text{for leaves,} \end{cases} \quad (7.2)$$

where ω_1 and ω_2 are user-defined weights that guide the optimization process by prioritizing either area minimization or HPWL reduction. The terms Δ_t^{ds} and Δ_t^{HPWL} represent the incremental increases in relative dead space and HPWL at time step t , while DS_T and $HPWL_T$ denote the dead space and HPWL values at the final time step T . This new reward formulation allows the optimization process to adapt to specific design priorities, making the solution more customizable and better suited to real-world industrial use cases.

The procedure starts by building the root node $\emptyset$, which represents the initial state s_0 , then q actions are sampled from $\pi^*(s_0)$. After the creation of each level, the tree is pruned with a probability ϵ , that can be chosen by the user to balance exploitation. If this happens, the BS algorithm is executed, leaving (at most) the Λ nodes with the highest ρ on each level. If not, all the nodes that have been created to this point in the tree, and which have not yet been pruned, are maintained.

The tree is then further explored from the remaining nodes in the deepest level created so far. A limit to the width of each level is given as 10^3 nodes, to reduce the computational and memory burden. If the new level exceeds this threshold, BS is executed to downsize the tree. Eventually, the building procedure is repeated in the same way for every children node, until the terminal states are found and the leaves are built.

In order to generate less congested floorplans, the RUDY routing demand estimator can be included in the RL agent pipeline. If the action predicted by the RL agent results in a congestion estimate exceeding a certain threshold c , the pipeline selects the top 60% of actions sampled from the agent's policy π^* , discarding the least probable ones. Each action in the selected subset is then evaluated using the RUDY metric. Among the candidates that satisfy the threshold, the most probable one is selected. However, if none meet the constraint, the action with the lowest RUDY score is chosen instead.

Finally, Figure 7.2 illustrates the iterative process of tree construction and pruning. Once all leaves have been created, the one with the highest ρ is returned, and the final floorplan computed by BS-RL can be retrieved from its state.

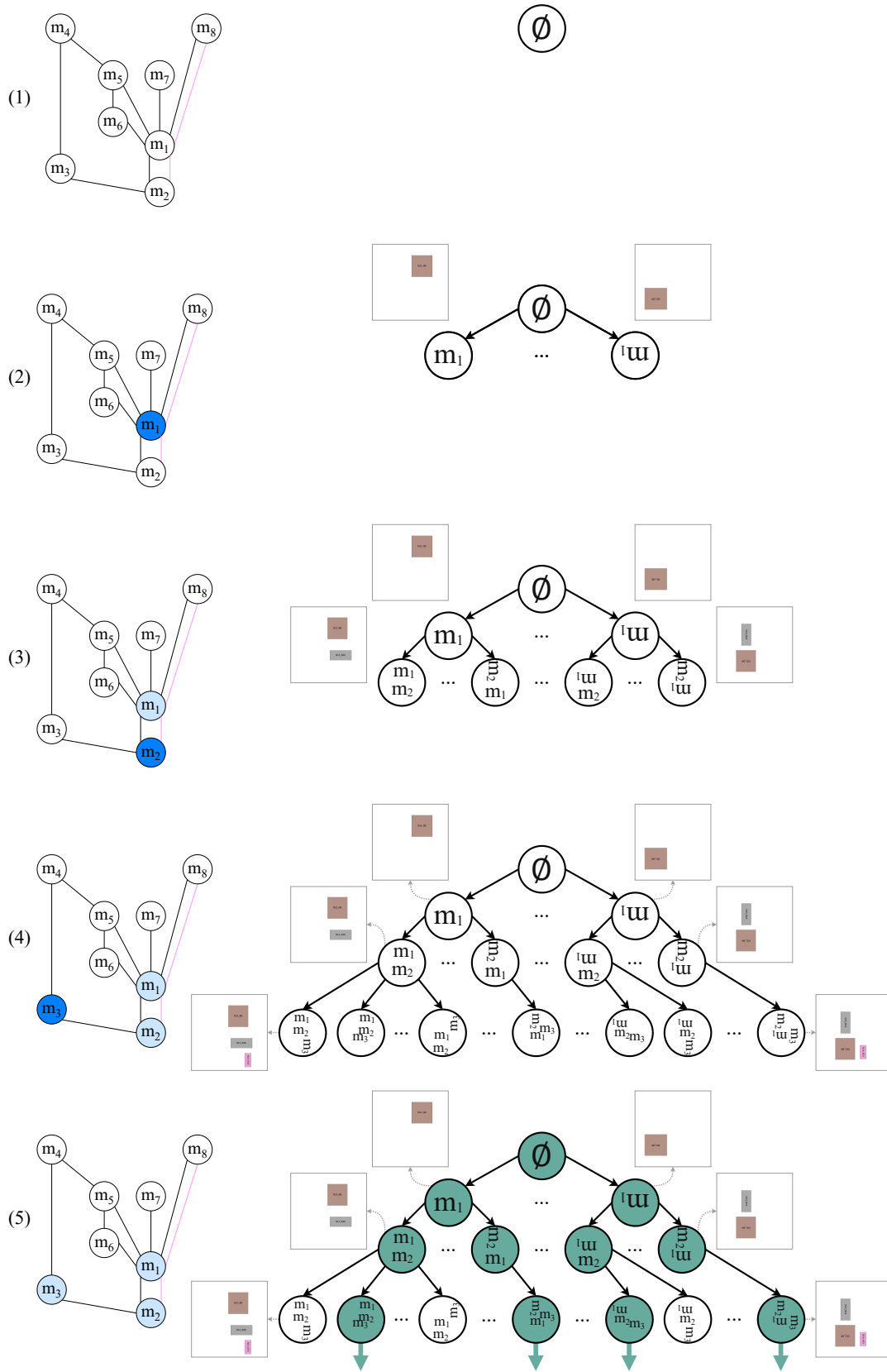
7.3 Experimental Results

7.3.1 Experiment Details

Various floorplanning algorithms have been executed on the same IC designs reported in Section 6.3. In particular, SA is employed as a standard benchmark, while the 0-shot RL agent (*R-GCN RL*) and the 1000-shot fine-tuned version (*R-GCN RL-FT*) are used for comparison with BS-RL.

Every experiment has been repeated 100 times, computing various metrics (Runtime, DS, HPWL, RUDY, and Reward). The results have been aggregated with the IQM, to reduce the influence of outliers, while the inter-quartile range (IQR) measures the spread in the results. A confidence interval analysis has been conducted, by computing 10^4 -bootstrap confidence intervals with confidence level $z = 10^{-5}$. Since each statistical measure fell within the corresponding confidence interval, we can confidently infer a high level of statistical significance.

BS-RL has been instanced using a tree arity $q = 5$, a pruning probability $\epsilon = 0.7$ and a beam width $\Lambda = 10$. Moreover, the optimization biasing hyperparameters ω_1 and ω_2

Figure 7.2: Example of BS execution on the OTA from Figure 5.2, with $\Lambda = 4$.

were both set to 1 during experimentation. Although various configurations were tested, this setup was chosen for evaluation due to its ability to produce good results within a reasonable time frame. Minor adjustments do not significantly alter the outcomes, but future investigations may uncover more interesting and potentially superior results. No symmetry, alignment, or aspect-ratio constraints have been specified for experimentation.

Table 7.1 summarizes the results of these experiments. The gray rows represent IC designs not seen during training by the agent. In BS-RL the congestion threshold is set to ∞ , to fairly compare it with R-GCN RL and R-GCN RL-FT, since they don't implement congestion handling. R-GCN RL[†] and R-GCN RL-FT[†] overcome this issue, and along BS-RL[†] they consider a low congestion threshold of 0.1. The bold values between brackets represent the percentage of improvement from R-GCN RL([†]) to either R-GCN RL-FT([†]) or BS-RL([†]). Highlighted values indicate the best results compared to the R-GCN RL baseline (**with** or **without** congestion handling).

7.3.2 Comparison Against Baselines

As expected, SA produces very compact floorplans, but their very high RUDY estimate suggests low practical usability, and the need for a lot of manual refinement. RL, and in particular R-GCN RL-FT, generate very compact but less congested floorplans. The improvement recorded by BS-RL aligns with the one given by R-GCN RL-FT, sometimes even surpassing it. Moreover, the runtime of BS-RL is generally similar to R-GCN RL-FT, often lower, but it has the advantage of running on CPU, while R-GCN RL-FT needs GPUs to compute in a reasonable time.

The only outlier is observed for the Bias-2 instance, but as suggested by the runtimes of smaller problem instances, it doesn't scale monotonically with respect to problem size. Supposedly, runtime doesn't increase significantly on problems of even higher complexity, but new data would be needed for assessment. Some negative percentage improvements are recorded, in particular for the RUDY in the R-GCN RL-FT and R-GCN RL-FT[†] columns. Since fine-tuning produces more compact outputs, it's likely that these floorplans don't leave much space for routing. Also the BS-RL[†] result of the RS Latch instance records a negative improvement, but the absolute value is close to the one accomplished by R-GCN RL[†], meaning that optimality is probably achieved by both procedures.

One of the most surprising results of this experimental evaluation is that the outputs of RL[†], compared to RL, even though are better in terms of RUDY, are far worse in terms of DS, HPWL and reward. With BS it is possible to obtain floorplans that are less congested, and as compact as the ones obtained without requiring the congestion constraint.

To conclude, the BS-RL([†]) algorithm improves the results obtained with R-GCN RL([†]), having average % improvement across all test cases: DS: 4.97%, HPWL: 16.05%, RUDY: 52.09%, Reward: 28.39% (DS: 36.05%, HPWL: 68.41%, RUDY: 34.69%, Reward: 84.86%). Moreover, the measured metrics align closely with the ones achieved by a state-of-the-art procedure like R-GCN RL-FT([†]), and often show higher stability, having lower IQR.

7.4 Discussion

This chapter presented a novel approach to enhance the strengths of RL by integrating BS to improve the robustness and efficiency of an already optimal automated floorplan-

Table 7.1: Comparative analysis of SA, R-GCN RL, R-GCN RL-FT, BS-RL, and their variations with congestion handling ([†]).

Circuit	# Struct	Metric	SA	R-GCN RL	R-GCN RL-FT	BS-RL	R-GCN RL [†]	R-GCN RL-FT [†]	BS-RL [†]
OTA-1	5	Runtime (s)	0.84 ± 0.07	0.12 ± 0.01	147.73 ± 3.19	17.46 ± 1.11	0.98 ± 0.33	149.37 ± 15.86	18.38 ± 1.47
		DS (%)	48.69 ± 4.04	46.79 ± 1.70	43.31 ± 11.46 (3.48%)	47.83 ± 6.35 (1.04%)	87.12 ± 6.0	63.79 ± 18.2 (23.33%)	48.73 ± 5.14 (38.39%)
		HPWL (μm)	167.94 ± 25.82	224.74 ± 10.84	163.91 ± 43.15 (27.07%)	200.23 ± 39.81 (10.91%)	560.1 ± 104.47	284.46 ± 114.95 (49.21%)	203.62 ± 24.02 (63.65%)
		RUDY Reward	3.1 ± 0.26 -2.13 ± 1.23	0.19 ± 0.11 -3.88 ± 0.31	0.16 ± 0.05 (15.79%) -1.72 ± 1.78 (55.67%)	0.08 ± 0.02 (57.89%) -3.12 ± 1.21 (19.59%)	0.12 ± 0.01 -23.44 ± 2.72	0.16 ± 0.08 (33.33%) -7.02 ± 5.27 (70.05%)	0.06 ± 0.01 (50%) -3.3 ± 0.88 (85.92%)
OTA-2	8	Runtime (s)	0.97 ± 0.07	0.2 ± 0.02	266.92 ± 7.32	119.17 ± 4.87	3.01 ± 0.14	274.77 ± 28.24	251.86 ± 252.6
		DS (%)	54.05 ± 7.45	48.31 ± 13.76	40.03 ± 4.47 (8.28%)	46.33 ± 6.58 (1.98%)	91.64 ± 1.87	66.19 ± 28.86 (25.45%)	47.67 ± 10.26 (43.97%)
		HPWL (μm)	219.54 ± 40.71	244.04 ± 54.35	195.86 ± 38.07 (19.74%)	211.89 ± 37.02 (13.17%)	1001.36 ± 140.85	455.81 ± 310.63 (54.48%)	215.86 ± 77.53 (78.44%)
		RUDY Reward	2.06 ± 0.56 -3.02 ± 1.45	0.22 ± 0.09 -4.12 ± 2.32	0.24 ± 0.06 (-9.09%) -2.19 ± 1.14 (46.84%)	0.18 ± 0.05 (18.18%) -2.95 ± 1.41 (28.4%)	0.11 ± 0.01 -39.86 ± 6.17	0.14 ± 0.08 (-27.27%) -12.43 ± 13.29 (68.82%)	0.1 ± 0.03 (9.09%) -3.17 ± 3.02 (92.05%)
Bias-1	9	Runtime (s)	1.24 ± 0.07	0.23 ± 0.02	282.63 ± 30.86	76.69 ± 2.94	4.76 ± 0.38	286.93 ± 31.48	131.95 ± 160.59
		DS (%)	63.95 ± 6.32	51.26 ± 6.14	55.54 ± 10.65 (-4.28%)	48.26 ± 9.49 (3%)	91.73 ± 0.52	76.85 ± 15.02 (14.88%)	51.16 ± 7.52 (40.57%)
		HPWL (μm)	286.92 ± 53.53	337.36 ± 65.07	266.79 ± 79.72 (20.92%)	270.66 ± 43.2 (19.77%)	1016.39 ± 142.79	446.54 ± 164.21 (56.07%)	284.25 ± 62.75 (72.03%)
		RUDY Reward	5.01 ± 0.54 -6.48 ± 2.28	0.37 ± 0.24 -8.2 ± 2.51	0.37 ± 0.1 (0%) -5.84 ± 3.42 (28.78%)	0.16 ± 0.04 (56.76%) -5.72 ± 1.52 (30.24%)	0.15 ± 0.04 -42.4 ± 4.99	0.3 ± 0.14 (-100%) -14.52 ± 8.68 (65.75%)	0.1 ± 0 (33.33%) -6.25 ± 2.03 (85.26%)
RS Latch	7	Runtime (s)	0.88 ± 0.05	0.17 ± 0.02	200.84 ± 23.22	216.59 ± 7.03	2.74 ± 0.37	201.34 ± 23.42	357.69 ± 43.79
		DS (%)	62.79 ± 6.21	52.22 ± 8.12	49.81 ± 12.83 (2.41%)	45.61 ± 8.58 (6.61%)	90.67 ± 2.2	76.13 ± 14.19 (14.54%)	50.9 ± 10.58 (39.77%)
		HPWL (μm)	124.2 ± 22.31	128.44 ± 15.78	123.83 ± 25.2 (3.59%)	121.21 ± 10.25 (5.63%)	336.37 ± 68.91	204.05 ± 69.18 (39.34%)	130.57 ± 16.37 (61.18%)
		RUDY Reward	3.76 ± 1.79 -5.1 ± 2.25	0.68 ± 0.51 -4.96 ± 1.54	0.94 ± 0.45 (-38.24%) -4.63 ± 2.2 (6.65%)	0.26 ± 0.12 (61.76%) -4.19 ± 0.92 (15.32%)	0.12 ± 0.02 -28.25 ± 6.65	0.6 ± 0.36 (-400%) -12.54 ± 7.52 (55.61%)	0.16 ± 0.07 (33.33%) -4.96 ± 1.57 (82.44%)
Driver	17	Runtime (s)	1.61 ± 0.06	0.45 ± 0.04	696.88 ± 71.18	254.64 ± 10.68	15.04 ± 1.43	848.91 ± 68.34	395.39 ± 279.74
		DS (%)	71.75 ± 9.59	72.62 ± 8.66	54.34 ± 11.55 (18.28%)	57.56 ± 8.63 (15.06%)	91.8 ± 0.77	61.62 ± 11.84 (30.18%)	57.78 ± 7.01 (34.02%)
		HPWL (μm)	1931.84 ± 407.06	2502.8 ± 374.68	1647.64 ± 290.09 (34.17%)	1767.78 ± 329.69 (29.37%)	5849.24 ± 878.71	1878.77 ± 369.52 (67.88%)	1791.86 ± 279.63 (69.37%)
		RUDY Reward	4.66 ± 0.88 -8.35 ± 3.22	0.25 ± 0.13 -13.1 ± 2.82	0.32 ± 0.24 (-28%) -6.26 ± 2.18 (52.21%)	0.08 ± 0.01 (68%) -7.16 ± 2.48 (45.34%)	0.27 ± 0.09 -42.18 ± 6.19	0.32 ± 0.25 (-18.52%) -8.16 ± 2.74 (80.65%)	0.07 ± 0.01 (74.07%) -7.26 ± 2.22 (82.79%)
Bias-2	19	Runtime (s)	1.91 ± 0.1	0.52 ± 0.05	983.98 ± 94.55	1293.42 ± 82.25	13.53 ± 0.98	1182.4 ± 62.7	1490.64 ± 321.83
		DS (%)	66.05 ± 5.92	75.73 ± 5.39	62.1 ± 7.89 (13.63%)	71.55 ± 6.11 (4.18%)	91.12 ± 0.73	63.75 ± 10.09 (27.37%)	71.56 ± 6.14 (19.56%)
		HPWL (μm)	2463.06 ± 571.35	3576.77 ± 520.26	2694.18 ± 429.98 (24.68%)	2953.08 ± 423.86 (17.44%)	8532.99 ± 1306.63	2815.12 ± 550.38 (67.01%)	2921.03 ± 468.06 (65.77%)
		RUDY Reward	2.33 ± 0.39 -2.99 ± 2.01	0.12 ± 0.05 -8.12 ± 1.88	0.15 ± 0.08 (-25%) -4.07 ± 1.56 (49.88%)	0.06 ± 0.01 (50%) -5.58 ± 1.37 (31.28%)	0.24 ± 0.11 -28.73 ± 3.43	0.11 ± 0.04 (54.17%) -4.64 ± 1.64 (83.85%)	0.06 ± 0.01 (75%) -5.55 ± 1.37 (80.68%)

ning methodology. By leveraging BS, the exploration capabilities of the RL agent are significantly enhanced in a simple yet effective manner, without altering the underlying structure of the original solution. This incorporation allows the RL agent to achieve performance comparable to that of a fine-tuned version while preserving the key features of the original solution, including its generalization ability, handling of diverse circuit features, avoidance of module overlaps, and adherence to various design constraints. Additionally, the proposed approach increases the flexibility of the underlying methodology, enabling optimization towards different objectives. A congestion management procedure further ensures that the generated layouts are more routing-friendly, improving the practicality of the algorithm for real-world applications. Importantly, these advancements do not require updating or fine-tuning the RL policy, thereby eliminating the dependence on computationally expensive resources such as GPUs, and they maintain runtime efficiency, making the solution more accessible and scalable.

Looking ahead, moving closer to real industrial applications will require additional advancements. One promising direction involves incorporating post-layout electrical simulation directly into the placement loop to enable early detection of parasitic effects and performance deviations. Such integration would guide the RL policy toward generating floorplans that require minimal manual refinement, ultimately paving the way for fully automated layouts. At this stage, it would be possible to achieve layouts that are LVS and DRC clean, thereby fulfilling the vision of complete automation in analog layout design.

Chapter 8

Advancing Routing-Awareness in Analog ICs Floorplanning

In this chapter, we address a prevalent concern among physical design engineers regarding the need for routing-ready floorplanning solutions. In previous works, the routability of the final floorplan solution was often overlooked in favor of area compactness. To this extent, while still preserving the structure and methodology introduced in Chapter 6, we develop an enhanced floorplanning engine that specifically conditions the floorplan generation towards more routable outcomes.

By increasing the grid resolution and incorporating pin information both at graph and RL agent level, we can leverage an higher awareness of the whole system for producing more routing-friendly floorplans. The methodology in fact, thanks also to the integration of dynamic routing resource estimation procedure, can accommodate enough room for routing wires while guaranteeing very compacted and HPWL optimized floorplan results.

Compared to the earlier methodologies presented in Section 6.3, the proposed approach achieves substantial performance enhancements, reducing dead space and HPWL by 4.1% and 26.3%, respectively, when compared to a 0-shot baseline approach. Moreover, when analyzing the place and route effectiveness in a simulated environment, our solution demonstrated a 73.4% increase in routing success compared to previous state-of-the-art techniques.

In Section 8.2, the new routing-aware floorplanning and routing engines are thoroughly described in all their components. Then, Section 8.3 presents the raw placement and routing related performances achieved with this new setup. Finally, concluding remarks are drawn in Section 8.4. The content of this chapter has been published in [9].

8.1 Introduction

Throughout this thesis, we have emphasized that analog IC layout should always be approached as a unique process rather than as two independent, non-communicating steps. Floorplanning and routing are in fact inherently interdependent, with the optimization objectives of one stage directly affecting the feasibility and quality of the other. For instance, a compact but unroutable floorplan makes the automation efforts ineffective, highlighting the importance of considering routing metrics early in the layout stage.

Fully automated layout engines, such as MAGICAL [23] and ALIGN [41], have proved

to be effective in delivering analog circuits layouts by employing analytical or metaheuristic based floorplanning engines, to then connect all the components together typically through the A* algorithm. Some efforts have been made to integrate learning-based solutions into these pipelines as well, targeting both the floorplanning [101, 109] and routing phases [25, 193]. However, these promising approaches have shown limited industrial adoption, highlighting the need for further exploration in this direction.

On the other hand, the combination of R-GCN and RL presented in Chapter 6 already proved to be optimal, ensuring high quality outcomes that can meet industrial expectations when inserted into a real-world working pipeline. For this reason, to increase the routing-awareness of such methodology, we opt to extend its capability with the following enhancements.

First, we develop a higher resolution grid to enable more compact and precise device placement, ensuring finer granularity in the layout space. Additionally, we incorporate supplementary information derived from terminals and pins into both the graph representation and RL agent. This enhancement leads to more accurate HPWL calculations and significantly improves the agent’s understanding of the layout problem.

Moreover, a dynamic routing resource estimation based on such new pin information, inspired by [134], is used to reserve enough space between floorplan elements, nicely accommodating the routing demands in an highly flexible manner.

Furthermore, we introduce a revised reward scheme that actively prefers the generation of more routing-friendly floorplans by prioritizing HPWL optimization without sacrificing area efficiency, ensuring improved final outcomes.

We devise a custom A* algorithm, inspired by works as [22, 188], to validate the quality of the generated floorplans and their corresponding routing solutions within a simulated environment. The proposed framework outperforms past approaches both in terms of raw placement and routing-related metrics, delivering results which are consistently more routable.

8.2 Methodology

8.2.1 Pin-Enhanced Circuit Graph

As discussed in Section 6.2.2, a circuit netlist can be naturally represented as a graph. Typically, a node $v_i \in V$ corresponds to a sub-block or an individual device, while edges $e_i \in E$ represent the connectivity between nodes. It has been shown that, by leveraging the capabilities of R-GCNs, it is possible to differentiate among the various types of edges that connect nodes. Specifically, edges can represent either electrical connections through a net or topological constraints that hold between components.

Likewise, R-GCNs can also differentiate nodes based on their roles within the circuit. To this extent, our novel graph representation improves upon recognizing only functional blocks and devices as nodes by also including pins and nets. While the representation of pins as nodes has been explored Li et al. in [101], the explicit inclusion of nets as nodes in the graph allows us to can encode detailed and precise connectivity related attributes directly as node features, rather than associating this information with the edges.

The feature vector of a device node, $x_d \in X$, includes the block area, width, height, the number of pins, and a 28-dimensional one-hot encoding representing the sub-block’s

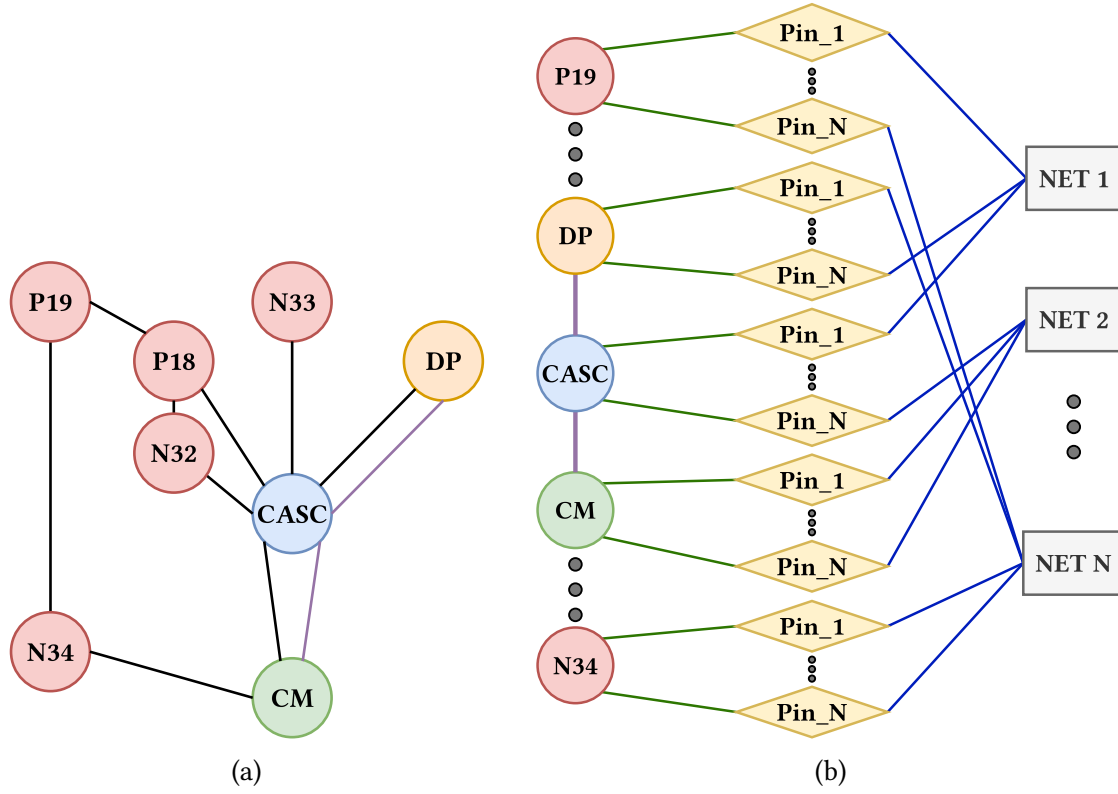


Figure 8.1: A comparison between the previous circuit graph configuration (a) and the enhanced near-tripartite one (b). In the enhanced graph, violet edges represent the topological constraints applied between recognized structures, green ones denote the *device has pin* relationship, and blue edges connect pins and nets through the *pin belongs to net* relationship.

functional structure, as described in Section 6.2.2. Differently from the previous node feature vector, the specific device parameters were omitted in this case to enhance generalization, ensuring that any type of device, including transistors, resistors, capacitors, or other components, can be taken as input.

Similarly, the feature vector of a pin node, $x_p \in X$, is characterized by a series of numerical encodings of pin’s direction, the type of net it is connected to (signal, ground, or power), and a one-hot encoding of both the bottom and top metal layers that the pin can accept, which varies according to the technology specifications.

Finally, the feature vector of a net node, $x_n \in X$, is constructed by concatenating the number of connected devices and pins.

Connections between nodes are established based on the relationships among them. By construction, device nodes and pin nodes are connected through a *device has pin* relationship, while pin nodes are connected to net nodes via the *pin belongs to net* relation. This approach results in a near-tripartite graph structure. We refer to it as “near-tripartite” because, in addition to the device-pin-net relationships, we need to consider also the constraints between devices that are also represented as edges. An example of the proposed circuit graph representation is shown in Figure 8.1b.

The updated R-GCN architecture we devised to handle the new circuit graph formulation consists of two R-GCN layers followed by a node mean aggregation block to generate

Table 8.1: Updated R-GCN hyperparameters.

R-GCN Model	
R-GCN units	64
Learning rate	$4.7e-3$
Dropout	0.1
Optimizer	ADAM
Weight Decay	$5.6e-5$
β_1, β_2	0.9, 0.999
FC Neural Network	
Input normalization	tanh
Activation	ReLU
Hidden layer neurons	[64,32,1]
Loss	MSE

the graph embedding, which is subsequently passed through two fully connected layers to predict the reward value. The training dataset now includes 91200 floorplans generated following the same approach as detailed in Section 6.2.2. The updated R-GCN architecture and training hyperparameters are presented in Table 8.1.

8.2.2 A More Routing-Aware RL Agent

The core methodology relies on the framework described in Section 6.2.3, retaining the foundational concepts of actions, state, and reward. However, each of these components has been refined and enhanced, as detailed in the following subsections.

Higher-Resolution Grid

Analog ICs layout is characterized by stringent requirements, many of which come from electrical needs, as highlighted in Section 2.1. Among these, achieving a compact placement of circuit components remains one of the most critical objectives. Additionally, the geometric constraints imposed to minimize parasitics and mismatch necessitate a high degree of precision during the floorplanning process. To address this aspect, the granularity of the placement grid is increased from 32×32 to a 256×256 . While this change expands the action space, it enables a much finer placement resolution, ensuring a higher level of precision in the final layout.

Policy Architecture Improvement

In order to sustain the larger action and state space, a new architecture for the policy that governs the action selection process of the RL agent is needed. Taking inspiration from the work of Geng et al. in *EfficientPlace* [49], we decide to move from a simple CNN model architecture to a more sophisticated U-Net [147].

Originally designed for medical images segmentation, the U-Net is perfectly suited for efficiently processing visual information coming from the visual masks that are part of the agent's state, f_g, f_w, f_{ds}, f_p . In fact, thanks to its architecture design, which is

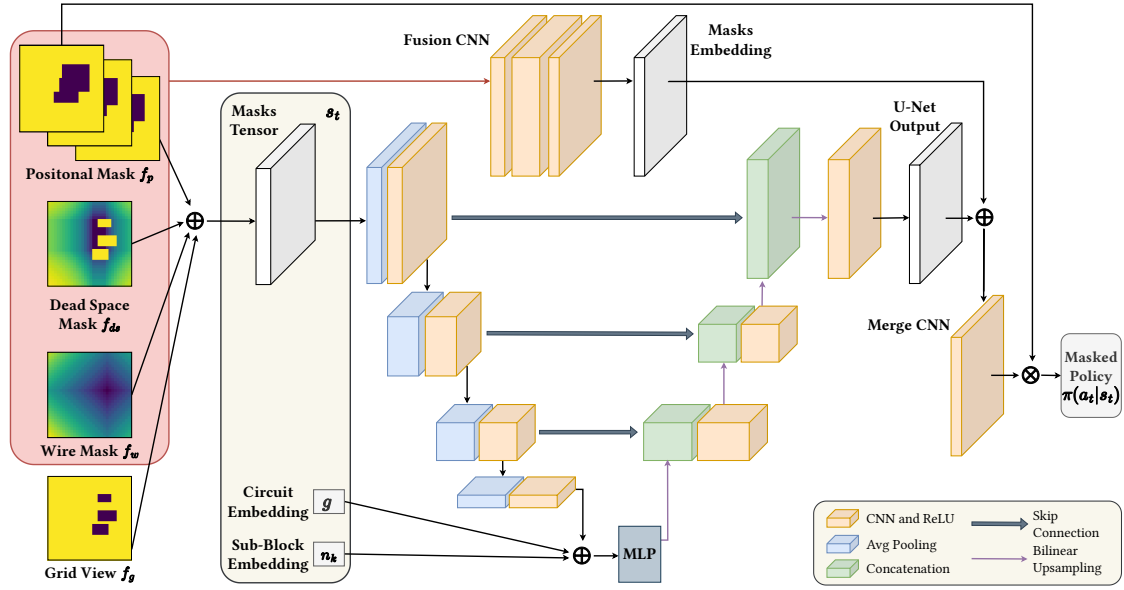


Figure 8.2: Overview of the U-Net based RL policy network.

detailed hereafter, it is possible to process high-dimensional input data without incurring in a significant computational overhead. As a result, despite the increased grid size and the large action space of $|\mathcal{A}| = 256 \times 256 \times 3 = 196608$, employing the U-Net enables effective training and inference, ensuring that the RL agent remains scalable.

The U-Net architecture adopts a U-shaped configuration, which can be conceptually divided into three main components: the encoding (or contracting) path, the bottleneck, and the decoding (or expansive) path.

The *contracting path*, located on the left side of the U-Net, serves as a feature extractor for the input image-shaped data while progressively reducing its spatial dimensions. This section typically consists of a sequence of convolutional layers, followed by ReLU activation functions and max-pooling operations.

The *bottleneck*, which forms the central portion of the U-Net, represents the most compact and low-dimensional encoding of the input data. In our setup, this bottleneck incorporates an MLP to efficiently combine not only the information extracted from the contracting path, but also the R-GCN generated circuit g , and single block n_k embeddings. Doing so, both global circuit and local block-specific details are integrated into the distilled encoding.

Eventually, the *expansive path*, i.e. the right side of the U-Net, reconstructs the spatial dimensions of the input data through a series of upsampling and convolutional layers. *Skip connections* are used to directly connect layers from the encoding path to their corresponding layers in the decoding section. These help to retain fine-grained spatial details lost during downsampling, enabling the network to reconstruct the spatial structure of the input with high precision.

In order to better guide the RL agent's decision-making process, as the final step of our policy, the state masks are fed to a CNN to eventually retrieve a learned feature vector of such combination. The output of this CNN is then integrated with the probability distribution produced by the U-Net architecture, using an additional CNN, to produce the final action probabilities for the RL agent. The value network instead follows the

Table 8.2: RL policy model hyperparameters.

Model Part	Layer ID	Kernel Size	N. Channels	Stride	Padding	Output Size
U-Net Encoder	E_CNN_1	3×3	8	1	1	(8, 128, 128)
	E_CNN_2	3×3	16	1	1	(16, 64, 64)
	E_CNN_3	3×3	32	1	1	(32, 16, 16)
	E_CNN_4	3×3	32	1	1	(32, 4, 4)
	Max_Pool_2	2×2	-	-	-	-
	Max_Pool_4	4×4	-	-	-	-
Bottleneck	FC_1	-	-	-	-	(640, 512)
	FC_2	-	-	-	-	(512, 512)
U-Net Decoder	D_CNN_1	3×3	16	1	1	(16, 16, 16)
	D_CNN_2	3×3	8	1	1	(8, 64, 64)
	D_CNN_3	1×1	3	1	0	(3, 256, 256)
	Bilinear_Up_2	-	-	-	-	(in $\times$ 2, in $\times$ 2)
	Bilinear_Up_4	-	-	-	-	(in $\times$ 4, in $\times$ 4)
Fusion CNN	F_CNN_1	1×1	8	1	0	(8, 256, 256)
	F_CNN_2	1×1	8	1	0	(8, 256, 256)
	F_CNN_3	1×1	3	1	0	(3, 256, 256)
Merge CNN	M_CNN_1	1×1	3	1	0	(3, 256, 256)

same rationale of the contracting path of the policy network, ultimately outputting the learned value function values through an MLP.

The overall architecture used for the policy network is depicted in Figure 8.2, while the corresponding hyperparameters are listed in Table 8.2.

Dynamic Routing Resource Allocation

Previous works, such as [1, 31], considered congestion as a fundamental proxy for routability during placement. As mentioned in Section 7.2.2, RUDY is a common metric used to estimate the routing demand for a circuit. Motivated by the goal of developing a highly optimized floorplanning engine capable of delivering routing-ready results, we considered integrating this metric into the optimization process. Specifically, a heatmap similar to f_w was generated to guide the RL agent research areas toward less congested regions of the chip canvas. However, due to the instability shown during training and the associated demanding computational efforts related to such procedure, we decided to not pursue this direction.

HPWL alone has been demonstrated to positively correlate with RUDY [160], and it offers a much more stable signal for ensuring efficient and effective training of the RL model. Nevertheless, to guarantee adequate spacing between circuit components and avoid excessive routing congestion, additional measures are required. In this regard, the dynamic routing resource (DRR) allocation mechanism, as first proposed by [134], provides a simple yet functional balance between simplicity and effectiveness.

The routing space, $\lambda_{i,[h,v]}$ required on either the horizontal or vertical sides of a block i , is defined as follows:

$$\lambda_{i,[h,v]} = \zeta + |p_{i,[h,v]}| \cdot \left(\max_{j \in Nets(i)} (\mathcal{W}_{j,spacing}) + \max_{j \in Nets(i)} (\mathcal{W}_{j,width}) \right), \quad (8.1)$$

where ζ is equal to the minimum chip canvas placement grid step, $|p_{i,[h,v]}|$ is the number of vertical or horizontal pins of block i , $\max_{j \in Nets(i)}(\mathcal{W}_{j,spacing})$ is the maximum parallel run spacing among the nets connected to block i , $Nets(i)$, and $\max_{j \in Nets(i)}(\mathcal{W}_{j,width})$ is the maximum wire width for each connection.

Since the output direction of the pins during placement is unknown, the routing space defined in Equation 8.1 must be padded symmetrically. More precisely, horizontal pins require padding on both the left and right sides, while vertical pins on the top and bottom ones. This approach inevitably results in an overestimation of the actual space needed to connect all components. However, as will be explained in Section 8.3, this added room ensures very low routing failure rates, while still achieving well-optimized floorplans in terms of area utilization and HPWL.

How to reduce such overestimation and the adoption of post-processing techniques to eliminate any unnecessary space after the routing stage will be surely object of future research.

Routing-Driven Reward Design

The reward function is a critical component of any RL system, as its definition significantly influences the quality of the model's training and the relevance of the resulting solutions. To better prioritize the HPWL optimization, being the key metric directly targeting more routing-efficient floorplans, a revision of the reward weights is necessary. With the inclusion of precise pin information for each device, the HPWL calculation now accounts for the exact location of each network pin. This enhancement provides higher fidelity estimates of the routing wirelength, in contrast to the previous approach that relied on the blocks center as the reference point for HPWL calculations. While sufficient for digital layout scenarios, this abstraction leads in fact to poor results when applied to analog circuits.

In multi-objective optimization, lexicographic optimization is a common approach where priorities are assigned to objectives. Objectives with higher priority are optimized first, and successive objectives are improved only if the higher-priority ones are already optimized. To steer the floorplanning process towards more HPWL optimized results without sacrificing area, a linear lexicographical weighting system is introduced. In theory, obtaining the lexicographic optimum would require solving all the m individual minimization problems in series, where m represents the number of objectives. However, a computationally efficient alternative is to solve a single minimization problem using a weighted linear combination of the objectives, where the weights satisfy $\alpha_1 \gg \alpha_2 \gg \dots \gg \alpha_m$. While this method approximates the lexicographic optimum, it is sufficient for discrete problems if the weights are distinct enough and well adapts to the single-objective nature of the RL reward system.

Following this idea, the reward weights α , β , and γ in Equation 6.5 are set to 10, 100, and 5, respectively, to prioritize HPWL optimization while maintaining area efficiency. Additionally, a penalty of -1000 is assigned to the RL agent for violating predefined constraints, ensuring that such violations are heavily discouraged during the optimization process.

Updated Dataset and RL Training Specifics

The procedure employed for training the RL agent is the same HCL described in Section 6.2.3. The training dataset was expanded to include 10 circuits that vary in nature, topology, and functionality, all based on the Infineon 130nm Commercial Technology node. These encompass various types of OTAs, biasing circuits, clock resynchronization circuits, sense amplifiers with RS latches, delay elements, band-gap voltage references with low-power operational amplifiers, and level shifters. The number of functional blocks within these circuits ranges from 3 to 10, with individual devices varying between 9 and 26.

8.2.3 A* Routing Engine

To finally close the loop between floorplanning and routing and validate the results obtained with the new routing-aware RL agent in terms of routability, following the promising methodologies employed by both state-of-the-art MAGICAL router [22] and SAGERoute [188, 189] works, we propose an A*-powered routing engine. This framework is capable to comply by common industrial design rules to avoid the arise of DRC errors by construction, while also supporting multiple metal layers routing. The prototype, despite being at early stages of its development, allows us to effectively assess the routing awareness of the proposed floorplans.

Routing Grid Graph

The routing grid is modeled as a 3-D graph, $G_r(V_r, E_r)$, representing the entire routing space. V_r denotes the set of possible interconnection points, i.e. vertices, and E_r the set of edges. The edges $(u, v) \in E_r$ can either be horizontal or vertical, depending on the connection type. In fact, horizontal edges correspond to routing track segments within the same metal layer, while vertical ones represent connections between different metal layers, realized physically as vias.

In our setup, the grid step used to interleave two routing vertices is determined by the minimum valid spacing between two physical components specified by the circuit technology. To efficiently represent G_r , since the number of vertices is typically large, an adjacency list is implemented using a hashmap structure, enabling scalable storage and fast lookup of the graph's components. To further limit the increase in size of G_r we limit the possible metal layers to two options, one for vertical and the other for horizontal tracks.

DRC Rules and Constraints

Analog ICs routing is characterized by a multitude of rules that must be satisfied to meet industrial requirements and practical usability of the final layout. For this reason, we guarantee the compliance to the following fundamental DRC rules:

- *Minimum wire length*: Specifies the minimum length for a wire segment in G_r .
- *Minimum wire area*: Specifies the minimum area for a wire segment in G_r .

- *End of line spacing*: Defines the minimum required spacing, *eolSpace*, from any other wire or via at the end of a wire segment within the *eolWithin* distance beyond the end of the line. This rule applies only if a parallel wire segment exists within the parallel edge region on the side of the wire segment of interest, defined by the *parWidth* and *parSpace* values.
- *Parallel run spacing*: Specifies the minimum parallel spacing from any other wire or via based on the parallel run length and the maximum width of the two wires. In our scenario, we set it to a minimum fixed valid value.

Obstacle avoidance is also considered in order to ensure that wires do not overlap with pins that are not part of the net of interest or intersect with devices. Moreover, given that analog circuit routing typically involves highly regular shapes to form interconnections between devices, a low-bending rule is introduced to discourage too tortuous paths, which would eventually lead to an increased number of vias and the possibility to incur in DRC violations.

The wire width is set to the minimum valid value to ensure compliance with DRC rules. However, the system is designed to accommodate varying wire widths for different metal layers, allowing flexibility based on specific routing requirements or technology constraints.

The Routing Scheme

The routing process is based on a negotiated rip-up and reroute scheme [124]. Whenever a net violates one of the aforementioned rules or constraints, a history cost associated to all the vertices that have been used to construct the failing wires is accredited before they are removed. Doing so, the A* engine can avoid to visit in successive routing iterations the same nodes that led to some source of errors in the past, incentivizing a different exploration of the grid space.

The net routing order is defined according to a heuristic ranking the nets based on their criticality and size. First of all, nets with more routing failures are prioritized. Then, among nets with the same number of routing failures, nets with higher HPWL are prioritized. If both routing failures and HPWL are identical, the net with more pins is favored. Doing so, not only the final quality of the layout is improved, but also the number of iterations to generate a valid routing.

The output direction of the pins to be connected through a net has to be defined at the very beginning of the routing stage. In fact, pins are typically positioned within a device layout and they need to be projected to one side of the block, either horizontally or vertically, depending on pin's characteristics. Since HPWL is an optimal estimator of the true circuit wirelength, the approach we follow is to select the block side for each pin that leads to the smallest increase in the HPWL value of the net to be routed. This is achieved by evaluating all the possible combinations of pin-to-block side assignments for the net of interest. While this naive approach increases slightly the computational effort and congestion, it keeps a strong correlation with the smallest estimated wirelength, which in the end is the key goal for an high quality routing solution. In such cases, the earlier padding overestimation around the blocks proves to be highly beneficial.

Once all the endpoints' positions are defined, we decompose each net into a two-pin one by using a MST decomposition.

The A* search follows a bidirectional scheme by alternating a forward and backward search, starting from the source and target pin respectively, in order to improve the speed efficiency of navigating into such a large grid space. Design rules and constraints are integrated into the node cost function $g(n)$ in a shallow manner. If n is a neighboring node of the current node k , the cost function can be expressed as:

$$D(n, k) = |x_n - x_k| + |y_n - y_k| + c_{\text{VIA}} \cdot |z_n - z_k| \quad (8.2)$$

$$g(n) = g(k) + D(n, k) + c_{\text{DRC}} \cdot \mathbb{1}(\text{violation}) + H(n), \quad (8.3)$$

where $D(n, k)$ is a Manhattan distance metric that accounts also for a via cost, c_{VIA} , to discourage excessive layer changes and jagged paths. Instead, $c_{\text{DRC}} \cdot \mathbb{1}(\text{violation})$ represents a penalty for DRC violations when visiting the neighboring node, while $H(n)$ is the accumulated history cost of n up to the current reroute iteration. Overall, this formulation ensures that the cost function penalizes undesirable routing behaviors while also promoting efficient and compliant routing paths. On the other hand, obstacles are directly removed from the routing grid, simplifying the routing process and ensuring that invalid paths are avoided by construction.

The heuristic $h(n)$ used to estimate the cost from the neighboring node to the target goal t is computed as $D(t, n)$. In order to gain some speedup in runtime we decided to introduce a *dynamic weighting* mechanism for the cost function as follows:

$$f(n) = \begin{cases} g(n) + h(n) & \text{if } g(n) < h(n), \\ \frac{g(n) + (2\kappa - 1) \cdot h(n)}{\kappa} & \text{otherwise.} \end{cases} \quad (8.4)$$

This idea is inspired by the work of Chen et al. in [26], which demonstrates how increasing the weight of the heuristic depending on κ as the routing path progresses can be used to exploit the proximity to the goal. In our implementation, we set $\kappa = 3$. Doing so, the algorithm prioritizes reaching the target rather than exploring the search space, which on the contrary is preferred during the early stages of the search.

Finally, a *tie-breaking* mechanism is introduced to prevent the A* router from exploring all paths of the same length instead of focusing on a single optimal path. This is achieved by multiplying $h(n)$ by $1 + q$, with $q = 0.0001$, which biases the algorithm toward expanding vertices closer to the goal and further reduces unnecessary exploration of the search grid. This approach, combined with the dynamic weighting from before, ensures reasonably fast execution runtimes for the whole pipeline when used both in testing and production environments.

8.3 Experimental Results

The agent is implemented in Python 3.11, leveraging the same stable-baselines3 and DGL libraries as in previous chapters. Due to the increased size of the training dataset, the training process requires slightly more time, taking approximately 3 days and 20 hours on a single Nvidia A30 GPU.

The A* routing engine, on the other hand, is implemented in C++, in favor of guaranteeing short runtimes and high computational efficiency. The ankerl [99] library is used for hashmap operations, the Boost [155] library for implementing an efficient R-tree spatial index, and the nlohmann [111] library for processing JSON input files.

Table 8.3: Comparison of the routing-aware approach versus R-GCN RL.

Algorithm	Finetune Shots	Runtime (s)	Dead Space (%)	HPWL (μm)	Reward
Routing-Aware	0	0.52 ± 0.29	57.44 ± 2.68	797.29 ± 46.51	-2.61 ± 0.13
Lex. Norm. Routing-Aware	0	0.47 ± 0.12	71.92 ± 1.26	1043.78 ± 60.9	-5.32 ± 0.22
Routing-Aware w/o DRR	0	0.49 ± 0.1	37.28 ± 7.72	673.36 ± 27.37	-1.65 ± 0.23
R-GCN RL	0	0.32 ± 0.12	59.78 ± 11.42	1006.75 ± 167.65	-7.87 ± 19.53
R-GCN RL	1000	788.82 ± 1505.63	55.98 ± 9.06	749.92 ± 74.77	-1.75 ± 0.46

8.3.1 Floorplanning Metrics Assessment

We compare the new routing-aware floorplanning engine against the *R-GCN RL* methodology presented in Chapter 6, which from previous assessments can be considered as state-of-the-art in the field. The comparison is not limited to just the 0-shot and 1000-shot R-GCN RL approaches. In fact, to further investigate whether an alternative reward normalization scheme could enhance training stability and performance, we implemented a new lexicographic normalization system for the reward. This solution aims to achieve a more rigorous standardization of the differently scaled metrics that are optimized simultaneously. Further details on this implementation can be found in Appendix A.

Performances are measured on a test set of 6 different designs. These include both circuits of similar nature to those in the training set, typically with greater complexity, and entirely different circuit types, such as comparators and oscillators. The number of functional blocks in the test circuits varies from 3 to 24, while the total number of individual devices ranges between 5 and 40, ensuring enough variety for validating our approach. No constraints are imposed on any circuit to ensure a fair comparison.

Table 8.3 showcases the IQM and IQR values obtained on the test set circuits for our new floorplanning methodology runtime, resulting floorplan dead space, HPWL and corresponding reward. We also report the results obtained using the routing-aware RL system without DRR, in order to clearly showcase the enhancement provided by the increased grid size and the capability that our new proposed solution can achieve. The best results are presented in bold, while the second and third best rewards are highlighted in italic underlined and underlined only, respectively.

As expected, the increased grid size delivers the best results across all floorplan quality metrics within less than half a second. More specifically, compared to the updated RL agent without DRR, the R-GCN RL 1000-shot approach shows increases in dead space and HPWL of 50.2% and 11.4%, respectively. For the R-GCN RL 0-shot case, these increases are even more evident, reaching 60.4% and 49.5% for the same metrics.

In contrast, the routing-aware system achieves results that nearly match the R-GCN RL 1000-shot produced floorplans, with dead space and HPWL larger by 2.6% and 6.3%, respectively. Nevertheless, the significant runtime improvement, a gain of 150769%, makes this new solution our preferred choice. Moreover, the lower IQR values for every routing-aware solution suggest much more stable performance compared to both previous approaches, making it particularly well-suited for generating fast and accurate floorplan drafts. Finally, the proposed lexicographic normalization scheme is outperformed by the previous normalization scheme. Consequently, this approach was not pursued further in subsequent experiments.

Table 8.4: Routing-related metrics comparison only successfully routed floorplans. The failure rate is instead computed on all available circuits.

Algorithm	Subset	Routing-Aware RL	R-GCN RL 0-shot	R-GCN RL 1000-shot
Failure Rate (%)	All	9.56	79.0	83.0
HPWL (μm)	All	392.70 ± 17.34	483.01 ± 21.66	316.75 ± 30.32
	Common	364.64 ± 13.86	390.02 ± 23.02	359.33 ± 33.70
Wirelength (μm)	All	6496.76 ± 448.58	8732.97 ± 881.36	5637.36 ± 731.86
	Common	5742.17 ± 212.71	7107.96 ± 666.75	6264.70 ± 813.04
Vias	All	144.16 ± 7.98	140.58 ± 6.86	107.75 ± 7.53
	Common	113.48 ± 5.29	117.51 ± 4.50	113.74 ± 7.88
Routing Iterations	All	3.94 ± 4.06	10.01 ± 2.46	3.64 ± 2.78
	Common	3.15 ± 2.96	10.26 ± 1.96	3.57 ± 2.83

8.3.2 Routing Metrics Assessment

To evaluate the routing effectiveness of both the new routing-aware solution and the previous R-GCN RL approach, we applied the A* routing engine to the final floorplans generated by each method. The evaluation involves three key metrics related to routing performance, namely the total routing wirelength, the failure rate of each floorplanning methodology during routing, and the average number of iterations required by the routing algorithm to produce a valid layout. Additionally, we also counted the final number of vias. The maximum number of rip-up and reroute iterations for the A* algorithm is set to 35. Since the routing-aware methodology does not incorporate explicit routing engine runs during the RL training stage, we utilized all available circuits to draw comprehensive conclusions regarding the routing effectiveness of the proposed approach.

Table 8.4 presents the IQM and IQR values for the aforementioned metrics, calculated on the subset of successfully routed circuits. Given that the routing-aware system successfully routes all available circuits, unlike the other methods, the comparison mainly focuses on metrics derived from circuits routed by all three approaches. Nevertheless, for the sake of completeness, the results obtained on the fully routed circuits for each algorithm are also reported. The best results on commonly routed circuits for each metric are highlighted in bold.

As can be seen, the routing-aware approach significantly outperforms prior methods. The inclusion of precise pin information and the prioritization of HPWL optimization have resulted in the generation of more routing-friendly floorplans. While the floorplanning metrics of the routing-aware approach are comparable to those of the finetuned R-GCN RL 1000-shot method, as reported in Section 8.3.1, the routing-aware system achieves a remarkable 73.44% reduction in routing failure rate.

Looking at the results obtained on circuits that were routed collectively by all methods, the routing-aware approach achieves an average reduction in total wirelength and routing iterations of 8.34% and 11.8%, respectively, compared to the R-GCN RL 1000-shot method. When compared to the R-GCN RL 0-shot approach, these improvements become even more pronounced, with decreases of 19.2% in wirelength and 69.3% in routing iterations, respectively.

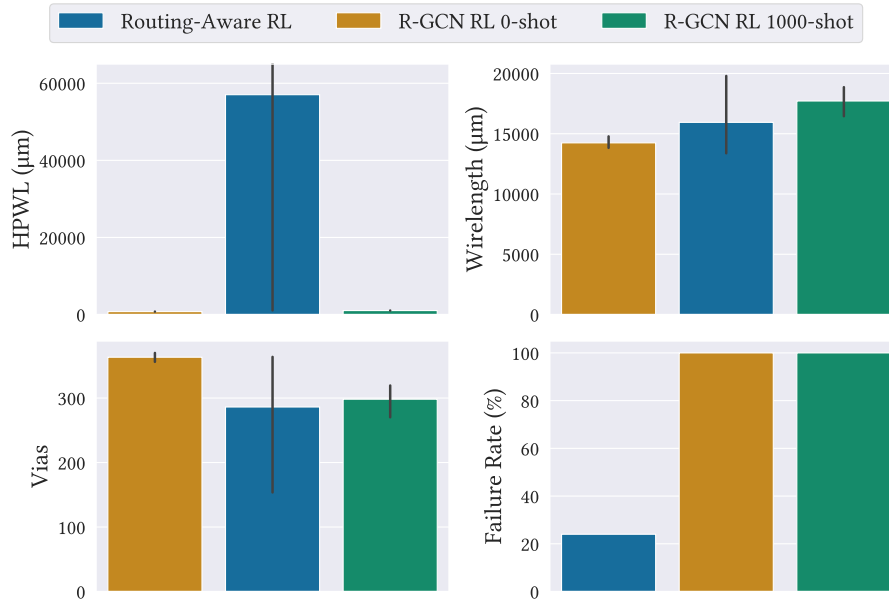


Figure 8.3: Performance comparison on delay line circuit.

A Visual Look At Complete Layouts in The Simulated Environment

As a final assessment, Figure 8.4 presents the complete layout of the 24-block delay line used in the analog core signal timing application generated in the simulated environment. This circuit is the largest and most complex available in the test set. The routing process benefits significantly from the effective spacing introduced between blocks through the DRR allocation procedure, which, when combined with an optimized HPWL during floorplanning, ensures optimal routing paths while adhering to the defined DRC rules.

As shown in Figure 8.3, the previous approaches fail to produce a single fully compatible routing solution. In contrast, the proposed routing-aware solution achieves successful routing in 76% of the cases in just 48.7s on average. This significant improvement demonstrates the effectiveness of the new methodology. Moreover, while the metrics reported in Figure 8.3 are partial due to the inability of the R-GCN RL approaches to complete the routing iterations, the results clearly indicate that the final wirelength is reduced by the routing-aware approach compared to its predecessors. These findings highlight the advantages of incorporating routing awareness into the placement and spacing process, enabling superior performance and compliance with design constraints.

8.4 Discussion

In this chapter, we tackled the analog IC layout problem as a whole. As a first step, we enhanced the R-GCN RL floorplanning system by integrating pins information in the circuit graph structure and by prioritizing the optimization of routing-related metrics, such as HPWL. Moreover, a flexible routing resource allocation strategy was proposed to produce sufficient spacing between components, mitigating congestion and avoiding potential DRC violations. On the placement side, we significantly improved precision by increasing the resolution of the placement grid, enabling a more accurate arrangement

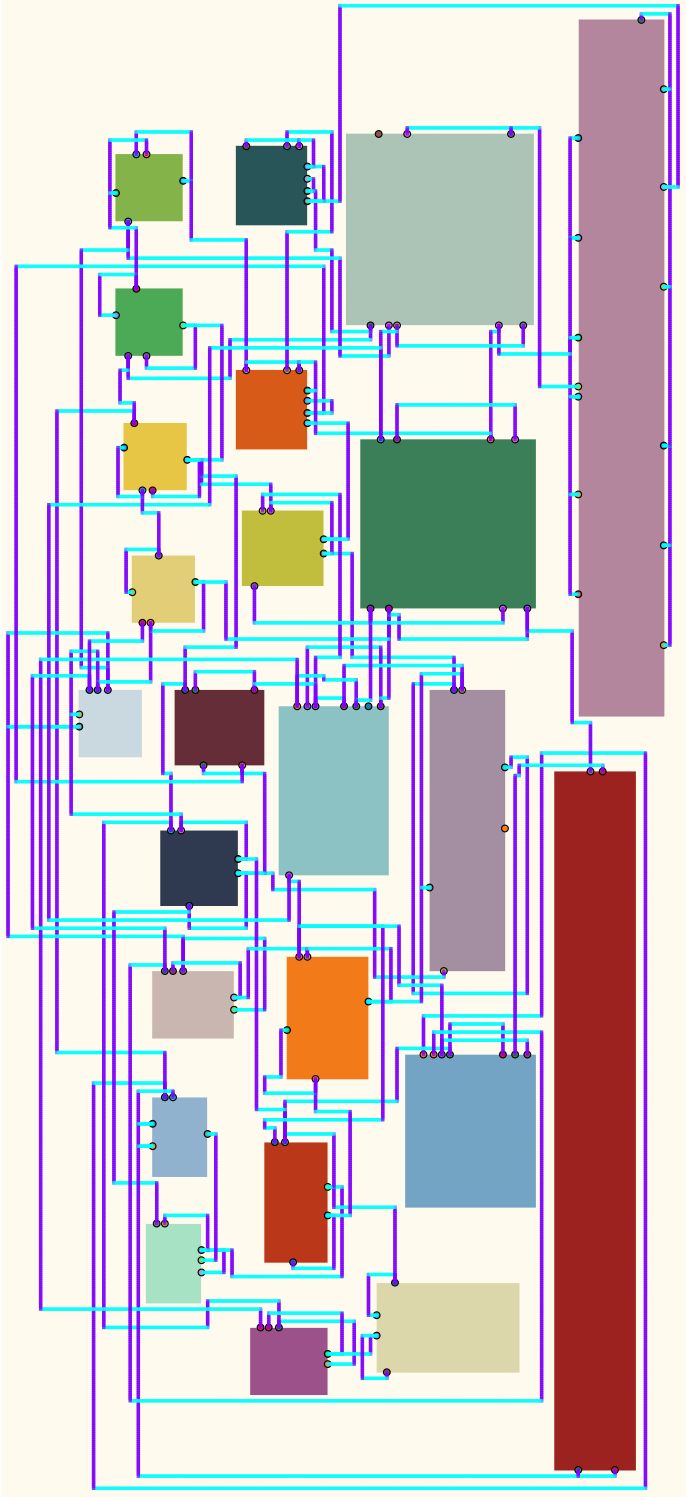


Figure 8.4: Complete layout of the delay line for analog core signal timing circuit.

of devices.

Furthermore, we proposed a prototype of an A*-based routing engine that integrates fundamental DRC rules. This ensures that the complete layouts generated in the simulated environment are correct by construction and can, in the future, be seamlessly imported into any commercial EDA software suite. The results demonstrate that the combination of these improvements leads to a significant reduction in routing failure rates compared to the previous floorplanning engine.

However, additional improvements can be pursued. Future efforts will focus on addressing the overestimation produced by the DRR technique. This will involve developing advanced methodologies to better estimate actual routing requirements during placement. Post-processing techniques to eliminate unnecessary spacing after the routing stage might be considered for improving floorplan efficiency while maintaining routing success. Additionally, integrating the final routing signal into the RL reward system could further influence the floorplan generation toward even more routing-friendly results. Lastly, improving the A*-based routing engine by adding more DRC rules and analog specific patterns, such as symmetry, remains a primary objective for future work.

Part III

Closing Remarks

Chapter 9

Conclusion

Analog ICs layout remains a critical step in the entire circuit design process. Despite the tentatives to automate this procedure since the first adoptions of EDA techniques, it still requires nowadays significant manual interventions to be carried out properly. The intrinsic complexity, attributed to the several electric and design constraints to be satisfied, has often led the industry to discard solutions proposed by both EDA vendors and academic research. In fact, the limited applicability of the proposed tools combined with the high standards required to guarantee circuit performance, are just some of the many reasons of the missed widespread adoption of automated methods for layouting. Currently, only few have tried to design a solution capable of leveraging a certain degree of knowledge about the layout to be produced, while strictly adhering to the aforementioned objectives.

In this thesis, we aimed to address this challenge by developing a tool tailored to meet the requirements of experienced layout engineers in the industry (more precisely Infineon). At the core of our approach, we leveraged RL as the driving engine for optimizing the floorplanning of analog ICs devices. The role of the RL agent evolved significantly throughout the course of this work.

First, the agent served as an auxiliary component within a SA-driven floorplanning procedure (*RL-SA*). In this setup, the RL agent's task was to generate initial, optimized placement configurations, which were then refined and finalized by the SA engine. Then, we expanded the agent's role, from a supporting to a central decision one (*Annealed-RL*). At this stage, the RL agent alone was responsible for producing the floorplan, by evaluating newly sampled floorplan configurations and deciding whether to accept or reject them, eventually mimicking a traditional SA-based optimization. Moreover, we cared to ensure compliance with fundamental layout constraints, such as alignment, symmetry, and overlap avoidance. To enhance the effectiveness of our approach, we devised a methodology to automatically generate multiple device configurations and shapes, significantly reducing the workload for engineers. Additionally, as a first attempt toward addressing the routing stage, an OARSMT global routing system was introduced and integrated into ANAGEN to provide guidance to interconnect the different circuit components together.

The initial presence of a SP model to mathematically represent the floorplan proved to be too restrictive for achieving optimal placement performance. Consequently, this approach was discarded in favor of an absolute coordinate system, which led to the development of the *R-GCN RL* floorplanning engine. This updated version, while still adhering to the constraints of its predecessor, introduces learned embeddings of the entire

circuit and individual devices into the RL agent knowledge, with the goal of increasing the comprehension of the problem instance to be solved. Combining these aspects together, this methodology demonstrates a remarkable ability to generalize across different and unseen circuit types, a critical requirement for industrial applications. Furthermore, the full layouts generated using the OARSMT-based routing approach exhibit quality that is comparable to layouts manually generated by human engineers, while achieving significant reductions in runtime.

We then proposed an enhancement of the R-GCN RL approach by integrating a classical combinatorial optimization algorithm, beam search, to provide more robust and efficient automated floorplanning solutions during inference (*BS-RL*). This search-based methodology, while introducing a slight runtime overhead comparable to that of a finetuning procedure, offers the flexibility to steer floorplan generation toward different objective metrics.

Thanks to thorough discussions with Infineon engineers, we recognized that producing highly compact floorplanning solutions underestimating the importance of routing-related metrics not only limits the utility of the solution but can also be counterproductive. However anticipating how routing would unfold and reducing potential source of errors that might arise during a later stage is not trivial. To address this challenge, we improved the R-GCN RL approach to be more *routing-aware* by prioritizing HPWL optimization while ensuring proper spacing between components to accommodate routing tracks. DRC violations are further mitigated thanks to the introduction of precise pin information during placement. In addition, increasing the placement grid granularity enables more compact solutions while preserving a zero-shot setup, and allows the floorplanning pipeline to remain within sub-second runtimes for the circuit sizes considered. Finally, the development of a custom A*-based rip-up and reroute routing engine enabled us to evaluate the quality of our approach within a simulated environment. The experimental results validate our insights, demonstrating that the new introduced features facilitate a seamless generation of complete layouts, with the RL agent successfully achieving an optimal tradeoff between area optimization and routing efficiency. These advancements collectively establish a strong foundation for the industrial application of such an automated framework.

9.1 Limitations and Future Research

We acknowledge that our work represents just the initial step in a promising research direction and hope it inspires further exploration and innovation. While the potential for improvements extends far beyond what is outlined hereafter, we aim for these ideas to serve as possible starting point for researchers and practitioners in the years to come.

On the floorplanning side, incorporating further analog-specific rules for the placement of devices and achieving a more refined balance between floorplanning and routing phases could yield remarkable improvements. In its current form, the routing-aware floorplanning engine uses a dynamic routing resource allocation methodology that inevitably overestimates the space required to connect components. To address this, a post-processing technique aimed at further compacting the floorplan and routing tracks would be highly beneficial. Moreover, while the current approach leverages HPWL as a proxy for routing-related signals, introducing information directly from the routing

engine, or ideally post-layout electrical simulation outcomes, into the reward function could provide richer signals to guide the optimization process. Regarding the reward definition, the current methodology treats the inherently multi-objective nature of the floorplanning problem as a single-objective task. Exploring RL algorithms explicitly designed for multi-objective optimization could represent a compelling direction for improving efficiency and effectiveness of the entire approach. Finally, although the proposed floorplanning engine demonstrates significant transferability properties, further efforts could be spent on the search-based enhancement of the RL policy's exploration capabilities. Employing more sophisticated techniques, as in AlphaGo [164], to refine and guide policy exploration could provide remarkable improvements in navigating the large floorplanning solution space.

In terms of routing, future research could focus on enhancing the A* routing engine to ensure compatibility with the several DRC and technology-induced constraints. A fundamental aspect of this task involves identifying a robust set of rules that can be effectively translated into geometrical or physics-based formulations and applied during the optimization procedure. Furthermore, incorporating learning-based methods to avoid violations leveraging past experience offers the potential to generate sign-off-ready layouts with minimal iterations. While this represents a promising direction, we are aware of the significant challenges involved in achieving this goal. From an implementation perspective, it may be interesting to explore novel fast and highly efficient programming languages, such as Rust, for searching in the routing space. This could provide a competitive edge over traditional engines, which are mainly implemented in C++, perhaps contributing to closing the gap between pure academic and industrial application.

List of Figures

1.1	Example of an analog circuit schematic.	2
1.2	Examples of a plain (a) and routed (b) floorplan of an analog circuit. . . .	3
2.1	Example of common centroid (a) and interdigitated (b) placement patterns.	12
2.2	Differential pair with dummies on the side placed in CC fashion.	13
2.3	WPE effect in CMOS technology (a) and its prevention by adding dummies (b). Adapted from [161].	14
2.4	Example of a Pcell in Cadence Virtuoso, from [173].	15
2.5	The compacted floorplan associated to $(\Gamma_+, \Gamma_-) = (ecadfb, fcbead)$. . .	16
2.6	Examples of a routing grid (a) and a via (b).	19
3.1	Simulated Annealing hill-climbing approach to escape local optima. . . .	25
3.2	Architecture of an MLP with multiple hidden layers. Every neuron executes the function defined in Equation 3.5.	31
3.3	Visual representation of a CNN architecture. The different convolutional and pooling layers are highlighted, along with the kernel width, length and depth (# of channels). A final MLP is placed as example for using learned image representation for subsequent tasks, e.g. classification. . .	33
3.4	Overview of the GCN architecture.	35
3.5	Interaction between agent and environment in Reinforcement Learning.	36
3.6	Bellman equation visualization. Each open circle represents a state and each solid circle represents a state-action pair. Starting from state s the agent could take any of some set of actions. From each of these, the environment could respond with one of several next states s' , along with a reward r . The Bellman equation averages over all the possibilities, weighting each by its probability of occurring. Figures from [163]. . . .	38
3.7	Same as fig. 3.6 but in this case arcs have been added at the agent's choice points to represent that the maximum over that choice is taken rather than the expected value given some policy. Figures from [163].	39
3.8	Plots of the “surrogate” objective L^{CLIP} as a function of the probability ratio $r(\theta)$ showing the clipping effect on positive (left) and negative (right) surrogates. Adapted from [157].	42
3.9	Plots of the “surrogate” objective L^{CLIP} as a function of the probability ratio $r(\theta)$ showing the undo regions for positive (left) and negative (right) surrogates. Adapted from [157].	43

4.1	Comparison between a complete grid graph and the Hanan grid generated from a set of four pins. The construction of the RSMT and the OARSMT is shown, with Steiner points highlighted in both cases.	48
5.1	High-level schematic of the AI-powered automated layout pipeline with this chapter contributions highlighted in green.	52
5.2	OTA schematic.	57
5.3	Floorplan and global routing configuration from Figure 5.2 (a), and rendering with ANAGEN tracklines (b).	57
5.4	Comparison of OTA layouts: (a) generated automatically via our pipeline and (b) designed manually by a physical design engineer.	58
6.1	Overview of the automatic layout pipeline enhanced with the R-GCN RL floorplanning algorithm.	62
6.2	8-structure OTA circuit schematic with its graph representation. Violet edges are for vertical alignment and black for connectivity. Nodes are colored according to the functional block type.	64
6.3	R-GCN architecture for circuit reward prediction.	64
6.4	Overview of the RL model, enriched with CNN-based feature extractor and policy network.	66
6.5	Dead space (left) and wire (right) masks. Darker areas highlight higher rewards regions in the case a block is placed there.	67
6.6	Episode reward mean and approximate Kullback-Leibler divergence during RL agent's HCL training schedule.	69
6.7	(a) RL-generated placement and OARSMT global routing, (b) post-adjustment floorplan and channels definitions, (c) resulting layout, (d) post-manual refinement optimized layout, and (e) full manual design.	72
7.1	Example state tree space for an RL agent. The root S_0 represents the initial state, at time-step 0. Nodes $S_{t;a}$ represent the states at time-step t , after taking action a	77
7.2	Example of BS execution on the OTA from Figure 5.2, with $\Lambda = 4$	79
8.1	A comparison between the previous circuit graph configuration (a) and the enhanced near-tripartite one (b). In the enhanced graph, violet edges represent the topological constraints applied between recognized structures, green ones denote the <i>device has pin</i> relationship, and blue edges connect pins and nets through the <i>pin belongs to net</i> relationship.	85
8.2	Overview of the U-Net based RL policy network.	87
8.3	Performance comparison on delay line circuit.	95
8.4	Complete layout of the delay line for analog core signal timing circuit.	96

List of Tables

5.1	Metaheuristics hyperparameters.	56
5.2	Runtime, floorplan empty space, and HPWL comparison between RL-based algorithms and traditional metaheuristics.	56
5.3	Comparative analysis of layout generation between our automated method and expert.	57
6.1	R-GCN and RL models hyperparameters.	65
6.2	Comparative analysis of R-GCN and RL method, across various fine-tuned setups, versus previous techniques.	71
6.3	Comparison of area, dead space, and layout generation time between our automated method and manual design.	73
7.1	Comparative analysis of SA, R-GCN RL, R-GCN RL-FT, BS-RL, and their variations with congestion handling ([†]).	81
8.1	Updated R-GCN hyperparameters.	86
8.2	RL policy model hyperparameters.	88
8.3	Comparison of the routing-aware approach versus R-GCN RL.	93
8.4	Routing-related metrics comparison only successfully routed floorplans. The failure rate is instead computed on all available circuits.	94

Bibliography

- [1] A. Agnesina et al. “AutoDMP: Automated DREAMPlace-based Macro Placement”. In: *Proceedings of the 2023 International Symposium on Physical Design (ISPD)*. ACM, Mar. 2023, pp. 149–157.
- [2] M. Ahmadi and L. Zhang. “Analog Layout Placement for FinFET Technology Using Reinforcement Learning”. In: *Proceedings - IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 2021-May. Institute of Electrical and Electronics Engineers Inc., 2021.
- [3] T. Akiba et al. “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. Association for Computing Machinery, 2019, pp. 2623–2631.
- [4] M. Amini et al. “Generalizable Floorplanner through Corner Block List Representation and Hypergraph Embedding”. In: *28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2022, pp. 2692–2702.
- [5] Y. P. Aneja. “An Integer Linear Programming Approach to the Steiner Problem in Graphs”. In: *Networks* 10.2 (1980), pp. 167–178.
- [6] F. Balasa and K. Lampaert. “Symmetry within the Sequence-Pair Representation in the Context of Placement for Analog Design”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 19.7 (2000), pp. 721–731.
- [7] A. G. Barto, R. S. Sutton, and C. W. Anderson. “Neuronlike Adaptive Elements that Can Solve Difficult Learning Control Problems”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (1983), pp. 834–846.
- [8] D. Basso et al. “Fast ML-Driven Analog Circuit Layout Using Reinforcement Learning and Steiner Trees”. In: *Proceedings of the 20th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. 2024, pp. 1–4.
- [9] D. Basso et al. “Advancing Routing-Awareness in Analog ICs Floorplanning”. In: *Workshop on Machine Learning for Systems. Proceedings of the 39th International Conference on Neural Information Processing Systems (NeurIPS)*. 2025.
- [10] D. Basso et al. “Effective Analog ICs Floorplanning with Relational Graph Neural Networks and Reinforcement Learning”. In: *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*. 2025, pp. 1–7.
- [11] P. W. Battaglia et al. “Relational Inductive Biases, Deep Learning, and Graph Networks”. In: *arXiv* (Oct. 2018).

- [12] R. Bellman. “A Markovian Decision Process”. In: *Journal of Mathematics and Mechanics* 6.5 (1957), pp. 679–684.
- [13] J. A. Bennell, M. Cabo, and A. Martínez-Sykora. “A Beam Search Approach to Solve the Convex Irregular Bin Packing Problem with Guillotine Cuts”. In: *European Journal of Operational Research* 270.1 (2018), pp. 89–102.
- [14] J. A. Bennell and X. Song. “A Beam Search Implementation for the Irregular Shape Packing Problem”. In: *Journal of Heuristics* 16.2 (Apr. 2010), pp. 167–188.
- [15] C. M. Bishop. “Pattern Recognition and Machine Learning (Information Science and Statistics)”. Berlin, Heidelberg: Springer-Verlag, 2006.
- [16] Cadence Design Systems. “Spectre X Simulator”. 2024.
- [17] Cadence Design Systems. “Virtuoso System Design Platform: A Comprehensive Solution for Analog, RF, and Mixed-Signal Design”. 2025.
- [18] Q. Cai et al. “Reinforcement Learning Driven Heuristic Optimization”. In: *CoRR* (2019). arXiv: 1906.06639.
- [19] A. Chakraborty and A. K. Kar. “Swarm Intelligence: A Review of Algorithms”. In: *Nature-Inspired Computing and Optimization: Theory and Applications*. Cham: Springer International Publishing, 2017, pp. 475–494.
- [20] Y.-C. Chang et al. “B*-Trees: A New Representation for Non-Slicing Floorplans”. In: *Proceedings 37th Design Automation Conference (DAC)*. 2000, pp. 458–463.
- [21] H. Chen et al. “Challenges and Opportunities Toward Fully Automated Analog Layout Design”. In: *Journal of Semiconductors* 41.11 (Nov. 2020), p. 111407.
- [22] H. Chen et al. “Toward Silicon-Proven Detailed Routing for Analog and Mixed-Signal Circuits”. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2020.
- [23] H. Chen et al. “MAGICAL: An Open-Source Fully Automated Analog IC Layout System from Netlist to GDSII”. en. In: *IEEE Design & Test* 38.2 (Apr. 2021), pp. 19–26.
- [24] H. Chen et al. “Universal Symmetry Constraint Extraction for Analog and Mixed-Signal Circuits with Graph Neural Networks”. In: *Proceeding of the 58th ACM/IEEE Design Automation Conference (DAC)*. Dec. 2021, pp. 1243–1248.
- [25] H. Chen et al. “Reinforcement Learning Guided Detailed Routing for Custom Circuits”. en. In: *Proceedings of the 2023 International Symposium on Physical Design (ISPD)*. Virtual Event, USA: ACM, Mar. 2023, pp. 26–34.
- [26] J. Chen and N. R. Sturtevant. “Necessary and Sufficient Conditions for Avoiding Reopenings in Best First Suboptimal Search with General Bounding Functions”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.5 (May 2021), pp. 3688–3696.
- [27] K. Chen and G. G. E. Gielen. “Self-Learning and Transfer Across Topologies of Constraints for Analog / Mixed-Signal Circuit Layout Synthesis”. In: *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2024, pp. 1–6.

- [28] T.-C. Chen and Y.-W. Chang. “Modern Floorplanning Based on B/sup */-Tree and Fast Simulated Annealing”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 25.4 (2006), pp. 637–650.
- [29] T.-C. Chen and Y. Chang. “Electronic Design Automation”. Morgan Kaufmann (Elsevier), 2009.
- [30] P.-Y. Chen et al. “A Reinforcement Learning Agent for Obstacle-Avoiding Rectilinear Steiner Tree Construction”. In: *Proceedings of the 2022 International Symposium on Physical Design (ISPD)*. ISPD ’22. Virtual Event, Canada: Association for Computing Machinery, 2022, pp. 107–115.
- [31] R. Cheng and J. Yan. “On Joint Learning for Solving Placement and Routing in Chip Design”. In: *Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS)*. Curran Associates Inc., 2021.
- [32] J. Choo et al. “Simulation-guided Beam Search for Neural Combinatorial Optimization”. In: *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*. NIPS ’22. New Orleans, LA, USA: Curran Associates Inc., 2022.
- [33] U. Choudhury and A. Sangiovanni-Vincentelli. “Automatic Generation of Parasitic Constraints for Performance-Constrained Physical Design of Analog Circuits”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 12.2 (1993), pp. 208–224.
- [34] W.-K. Chow et al. “Obstacle-Avoiding Rectilinear Steiner Tree Construction in Sequential and Parallel Approach”. In: *Integration* 47.1 (2014), pp. 105–114.
- [35] C. Chu and Y.-C. Wong. “FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 27.1 (2007), pp. 70–83.
- [36] T. H. Cormen et al. “Introduction to Algorithms”. MIT Press, 2022.
- [37] C. Cortes and V. Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297.
- [38] S. J. Della Rovere et al. “Enhancing Reinforcement Learning for the Floorplanning of Analog ICs with Beam Search”. In: *Proceedings of the 21st International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. IEEE, July 2025.
- [39] D. Demiri et al. “A Procedural Generator for the Sizing and Physical Synthesis of a MOSFET Low-Side Driver”. In: *Proceedings of the 19th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. 2023, pp. 1–4.
- [40] D. N. Deutsch. “A “Dogleg” Channel Router”. In: *Proceedings of the 13th Design Automation Conference (DAC)*. DAC ’76. San Francisco, California, USA: Association for Computing Machinery, 1976, pp. 425–433.
- [41] T. Dhar et al. “ALIGN: A System for Automating Analog Layout”. In: *IEEE Design & Test* 38.2 (2021), pp. 8–18.

- [42] E. W. Dijkstra. “A Note on Two Problems in Connexion with Graphs”. In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.
- [43] F. Duchoň et al. “Path Planning with Modified A Star Algorithm for a Mobile Robot”. In: *Modelling of Mechanical and Mechatronic Systems* 96 (Jan. 2014), pp. 59–69.
- [44] R. Ettrich, M. Huber, and G. R. Raidl. “A Policy-Based Learning Beam Search for Combinatorial Optimization”. In: *Evolutionary Computation in Combinatorial Optimization*. Ed. by L. Pérez Cáceres and T. Stützle. Cham: Springer Nature Switzerland, 2023, pp. 130–145.
- [45] A. Fallin et al. “A Simple, Fast, and GPU-Friendly Steiner-Tree Heuristic”. In: *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, May 2022, pp. 838–847.
- [46] M. Fampa, J. Lee, and W. Melo. “A Specialized Branch-and-Bound Algorithm for the Euclidean Steiner Tree Problem in N-Space”. In: *Computational Optimization and Applications* 65.1 (Sept. 2016), pp. 47–71.
- [47] D. Foead et al. “A Systematic Literature Review of A* Pathfinding”. In: *Procedia Computer Science* 179 (2021), pp. 507–514.
- [48] A. G. Gad. “Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review”. en. In: *Archives of Computational Methods in Engineering* 29.5 (Aug. 2022), pp. 2531–2561.
- [49] Z. Geng et al. “Reinforcement Learning within Tree Search for Fast Macro Placement”. In: *Proceedings of the 41st International Conference on Machine Learning (ICML)*. ICML ’24. Vienna, Austria: JMLR.org, 2024.
- [50] J. Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1263–1272.
- [51] A. Goldie et al. “Addendum: A Graph Placement Methodology for Fast Chip Design”. In: *Nature* 634.8034 (Oct. 2024), E10–E11.
- [52] I. Goodfellow, Y. Bengio, and A. Courville. “Deep Learning”. MIT Press, 2016.
- [53] M. Gori, G. Monfardini, and F. Scarselli. “A New Model for Learning in Graph Domains”. In: *Proceedings of the IEEE International Joint Conference on Neural Networks*. Vol. 2. July 2005, 729–734 vol. 2.
- [54] A. Goyal and Y. Bengio. “Inductive Biases for Deep Learning of Higher-Level Cognition”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 478.2266 (Oct. 2022), p. 20210068.
- [55] D. Guerra et al. “Artificial Neural Networks as an Alternative for Automatic Analog IC Placement”. In: *Proceedings of the 16th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design, Proceedings (SMACD)*. Institute of Electrical and Electronics Engineers Inc., July 2019, pp. 1–4.

- [56] A. Gusmão et al. “A Deep Learning Toolbox for Analog Integrated Circuit Placement”. In: *Proceedings of the 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design, Proceedings (SMACD)*. IEEE, Sept. 2021, pp. 1–5.
- [57] A. Gusmão et al. “DeepPlacer: A Custom Integrated OpAmp Placement Tool Using Deep Models”. In: *Applied Soft Computing* 115 (Jan. 2022), p. 108188.
- [58] A. Gusmão et al. “Exploiting a Deep Learning Toolbox for Human-Machine Feedback Towards Analog Integrated Circuit Placement Automation”. In: *Electronics* 11 (Nov. 2022), p. 3964.
- [59] A. P. L. d. Gusmão et al. “Scalable and Order Invariant Analog Integrated Circuit Placement with Attention-Based Graph-to-Sequence Deep Models”. In: *Expert Systems with Applications* 207 (2022), p. 117954.
- [60] S. H. Haji and A. M. A. Abdulazeez. “Comparison of Optimization Techniques Based on Gradient Descent Algorithm: A Review”. In: *PalArch’s Journal of Archaeology of Egypt / Egyptology* 18.4 (Feb. 2021), pp. 2715–2743.
- [61] W. L. Hamilton. “Graph Representation Learning”. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020.
- [62] J. Han et al. “LAYGO: A Template-and-Grid-Based Layout Generation Engine for Advanced CMOS Technologies”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.3 (2021), pp. 1012–1022.
- [63] M. Hanan. “On Steiner’s Problem with Rectilinear Distance”. In: *SIAM Journal on Applied Mathematics* 14.2 (1966), pp. 255–265.
- [64] F. E. Harrell. “Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis”. Springer Series in Statistics. Springer International Publishing, 2015.
- [65] P. E. Hart, N. J. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.
- [66] R. Hastings. “The Art of Analog Layout”. Pearson Prentice Hall, 2024.
- [67] W. K. Hastings. “Monte Carlo Sampling Methods Using Markov Chains and Their Applications”. In: *Biometrika* 57.1 (1970), pp. 97–109.
- [68] K. He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.
- [69] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780.
- [70] J. H. Holland. “Genetic Algorithms”. In: *Scientific American* 267.1 (1992), pp. 66–73.
- [71] X. Hong et al. “Corner Block List Representation and Its Application to Floorplan Optimization”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 51.5 (2004), pp. 228–233.

- [72] S. Huang and S. Ontañón. “A Closer Look at Invalid Action Masking in Policy Gradient Algorithms”. In: *The International FLAIRS Conference Proceedings*. Vol. 35. 2022.
- [73] T. Huang, L. Li, and E. F. Y. Young. “On the Construction of Optimal Obstacle-Avoiding Rectilinear Steiner Minimum Trees”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 30.5 (May 2011), pp. 718–731.
- [74] T. Huang and E. F. Y. Young. “Obstacle-avoiding rectilinear Steiner minimum tree construction: An optimal approach”. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2010, pp. 610–613.
- [75] M. Huber and G. R. Raidl. “Learning Beam Search: Utilizing Machine Learning to Guide Beam Search for Solving Combinatorial Optimization Problems”. In: *Machine Learning, Optimization, and Data Science: 7th International Conference, LOD 2021, Grasmere, UK, October 4–8, 2021, Revised Selected Papers, Part II*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 283–298.
- [76] F. K. Hwang. “On Steiner Minimal Trees with Rectilinear Distance”. In: *SIAM Journal on Applied Mathematics* 30.1 (1976), pp. 104–114.
- [77] C. Janiesch, P. Zschech, and K. Heinrich. “Machine Learning and Deep Learning”. en. In: *Electronic Markets* 31.3 (Sept. 2021), pp. 685–695.
- [78] K. Jebari. “Selection Methods for Genetic Algorithms”. In: *International Journal of Emerging Sciences* 3 (Dec. 2013), pp. 333–344.
- [79] X. Jin and J. Han. “K-Means Clustering”. In: *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010, pp. 563–564.
- [80] A. Kahng and G. Robins. “A New Class of Steiner Tree Heuristics with Good Performance: The Iterated 1-Steiner Approach”. In: *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD)*. 1990, pp. 428–431.
- [81] A. B. Kahng. “Classical Floorplanning Harmful?” In: *Proceedings of the 2000 International Symposium on Physical Design*. ISPD ’00. San Diego, California, USA: Association for Computing Machinery, 2000, pp. 207–213.
- [82] A. B. Kahng et al. “NN-steiner: A Mixed Neural-Algorithmic Approach for the Rectilinear Steiner Minimum Tree Problem”. In: *Proceedings of the 38th AAAI Conference on Artificial Intelligence, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, and Fourteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI’24/IAAI’24/EAAI’24. AAAI Press, 2024.
- [83] N. Karmokar et al. “Common-Centroid Layout for Active and Passive Devices: A Review and the Road Ahead”. In: *Proceedings of the 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2022, pp. 114–121.
- [84] J. Kennedy and R. Eberhart. “Particle Swarm Optimization”. In: *Proceedings of ICNN’95 - International Conference on Neural Networks*. Vol. 4. Nov. 1995, 1942–1948 vol.4.
- [85] P. R. Kinget. “Device Mismatch and Tradeoffs in the Design of Analog Circuits”. In: *IEEE Journal of Solid-State Circuits* 40.6 (2005), pp. 1212–1224.

- [86] D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes”. In: *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. Ed. by Y. Bengio and Y. LeCun. 2014.
- [87] T. N. Kipf and M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *Proceedings of the 5th International Conference on Learning Representations (ICLR) (ICLR)*. ICLR ’17. Palais des Congrès Neptune, Toulon, France, 2017.
- [88] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680.
- [89] J. Kober and J. Peters. “Reinforcement Learning in Robotics: A Survey”. In: *Learning Motor Skills: From Algorithms to Robot Experiments*. Cham: Springer International Publishing, 2014, pp. 9–67.
- [90] P. Kora and P. Yadlapalli. “Crossover Operators in Genetic Algorithms: A Review”. In: *International Journal of Computer Applications* 162 (Mar. 2017), pp. 34–36.
- [91] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems (NeurIPS)*. NIPS’12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.
- [92] K. Kunal et al. “A General Approach for Identifying Hierarchical Symmetry Constraints for Analog Circuit Layout”. In: *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*. New York, NY, USA: Association for Computing Machinery, 2020.
- [93] Y. Lai, Y. Mu, and P. Luo. “MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning”. In: *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2022, pp. 24019–24030.
- [94] Y. Lai et al. “ChiPFormer: Transferable Chip Placement via Offline Decision Transformer”. In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. JMLR.org, 2023, pp. 18346–18364.
- [95] S. R. Lawande et al. “A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games”. In: *Applied Sciences* 12.11 (2022).
- [96] T. P. Le et al. “Toward Reinforcement Learning-based Rectilinear Macro Placement Under Human Constraints”. In: *CoRR* (2023). arXiv: 2311.03383.
- [97] Y. LeCun, Y. Bengio, and G. Hinton. “Deep Learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444.
- [98] Y. LeCun et al. “Handwritten Digit Recognition with a Back-Propagation Network”. In: *Advances in Neural Information Processing Systems 2 (NIPS Conference, Denver, Colorado, USA, November 27-30, 1989)*. 1989, pp. 396–404.
- [99] M. Leitner-Ankerl. “A fast & densely stored hashmap and hashset based on robin-hood backward shift deletion”. C++. Aug. 2025.

- [100] L. Li and E. F. Y. Young. "Obstacle-Avoiding Rectilinear Steiner Tree Construction". In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2008, pp. 523–528.
- [101] Y. Li et al. "A Customized Graph Neural Network Model for Guiding Analog IC Placement". In: *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*. Vol. 2020-November. New York, NY, USA: ACM, Nov. 2020, pp. 1–9.
- [102] Y. Li et al. "A Circuit Attention Network-Based Actor-Critic Learning Approach to Robust Analog Transistor Sizing". In: *ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*. 2021, pp. 1–6.
- [103] H. Liao et al. "A Deep Reinforcement Learning Approach for Global Routing". In: *Journal of Mechanical Design* 142.061701 (Nov. 2019).
- [104] Z. Lin et al. "Obstacle-Avoiding Rectilinear Steiner Minimal Tree Algorithm Based on Deep Reinforcement Learning". In: *2023 International Conference on Artificial Intelligence of Things and Systems (AIoTSys)*. Xi'an, China: IEEE, Oct. 2023, pp. 149–156.
- [105] C.-H. Liu et al. "An $O(n \log n)$ Path-Based Obstacle-Avoiding Algorithm for Rectilinear Steiner Tree Construction". In: *Proceedings of the 46th Annual Design Automation Conference (DAC)*. 2009, pp. 314–319.
- [106] C.-H. Liu et al. "Obstacle-Avoiding Rectilinear Steiner Tree Construction: A Steiner-Point-Based Algorithm". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 31.7 (2012), pp. 1050–1060.
- [107] J. Liu et al. "Symmetry Constraint Based on Mismatch Analysis for Analog Layout in SOI Technology". In: *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2008, pp. 772–775.
- [108] M. Liu et al. "S3DET: Detecting System Symmetry Constraints for Analog Circuits with Graph Similarity". In: *Proceeding of the 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2020, pp. 193–198.
- [109] M. Liu et al. "Towards Decrypting the Art of Analog Layout: Placement Quality Prediction via Transfer Learning". In: *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Mar. 2020, pp. 496–501.
- [110] I. Ljubić. "Solving Steiner Trees: Recent Advances, Challenges, and Perspectives". In: *Networks* 77.2 (2021), pp. 177–204.
- [111] N. Lohmann. "JSON for Modern C++". C++. Apr. 2025.
- [112] D. S. Lopera et al. "A Survey of Graph Neural Networks for Electronic Design Automation". In: *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*. Raleigh, NC, USA: IEEE, Aug. 2021, pp. 1–6.
- [113] N. Lourenco et al. "LAYGEN - Automatic Layout Generation of Analog ICs from Hierarchical Template Descriptions". In: *2006 Ph.D. Research in Microelectronics and Electronics*. 2006, pp. 213–216.

- [114] B. Lowerre and R. Reddy. "The Harpy Speech Recognition System: performance with large vocabularies". In: *The Journal of the Acoustical Society of America*, AIP Publishing 60.S1 (Nov. 1976), S10–S11.
- [115] Q. Ma et al. "Simultaneous Handling of Symmetry, Common Centroid, and General Placement Constraints". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 30.1 (2011), pp. 85–95.
- [116] A. Maćkiewicz and W. Ratajczak. "Principal Components Analysis (PCA)". In: *Computers & Geosciences* 19.3 (1993), pp. 303–342.
- [117] O. Z. Maimon and L. Rokach. "Data Mining with Decision Trees: Theory and Applications". Vol. 81. World Scientific, 2014.
- [118] E. Malavasi et al. "Automation of IC Layout with Analog Constraints". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 15.8 (1996), pp. 923–942.
- [119] I. L. Markov. "Reevaluating Google's Reinforcement Learning for IC Macro Placement". In: *Commun. ACM* 67.11 (Oct. 2024), pp. 60–71.
- [120] R. Martins. "Closing the Gap Between Electrical and Physical Design Steps with an Analog IC Placement Optimizer Enhanced with Machine-Learning-Based Post-Layout Performance Regressors". In: *Electronics* 13.22 (Jan. 2024), p. 4360.
- [121] R. Martins, N. Lourenco, and N. Horta. "LAYGEN II—Automatic Layout Generation of Analog Integrated Circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 32 (Nov. 2013), pp. 1641–1654.
- [122] R. Martins et al. "AIDA: Robust Layout-Aware Synthesis of Analog ICs Including Sizing and Layout Generation". In: *Proceedings of the International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. IEEE. 2015, pp. 1–4.
- [123] N. Mazyavkina et al. "Reinforcement Learning for Combinatorial Optimization: A Survey". In: *Computers & Operations Research* 134 (2021), p. 105400.
- [124] L. McMurchie and C. Ebeling. "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs". In: *Proceedings of the 1995 ACM Third International Symposium on Field-Programmable Gate Arrays*. FPGA '95. New York, NY, USA: Association for Computing Machinery, 1995, pp. 111–117.
- [125] N. Metropolis et al. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092.
- [126] A. Mirhoseini et al. "A graph placement methodology for fast chip design". In: *Nature* 594.7862 (2021), pp. 207–212.
- [127] V. Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* (2013). arXiv: 1312.5602.
- [128] H. Murata et al. "Rectangle-Packing-Based Module Placement". In: *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*. 1995, pp. 472–479.
- [129] K. P. Murphy. "Machine Learning: A Probabilistic Perspective". The MIT Press, 2012.

- [130] V. Nagendra and S. Manavalan. "Optimization of Placement and Routing Techniques: Congestion Estimation and Control in VLSI Design". In: *Proceedings of the 5th International Conference on Data Science, Machine Learning and Applications; Volume 1*. Ed. by A. Kumar et al. Singapore: Springer Nature Singapore, 2025, pp. 269–276.
- [131] S. Nakaya, T. Koide, and S. Wakabayashi. "An Adaptive Genetic Algorithm for VLSI Floorplanning Based on Sequence-Pair". In: *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 3. 2000, 65–68 vol.3.
- [132] Y. Nourani and B. Andresen. "A Comparison of Simulated Annealing Cooling Strategies". In: *Journal of Physics A: Mathematical and General* 31.41 (Oct. 1998), p. 8373.
- [133] I. Nusrat and S.-B. Jang. "A Comparison of Regularization Techniques in Deep Neural Networks". In: *Symmetry* 10.11 (2018), p. 648.
- [134] H.-C. Ou, H.-C. C. Chien, and Y.-W. Chang. "Simultaneous Analog Placement and Routing with Current Flow and Current Density Considerations". In: *Proceedings of the 50th Annual Design Automation Conference (DAC)*. DAC '13. New York, NY, USA: Association for Computing Machinery, 2013, 5:1–5:6.
- [135] H.-C. Ou et al. "Layout-Dependent Effects-Aware Analytical Analog Placement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 35.8 (2016), pp. 1243–1254.
- [136] F. Passerini et al. "ANAGEN: A Methodology for ANalog Circuit GENeration". In: *European Solid-State Circuits' Conference Educational Events & European Solid-State Device Research Conference*. IEEE, Sept. 2021.
- [137] R. Patel, H. Habal, and K. R. Venkata. "Machine Learning based Structure Recognition in Analog Schematics for Constraints Generation". In: *Design and Verification Conference and Exhibition Europe*. Accellera Systems Initiative, Oct. 2021.
- [138] V. Pathak et al. "Analog Circuit Classification Using Graph Convolution Networks". In: *Proceedings of the 20th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. July 2024, pp. 1–4.
- [139] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers. "Matching Properties of MOS Transistors". In: *IEEE Journal of Solid-State Circuits* 24.5 (1989), pp. 1433–1439.
- [140] M. J. M. Pelgrom, H. P. Tuinhout, and M. Vertregt. "Transistor Matching in Analog CMOS Applications". In: *International Electron Devices Meeting 1998. Technical Digest (Cat. No.98CH36217)*. 1998, pp. 915–918.
- [141] S. Poddar et al. "A Data-Driven Analog Circuit Synthesizer with Automatic Topology Selection and Sizing". In: *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2024, pp. 1–6.
- [142] B. Prautsch, U. Eichler, and U. Hatnik. "Generating the Generator: A User-Driven and Template-Based Approach towards Analog Layout Automation". In: *Electronics* 12.4 (2023).

- [143] Pulsic. Jan. 29, 2024.
- [144] T. Qu et al. “Asynchronous Reinforcement Learning Framework for Net Order Exploration in Detailed Routing”. In: *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Feb. 2021, pp. 1815–1820.
- [145] A. Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8.
- [146] H. E. Robbins. “A Stochastic Approximation Method”. In: *Annals of Mathematical Statistics* 22 (1951), pp. 400–407.
- [147] O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by N. Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241.
- [148] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. In: *Psychological Review* 65.6 (1958), pp. 386–408.
- [149] S. J. D. Rovere. “Enhancing Reinforcement Learning for the Floorplanning of Analog Integrated Circuits with Beam Search”. MA thesis. University of Trieste, 2025.
- [150] S. Ruder. “An Overview of Gradient Descent Optimization Algorithms”. In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747.
- [151] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning Representations by Back-Propagating Errors”. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536.
- [152] B. Sanchez-Lengeling et al. “A Gentle Introduction to Graph Neural Networks”. In: *Distill* 6.8 (Aug. 2021).
- [153] J. Scheible and J. Lienig. “Automation of Analog IC Layout: Challenges and Solutions”. In: *Proceedings of the 2015 Symposium on International Symposium on Physical Design (ISPD)*. ISPD ’15. Monterey, California, USA: Association for Computing Machinery, 2015, pp. 33–40.
- [154] M. S. Schlichtkrull et al. “Modeling Relational Data with Graph Convolutional Networks”. In: *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*. Ed. by A. Gangemi et al. Vol. 10843. Lecture Notes in Computer Science. Springer, 2018, pp. 593–607.
- [155] B. Schling. “The Boost C++ Libraries”. XML Press, 2011.
- [156] R. M. Schmidt. “Recurrent Neural Networks (RNNs): A Gentle Introduction and Overview”. In: *arXiv* (Nov. 2019).
- [157] J. Schulman et al. “Proximal Policy Optimization Algorithms”. In: *CoRR* (2017). arXiv: 1707.06347.
- [158] C.-W. Sham, E. F. Y. Young, and J. Lu. “Congestion Prediction in Early Stages of Physical Design”. In: *ACM Transactions on Design Automation of Electronic Systems* 14.1 (Jan. 2009), pp. 1–18.

- [159] A. K. Sharma et al. "Performance-Aware Common-Centroid Placement and Routing of Transistor Arrays in Analog Circuits". In: *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. Vol. 2021-November. IEEE, Nov. 2021, pp. 1–9.
- [160] Y. Shi et al. "Macro Placement by Wire-Mask-Guided Black-Box Optimization". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS)*. NIPS '23. Curran Associates Inc., 2023.
- [161] A. Sidun. "Layout-Dependent Effects". 2024.
- [162] Siemens EDA. "Calibre Design". Accessed: 2024-01-11. 2024.
- [163] D. Silver. "Lectures on Reinforcement Learning". 2015.
- [164] D. Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489.
- [165] D. Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *CoRR* (2017). arXiv: 1712.01815.
- [166] K. P. Sinaga and M.-S. Yang. "Unsupervised K-Means Clustering Algorithm". In: *IEEE Access* 8 (2020), pp. 80716–80727.
- [167] J. C. Spall. "Stochastic Optimization". en. In: 1st. *Handbook of Computational Statistics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Chap. 7, pp. 173–201.
- [168] P. Spindler and F. M. Johannes. "Fast and Accurate Routing Demand Estimation for Efficient Routability-Driven Placement". In: *Proceedings of the Design, Automation and Test in Europe. DATE '07*. San Jose, CA, USA: EDA Consortium, 2007, pp. 1226–1231.
- [169] M. Stănescu et al. "Multi-Objective Optimization Algorithms for Automated Circuit Sizing of Analog/Mixed-Signal Circuits". In: *International Semiconductor Conference (CAS)*. 2021, pp. 117–120.
- [170] R. S. Sutton and A. Barto. "Reinforcement learning: an introduction". The MIT Press, 2020.
- [171] R. S. Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999.
- [172] Synopsys. "AMS Simulation". Accessed: 2024-01-11. 2024.
- [173] C. D. Systems. "Cadence PCell Designer". 2025.
- [174] M. Torabi and L. Zhang. "Electromigration- and Parasitic-Aware ILP-Based Analog Router". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.10 (Oct. 2018), pp. 1854–1867.
- [175] L.-Y. Tseng and T.-Y. Han. "A Hybrid Genetic Algorithm for the Floorplan Optimization Problem". In: *International Symposium on Signals, Circuits and Systems, 2005 (ISSCS 2005)*. Vol. 2. 2005, 781–784 Vol. 2.

- [176] H. Wang et al. "HAS: Hierarchical A-Star Algorithm for Big Map Navigation in Special Areas". In: *2014 5th International Conference on Digital Home*. 2014, pp. 222–225.
- [177] L. Wang et al. "AIPlace: Analog IC Placement with Multi-Task Learning Framework". In: *Proceedings of the 30th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Tokyo, Japan: ACM, Jan. 2025, pp. 204–210.
- [178] M. Wang et al. "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks". In: *CoRR* (2020). arXiv: 1909.01315.
- [179] C. Watkins. "Learning from Delayed Rewards". PhD thesis. King's College, Oxford, 1989.
- [180] P.-H. Wei and B. Murmann. "Analog and Mixed-Signal Layout Automation Using Digital Place-and-Route Tools". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29.11 (Nov. 2021), pp. 1838–1849.
- [181] R. J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3 (May 1992), pp. 229–256.
- [182] D. F. Wong and C. L. Liu. "A New Algorithm for Floorplan Design". In: *23rd ACM/IEEE Design Automation Conference (DAC)*. 1986, pp. 101–107.
- [183] L. Xiao and E. F. Y. Young. "Analog Placement with Common Centroid and 1-D Symmetry Constraints". In: *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, Jan. 2009, pp. 353–360.
- [184] B. Xu et al. "MAGICAL: Toward Fully Automated Analog IC Layout Leveraging Human and Machine Intelligence: Invited Paper". In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2019, pp. 1–8.
- [185] Q. Xu et al. "GoodFloorplan: Graph Convolutional Network and Reinforcement Learning-Based Floorplanning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 41.10 (2022), pp. 3492–3502.
- [186] B. Yang et al. "Miracle: Multi-Action Reinforcement Learning-Based Chip Floorplanning Reasoner". In: *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2024, pp. 1–6.
- [187] W. Zaremba and I. Sutskever. "Learning to Execute". In: *CoRR* abs/1410.4615 (2014).
- [188] H. Zhang et al. "SAGERoute: Synergistic Analog Routing Considering Geometric and Electrical Constraints with Manual Design Compatibility". en. In: *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Antwerp, Belgium: IEEE, Apr. 2023, pp. 1–6.
- [189] H. Zhang et al. "SAGERoute 2.0: Hierarchical Analog and Mixed Signal Routing Considering Versatile Routing Scenarios". In: *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Mar. 2024, pp. 1–6.
- [190] M. Zhang and P. Li. "Nested Graph Neural Networks". In: *Proceedings of the 35th International Conference on Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates Inc., 2021, pp. 15734–15747.
- [191] X. Zhao et al. "A Review of Convolutional Neural Networks in Computer Vision". en. In: *Artificial Intelligence Review* 57.4 (Mar. 2024), p. 99.

- [192] R. Zhong et al. “FlexPlanner: Flexible 3D Floorplanning via Deep Reinforcement Learning in Hybrid Action Space with Multi-Modality Representation”. In: *Proceedings of the 38th International Conference on Neural Information Processing Systems*. Curran Associates, Inc., 2024, pp. 49252–49278.
- [193] K. Zhu et al. “GeniusRoute: A New Analog Routing Paradigm Using Generative Neural Network Guidance”. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2019, pp. 1–8.
- [194] K. Zhu et al. “Automating Analog Constraint Extraction: From Heuristics to Learning”. In: *Proceedings of the 27th Asia and South Pacific Design Automation Conference*. ASPDAC '22. Taipei, Taiwan: IEEE Press, 2022, pp. 108–113.

Appendix A

Supplementary Material for Chapter 8

Lexicographic normalization. We tried an alternative approach to normalize the different reward terms defined in Equation 6.5. Given the diverse scale of the objectives, a typical approach to make these quantities comparable involves normalizing the metrics to a $[0, 1]$ range using *min-max normalization*, also known as feature scaling, which is defined as:

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}.$$

However, this approach is not feasible in our case, as the boundary values for both area and HPWL are theoretically unknown, making it impossible to define $\min(x)$ or $\max(x)$ accurately.

To address this limitation, an alternative method can be adopted to scale the functions directly into the $[0, 1]$ range by performing a linear transformation of the form:

$$\hat{f}_1(x) = \frac{f_1(x) - f_1(x^{ul})}{f_1(x^{lr}) - f_1(x^{ul})}, \quad (\text{A.1})$$

$$\hat{f}_2(x) = \frac{f_2(x) - f_2(x^{lr})}{f_2(x^{ul}) - f_2(x^{lr})}, \quad (\text{A.2})$$

where the transformation effectively maps the Pareto front into the square $[0, 1] \times [0, 1]$. Here, x^{ul} and x^{lr} represent the lexicographic solutions of the two objectives and are defined as:

$$x^{ul} \in \arg \text{lex min} \{ (f_1(x), f_2(x)) : x \text{ feasible} \}, \quad (\text{A.3})$$

$$x^{lr} \in \arg \text{lex min} \{ (f_2(x), f_1(x)) : x \text{ feasible} \}. \quad (\text{A.4})$$

$$(\text{A.5})$$

To compute x^{ul} and x^{lr} , we solve two weighted sum scalarizations as follows:

$$\min w_1 f_1(x) + w_2 f_2(x) \quad (\text{A.6})$$

by appropriately choosing the weights. For instance, x^{ul} can be obtained by selecting weights such that $w_1 \gg w_2$, while x^{lr} can be computed by reversing the priority, i.e., $w_2 \gg w_1$.

In our approach, these solutions are generated using a traditional SA-based floorplanning engine, optimizing separately for area and HPWL.