



UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

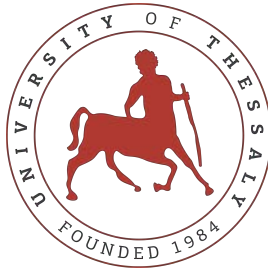
**Packing-based Macro Placement Algorithm for VLSI
Circuits**

Diploma Thesis

Marios Karagiannis

Supervisor: Christos Sotiriou

September 2024



UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Packing-based Macro Placement Algorithm for VLSI Circuits

Diploma Thesis

Marios Karagiannis

Supervisor: Christos Sotiriou

September 2024



ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

**Χωρική Τοποθέτηση Μεγάλων Αντικειμένων για
Ολοκληρωμένα Κυκλώματα πολύ Μεγάλης Κλίμακας με
Βάση Αλγορίθμους Πακετοποίησης**

Διπλωματική Εργασία

Μάριος Καραγιάννης

Επιβλέπων: Χρήστος Σωτηρίου

Σεπτέμβριος 2024

Approved by the Examination Committee:

Supervisor **Christos Sotiriou**

Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Fotios Plessas**

Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Georgios Stamoulis**

Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

First, I would like to thank my supervisor, Prof. Christos Sotiriou, for his invaluable advice and guidance, as well as for his influence on my academic interests and career. I am truly grateful to have grown as an engineer under his mentorship.

I would also like to extend my gratitude to all former and current members of CASLab for their support and help whenever I needed it. I am especially thankful to Nikolaos Sketopoulos, whose continuous assistance and support were instrumental in the completion of this thesis, and to George Goudroumanis for his guidance in the early stages of planning and organizing this work.

Last but not least, I would like to sincerely thank my friends and family for their constant support throughout my five years of study at the University of Thessaly. In particular, I am deeply grateful to my partner, Maria, for her understanding, patience, and unconditional love.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Marios Karagiannis

Diploma Thesis

Packing-based Macro Placement Algorithm for VLSI Circuits

Marios Karagiannis

Abstract

As modern VLSI circuits develop rapidly, they incorporate an increasing number of large objects, such as memories and processors, posing new challenges for the EDA industry. These large objects, known as Hard Macros, can negatively affect the operation of a digital circuit, leading to difficulties throughout the entire Physical Design flow. Consequently, there is a growing need for innovative and creative algorithms that determine the optimal placement of Hard Macros.

While these cells could originally be handled by Floorplanning approaches, the increasing number of Hard Macros in modern designs has rendered these methods unable to scale. Therefore, Hard Macros are now handled by Macro Placement algorithms which consider multiple design criteria, and aim to support the placement of Standard Cells. In this work, a Macro Placement algorithm based on the Packing method is presented, along with experimental results that demonstrate both its capabilities when customized, and its improved results compared to the simultaneous placement of Macros and Standard Cells.

Keywords:

EDA, VLSI, Physical Design, Hard Macros, Macro Placement, Packing

Διπλωματική Εργασία

Χωρική Τοποθέτηση Μεγάλων Αντικειμένων για Ολοκληρωμένα

Κυκλώματα πολύ Μεγάλης Κλίμακας με Βάση Αλγορίθμους

Πακετοποίησης

Μάριος Καραγιάννης

Περίληψη

Καθώς τα σύγχρονα κυκλώματα VLSI αναπτύσσονται με ραγδαίους ρυθμούς, χρησιμοποιούν έναν αυξανόμενο αριθμό από μεγάλα αντικείμενα, όπως μνήμες ή επεξεργαστικές μονάδες, δημιουργώντας με αυτόν τον τρόπο καινούριες προκλήσεις για τη βιομηχανία αυτοματοποίησης της σχεδίασης των κυκλωμάτων. Τα μεγάλα αυτά αντικείμενα, γνωστά και ως Hard Macros, μπορούν να επηρεάσουν αρνητικά τη λειτουργία ενός ψηφιακού κυκλώματος, οδηγώντας σε δυσκολίες κατά το στάδιο της φυσικής σχεδίασής του. Συνεπώς, εμφανίζεται η ανάγκη για καινοτόμους και δημιουργικούς αλγορίθμους οι οποίοι καθορίζουν τη βέλτιστη τοποθέτηση των Hard Macros.

Ενώ αυτά τα αντικείμενα μπορούσαν αρχικά να τα διαχειριστούν αλγόριθμοι Floorplanning, ο αυξανόμενος αριθμός των Hard Macros στα σύγχρονα κυκλώματα περιορίζει την ικανότητα αυτών των αλγορίθμων να συνεχίσουν να λειτουργούν αποδοτικά. Ως αποτέλεσμα, τα Hard Macros πλέον διευθετούνται από αλγορίθμους Macro Placement, οι οποίοι λαμβάνουν υπόψην πολλαπλές μετρικές και αποσκοπούν στην υποστήριξη της τοποθέτησης των Standard Cells. Σε αυτή τη δουλειά παρουσιάζεται ένας αλγόριθμος τοποθέτησης των Macros βασισμένος στη μεθοδολογία της Πακετοποίησης (Packing), συνοδευόμενος από πειραματικά αποτελέσματα που επιδεικνύουν τόσο τις δυνατότητες προσαρμογής του σε διαφορετικές ροές, όσο και την υπεροχή του σε σχέση με την ταυτόχρονη τοποθέτηση των Macros και των Standard Cells.

Λέξεις-κλειδιά:

EDA, VLSI, Φυσική Σχεδίαση, Hard Macros, Χωρική Τοποθέτηση, Πακετοποίηση

Table of contents

Acknowledgements	ix
Abstract	xii
Περίληψη	xiii
Table of contents	xv
List of figures	xvii
List of tables	xxi
Abbreviations	xxiii
1 Introduction	1
2 Theoretical Background	5
2.1 Physical Design Background and Flow	5
2.2 Floorplanning	7
3 Macro Placement Introduction and Existing Work	9
3.1 Introduction to Macro Placement	9
3.2 Floorplanning Representations	11
3.3 Macro Placement Algorithms	14
3.3.1 Macro Placement Methodology Overview	14
3.3.2 Non-Packing-based Macro Placement Algorithms	14
3.3.3 Packing-based Macro Placement	16

4	Our Macro Placement Contribution	19
4.1	Revisiting the existing algorithms	19
4.1.1	Explanation and Refinement of the CPtree structure	19
4.1.2	Packing information of the CPtree structure	22
4.2	Filling the CPtree	23
4.3	Boundary Packing	26
4.4	Children Packing	28
4.5	Contour Packing	33
5	Macro Placement Evaluation and Flow Settings	37
5.1	Top-level Macro Placement flow	37
5.2	Results Evaluation	37
5.3	CPtree Perturbations	40
5.4	Optimization Settings	42
5.5	Top-level Placement flow	45
6	Experimental Results	47
6.1	Macro Placement Flow Experiments	47
6.2	Top-level Placement Flow Experiments	50
7	Conclusions and Future Work	55
	Bibliography	57

List of figures

1.1	Moore's Law on the increasing number of transistors on a chip per year . . .	2
1.2	The steps required to design a chip using EDA	2
2.1	The steps of the Placement Flow	5
2.2	The steps of the Placement Flow	6
2.3	Global Placement vs Legalized Placement	6
2.4	Computation of the Half-Perimeter Wire Length of a simple net. The HPWL is computed as $w + h$	7
3.1	Visualization of the difference between Standard Cells (yellow) and Macros (red)	10
3.2	Example of a placement (a) and its corresponding O-Tree (b). Picture taken from [1]	12
3.3	Example of a placement (a) and its corresponding B-Tree (b). Picture taken from [1]	12
3.4	Example of a placement (a) and the potential new node insertions in the Corner Sequence (b). Picture taken from [1]	13
3.5	A representation of two sequences in the Sequence Pair algorithm. Picture taken from [1]	13
3.6	The two-stage approach followed in the Macro Placement algorithms . . .	14
3.7	The flow of Circuit Training. Picture taken from [2]	15
3.8	The flow of AutoDMP. Picture taken from [3]	16
3.9	An example of an MP-Trees Macro Placement. Picture taken from [4] . . .	17
3.10	An example of an CPtree Macro Placement. Picture taken from [5]	17
3.11	An example of a Circular Contour construction. Picture taken from [6] . . .	17

4.1	A top-level view of the CPtree structure. Picture taken from [5]	20
4.2	An example of a CPtree represented as a graph	21
4.3	Pre-placed Macros and their position in the CPtree	22
4.4	Construction of the CPtree boundaries, based on the initial placement . . .	24
4.5	Boundary Packing and potential issues	25
4.6	Post-processing of the CPtree boundaries	26
4.7	An example of boundary packing for the bottom side	27
4.8	An example of Bounding Boxes	28
4.9	An example of contour tracing	29
4.10	An example of a packing subtree and its respective placement. Numbers indicate the order of initial position computation.	29
4.11	An example of child packing when no overlaps exist for the starting positions.	30
4.12	An example of a starting position yielding overlaps.	30
4.13	An example of a rotation fixing a starting position overlap.	31
4.14	Examples of shifting for overlap resolving	32
4.15	A CPtree and its Macro Placement after the end of the child packing	33
4.16	Convex (blue) vs non-convex (red) corners.	34
4.17	Examples of the fitting criterion for contour Macros.	34
4.18	An example of the steps followed during the Contour Packing	35
5.1	Illustration of the Wasted Area collected, as well as the optimal Placement Rectangle computed after a Macro Placement	40
5.2	An example of the Rotation perturbation failing due to the Anchor Width Constraint.	41
5.3	An example of the impact of a Switching Point change.	42
5.4	An example illustrating the criticality of Macros C, E and G.	44
5.5	A graph displaying the function used for the Simulated Annealing-like algorithm.	44
5.6	The two alternative flows which we experiment on.	45
5.7	The top-level Placement flow, when an initial Macro Placement is performed.	46
6.1	Comparison of HPWL between different flow approaches. The red colour indicates a worse result, while the green colour indicates a better result. . .	51

6.2	The discrete stages of the Placement flow for a design with Macros	51
6.3	Comparison between Global and Macro Placement for Ariane	53

List of tables

6.1	Designs experimented on	47
6.2	Increase in the evaluation metrics, for dynamic vs static probabilities	48
6.3	Increase in the evaluation metrics, for critical Macros vs random Macros . .	49
6.4	Increase in the evaluation metrics, for Greedy vs SA-like solution exploring	50
6.5	Success percentage for each perturbation type	52
6.6	Increase in metrics between Probabilistic approach and Flow1/Flow2	52
6.7	Comparison of HPWL between Global Placement and the best Macro Place- ment result	53

Abbreviations

EDA	Electronic Design Automation
VLSI	Very Large Scale Integration
MP	Macro Placement
GP	Global Placement
QP	Quadratic Placement
DEF	Design Exchange Format

Chapter 1

Introduction

Nowadays, integrated circuits, also known as chips, are increasingly prevalent in all aspects of society, powering devices that shape our everyday lives. From laptops and smartphones to medical, military or space applications, chips offer capabilities that were unimaginable when the first electronic circuit was presented. During those early years of semiconductor technology, the tasks required to form a chip were carried out manually. Things would soon change, however, as it was rendered possible to incorporate a large number of transistors on a single chip, as was formulated by Gordon Moore in 1965 (Figure 1.1). As the size of the transistor scaled down, it was imperative for the design community to tackle the issue of scaling. Soon, the demand for faster and more power-efficient devices arose, and chip development became more challenging, therefore automation was the clear solution.

Electronic Design Automation, commonly referred to as EDA, refers to the algorithms, methodologies and software used to design, simulate and verify integrated circuits. While the first problem it was used to solve was affiliated with the physical layout of the chip, nowadays every aspect of the development of chips benefits from the development of EDA, starting from high-level design and ending with fabrication. Furthermore, the conception of multiple complicated algorithms has enabled the designer to focus on the optimization of metrics such as chip area or power, while minimizing the time required to execute the automated flow.

As displayed in Figure 1.2, the EDA flow consists of multiple steps, each of which must be optimised to create a functioning and efficient chip. First, the design must be clearly specified and described in architectural and behavioral level, with the aid of a hardware description language. After that, it must be synthesized and verified at a gate level. The next step is the physical design, which includes deciding the optimal position of the physical cells, pins and

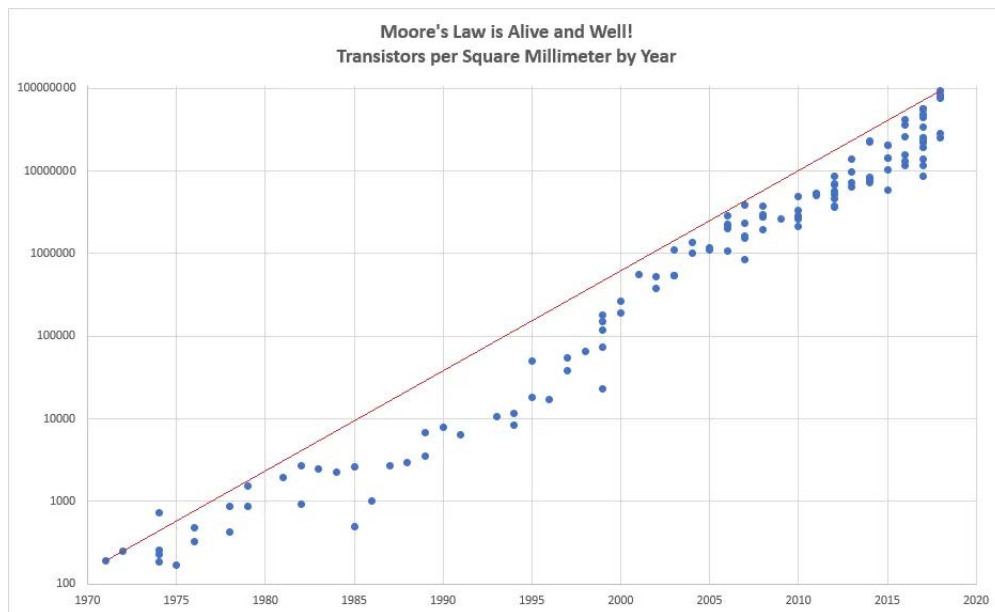


Figure 1.1: Moore's Law on the increasing number of transistors on a chip per year

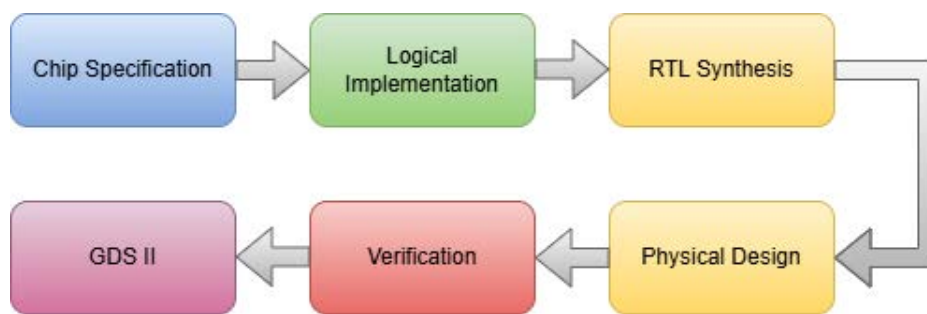


Figure 1.2: The steps required to design a chip using EDA

metal layers. Finally, the chip is verified and, ultimately, taped out.

Over the years, physical design for integrated circuits has evolved significantly, adapting to the ever-increasing complexity that was previously described. In the early days, the physical components of a chip would be arranged manually on silicon, relying on the experience of some designers. But as the need for more complex devices arose, automation was inevitably employed to detect the optimal formation of the millions of components. Meanwhile, challenges such as timing management and signal integrity would emerge due to the constant changes in technology. Therefore, the importance of high-quality automated physical design is evident.

One recent challenge that has emerged, in the context of physical design, is the treatment of particularly large objects, also known as Macros. While the number of Macros inside a chip was initially relatively moderate, allowing manual methods to be effective enough, the

aforementioned evolution of the chips over the years has caused that number to increase, creating the need for innovative algorithms that can handle hundreds of Macros.

The aim of this work is to provide a state-of-the-art, effective approach for the placement of Macros. First, the problem will be analyzed thoroughly and the current literature will be presented. Afterwards, we will present our implementation of a packing-based Macro Placement algorithm, explaining each of its steps exhaustively, as well as the proposed flows for the cell placement stage. Finally, the experimental results will be demonstrated.

Chapter 2

Theoretical Background

2.1 Physical Design Background and Flow

As explained earlier, the physical design of an integrated circuit is an essential step in the EDA flow. In figure 2.1, the separate stages are displayed. First, Floorplanning determines the fundamental characteristics of the chip layout, such as the size and aspect ratio of the core area, the position of the I/O pins and the Standard Cell rows. Then, Placement finds an optimal position for each component based on the connections between them. Clock Tree Synthesis distributes the clock signal across the chip, and finally Routing translates the connections between components into multiple metal layers.

The basic building block of a modern chip is the Standard Cell. When a circuit is described using RTL (Register Transfer Level), the components are described in an abstract manner, which is later translated to a physical formation through the EDA flow. These components, which are boolean gates, flip-flops, adders, or other logic units, represent a specific formation of transistors, which, on a metal level, fulfils the boolean logic. They are arranged as predefined blocks, with specific dimensions and characteristics, and are connected to each other with multiple metal layers, which are, on a higher level, represented with wires. These blocks are called "Standard Cells" and constitute the majority of the circuit logic on the phys-

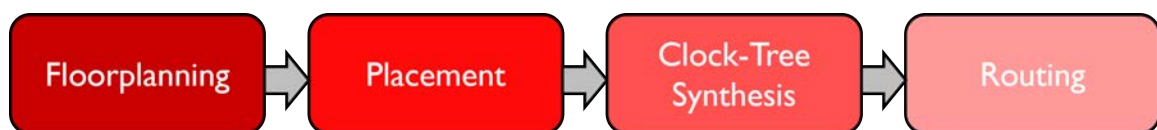


Figure 2.1: The steps of the Placement Flow

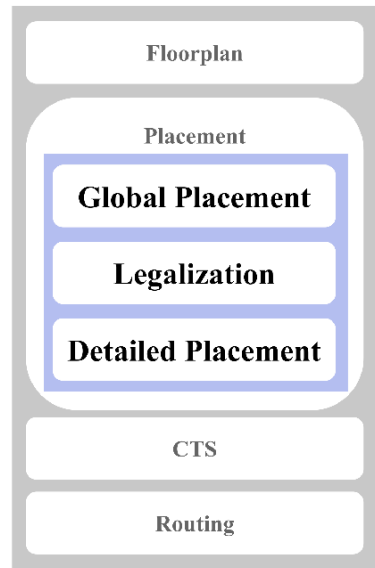


Figure 2.2: The steps of the Placement Flow

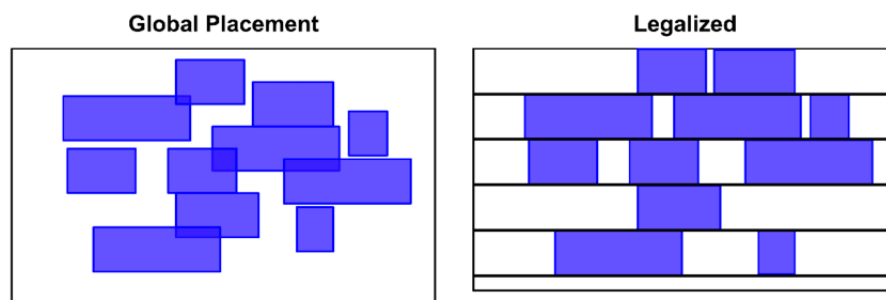


Figure 2.3: Global Placement vs Legalized Placement

ical level. A broad set of algorithms computes the optimal formation of all cells, constituting the Placement flow.

In Figure 2.2, the algorithms that encapsulate the process described as Placement are shown. First, the Global Placement algorithm aims to place all cells using arithmetical methods, so that the length of the connecting wires is minimized, while allowing cells to overlap with each other to an extent. Afterwards, the Legalization algorithm places the cells on legal positions based on the rows of the chip area, removing all overlaps in the process, as displayed in Figure 2.3. Finally, the Detailed Placement algorithm performs small moves, so that the wirelength or other criteria are optimised, while retaining an overlap-free placement.

A placement is evaluated based on multiple metrics, such as timing, power or congestion. The main criterion that signifies the quality of a placement is the total wirelength needed to connect all cells with each other. However, wires correspond to metal layers and, since routing

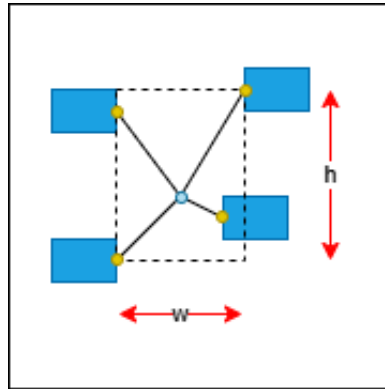


Figure 2.4: Computation of the Half-Perimeter Wire Length of a simple net. The HPWL is computed as $w + h$.

is a separate step in the placement flow, using it to calculate the wirelength at this point would be suboptimal. Therefore, an approximation is used to measure this important criterion. The most common one is the HPWL (Half-Perimeter Wire Length) which, as its name suggests, assumes that the total length of a net can be approximated to the length of half its perimeter, as shown in Figure 2.4.

2.2 Floorplanning

It is important to discuss about Floorplanning, as it is the step that initiates the placement flow. Even though it precedes Global Placement in the EDA flow, it follows it in this theoretical discussion, because it forms an efficient bridge with the main topic of this thesis. Besides, successful Floorplanning greatly improves the following placement, so there is affiliation between these two steps. In an abstract way, it serves as a preparation of the placement. Some of the key roles of Floorplanning are:

- Determination of die area aspect ratio
- Determination of sub-areas for partitions
- I/O Pin Placement
- Power Planning
- Placement of large objects

The most important of these roles, regarding this work, is the placement of large objects. It is common for a circuit to contain objects such as memory blocks, processor cores or other Intellectual Property (IP) blocks. Originally, since the circuits were smaller in size, these large objects were relatively few, so the designer could simply determine their position by hand. However, since the dimensions of modern chips are constantly increasing, this approach is becoming less and less feasible. Therefore, the requirement of developing algorithms that handle these large objects in particular has emerged.

Chapter 3

Macro Placement Introduction and Existing Work

3.1 Introduction to Macro Placement

It was discussed in the previous chapter, that the scaling of modern integrated circuits, and the resulting increase of the components of a chip, has significantly altered the way their design process is tackled. One of the aspects that requires particular handling is the placement of large objects, or in other words, the Macro Placement. Before continuing with a more detailed presentation of the problem, it is vital to clarify the definition of a Macro, and its subcategories.

A Macro refers to a functional block within a VLSI circuit, whose size is significantly larger than that of a Standard Cell. Macros usually represent major units such as memories (SRAM, DRAM), processing cores (CPU, GPU), but they can be anything that the designer decided that should not be represented with Standard Cells. The difference between Macros and Standard Cells is evident in Figure 3.1.

Macros can be categorized as Hard Macros or Soft Macros. On the one hand, Hard Macros are blocks that cannot be modified, have a fixed aspect ratio and are considered optimized. Practically, they serve as a black box for the designer. On the other hand, Soft Macros can have configurable aspect ratios and usually do not represent a finalized unit. They may even contain other Standard Cells or Soft Macros inside them. A Physical Cluster is an example of a Soft Macro.

It is crucial that attention is paid to the placement of Macros. Suboptimal Macro Place-

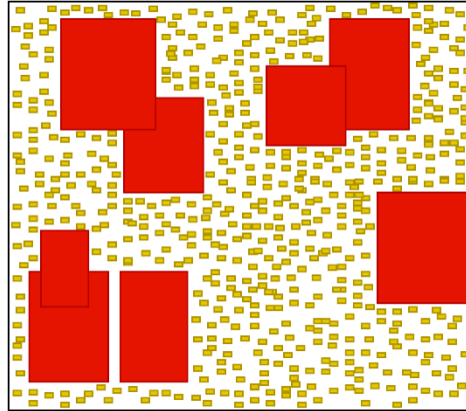


Figure 3.1: Visualization of the difference between Standard Cells (yellow) and Macros (red)

ment can expose the circuit to various issues that are caused by the existence of these large objects. In more detail, these issues include:

- **Global Placement Obstruction:** If a large block is placed at the center of the core area, it hinders the Global Placement algorithm, because it leaves less space for the small objects to be placed optimally.
- **Legalization Issues:** For Standard Cells, overlaps are not a big concern, due to their relatively small area. However, for blocks that are 100 or 1000 times the size of a Standard Cell, finding an overlap-free position is challenging. If the Global Placement algorithm places Macros with overlap, the Legalization algorithm may have trouble retaining an optimal formation.
- **Routing Issues:** If Macros are placed without considering the routability of the circuit, the routing solution may be suboptimal, since a large block is often connected to many other components.

It is possible to disregard the Macro Placement algorithms and try placing the Macros and the Standard Cells concurrently. This approach is called Mixed-size Placement and is followed by many state-of-the-art algorithm, as will be discussed. However, attempts to apply a Global Placement algorithm, without considering the existence of Macros, will usually face some of the issues that were previously described. To prevent this scenario, many Macro Placement algorithms have been developed.

3.2 Floorplanning Representations

The first algorithms that targeted large blocks were presented in the form of Floorplanning representations. There are a few differences between Floorplanning and Macro Placement, even though both steps of the Physical Design flow deal with large objects. On the one hand, Floorplanning is more related to determining the dimensions and the aspect ratio of the core area that fit all objects optimally. They usually take advantage of the Simulated Annealing algorithm [7]. This algorithm is inspired by the annealing process in metallurgy, and explores solutions based on a probabilistic scheme. Initially, both better and worse solutions are accepted, but as the iterations pass, the algorithm favours the positive moves, converging to an optimal solution. On the other hand, Macro Placement is more related to Global Placement, as the former aims to support and improve the latter. Macro Placement sometimes includes Floorplanning representations in its algorithms, often combining them with more fresh approaches, hence their continuous importance in the modern era.

It is important to note that, most Floorplanning representations implement packing of the large objects. Packing is a formation in which all Macros are physically connected, either to each other or to the chip boundaries. This approach is widely used in Macro Placement as well, since the general consensus has been that Macros should be placed near the chip boundaries, instead of towards the center. By keeping the Macros physically connected to the boundaries and to each other, it is easier to represent them using structures such as trees, and extract information about the current status of the floorplan.

O-Trees [8] represent the formation of the large objects, with reference to the chip boundaries, using a tree with an arbitrary number of branches. Two trees are required, one to model the horizontal and one to model the vertical formation of the floorplan. Each child of the tree root represents an object whose horizontal or vertical distance from the respective boundary (left or bottom) is equal to zero, and the successive children represent objects whose horizontal or vertical distance from their parent is zero. An example is shown in Figure 3.2.

B*-Trees [9], like O-Trees, represent the floorplan with a tree. However, they take advantage of the properties of the binary tree to model positional relationship between the modules. The root of the tree is the Macro that sits on the bottom-left corner, its left child is the Macro on its left and its right child is the Macro on its right. This logic is carried on to all children, and the final binary tree includes all nodes, as shown in Figure 3.3

In contrast to the previous two approaches, Corner Sequence [10] does not use a tree, but

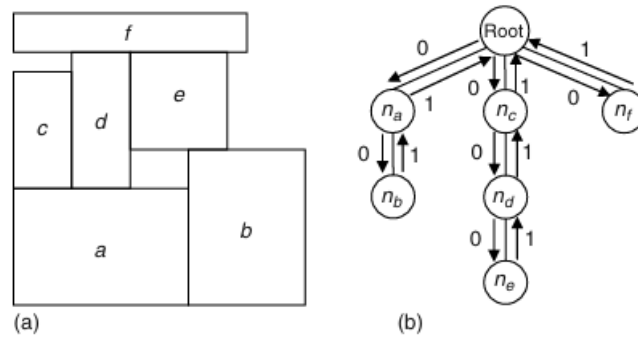


Figure 3.2: Example of a placement (a) and its corresponding O-Tree (b). Picture taken from [1]

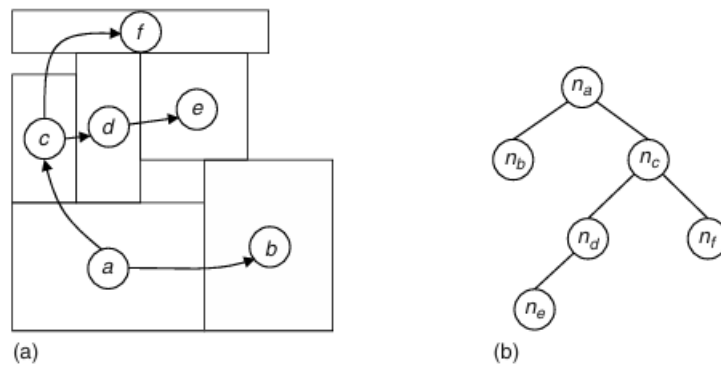


Figure 3.3: Example of a placement (a) and its corresponding B-Tree (b). Picture taken from [1]

rather a list. Each element of this list consists of a node and two already defined nodes, which form the corner on which the Macro is placed. This is continued for all Macros, forming a final list with as many elements as the number of Macros. The left and bottom boundaries are used as dummy nodes. An example is displayed in Figure 3.4.

The Sequence Pair [11] encodes the relative positioning between two modules using two sequences, one for the horizontal and one for the vertical relation between the modules. From these sequences, horizontal and vertical constraints are extracted, and the final layout is computed. A physical representation of the two sequences is shown in Figure 3.5.

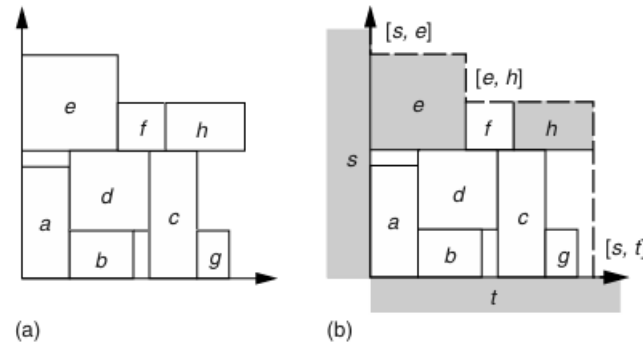


Figure 3.4: Example of a placement (a) and the potential new node insertions in the Corner Sequence (b). Picture taken from [1]

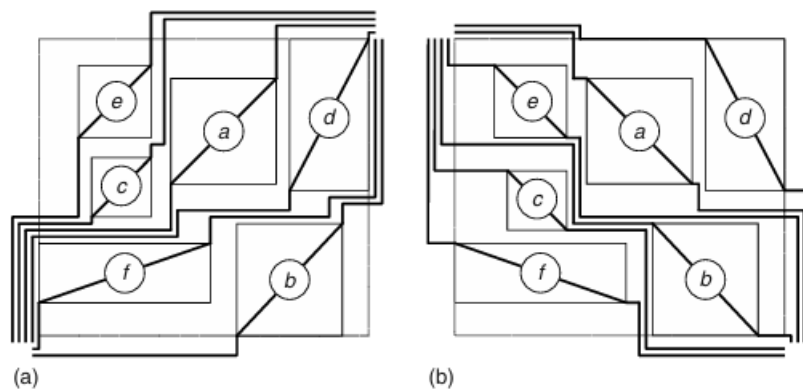


Figure 3.5: A representation of two sequences in the Sequence Pair algorithm. Picture taken from [1]

3.3 Macro Placement Algorithms

3.3.1 Macro Placement Methodology Overview

A modern chip consists of Macros as well as Standard Cells, and the goal of the designer is to place both optimally. In order to achieve this, the naive approach of applying a standard Global Placement algorithm can be followed. However, as was already mentioned, there are potential issues that may arise. Instead, there is a large pool of available Macro Placement algorithms that can be utilized.

The majority of modern Macro Placement algorithms follow the two-stage approach, which is shown in 3.6. An initial placement, also called the "Placement Prototype" is given as input to the Macro Placement algorithm. First, Macro Placement is performed, taking the initial placement information into consideration, and then, after the Macros are fixed into their final positions, Global Placement places the Standard Cells optimally in the available area.



Figure 3.6: The two-stage approach followed in the Macro Placement algorithms

Multiple Macro Placement categorizations have been mentioned in the literature, but the one that interests us for the purposes of this work is between the Packing and the Non-Packing Algorithms. The first category includes algorithms that rely on previously-mentioned Floor-planning representations, or expansions of them, in order to place Macros next to each other, while the second category includes algorithms that can be characterized with various terms, such as Analytical Macro Placement or Reinforcement Learning-based Macro Placement.

3.3.2 Non-Packing-based Macro Placement Algorithms

In the recent years, there has been a plethora of significant and innovative algorithms associated with Macro Placement. The aforementioned innovation explains why it is challenging to categorize them. Each algorithm unearths a new piece of information and novelty, and all of them play a vital role in further breakthroughs.

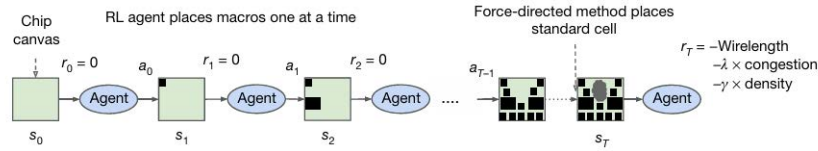


Figure 3.7: The flow of Circuit Training. Picture taken from [2]

To begin with, Google has presented a state-of-the-art solution to the problem of Macro Placement in the form of Circuit Training [2]. Circuit Training approaches Macro Placement as a Reinforcement Learning problem. Thus, a training process is applied, so that the decision of the optimal Macro positions gradually improves and learns from past placements. As shown in Figure 3.7, the agent places Macros one at a time, and afterwards, the Standard Cells are placed using a force-directed algorithm. When each Standard Cell placement ends, the reward is calculated as a function of wirelength, congestion and density, and the next iteration begins until convergence. This approach, while theoretically remarkable, caused controversy due to the lack of clarification [12], and therefore still needs to prove its worth.

Next, RTL-MP [13] aims to place Macros the way a human expert would. It takes advantage on the logical hierarchy and the dataflow information defined by RTL designers, as well as other exploitable information such as critical timing paths, while it heavily involves clustering. The flow of the algorithm consists of Netlist Clustering, which differentiates Macro clusters from Standard Cell clusters, Cluster Shaping, Cluster Placement and Macro Placement. The final step utilizes Sequence Pair and Simulated Annealing to produce an ideal formation of Macros inside each cluster. This algorithm was expanded as Hier-RTLMP [14], incorporating multi-level clustering and more detailed cluster shaping.

An interesting mixed-size approach is Auto-DMP [3]. This algorithm is based on DREAM-Place [15], a GPU-accelerated analytical placer. Macros and Standard Cells are placed concurrently, and a vast parameter space consisting of 16 parameters is explored. To evaluate the solutions, a two-level approach is employed. First, Multi-objective Bayesian Optimization evaluates solution based on wirelength, density and congestion, and then the best candidates are evaluated inside an EDA tool, based on wirelength, timing and power. The flow is displayed in Figure 3.8.

Another state-of-the-art algorithm, IncreMacro [16], could be characterized as a Macro Refinement algorithm, as it uses an initial placement generated by an analytical placer, and optimizes it. It emphasizes on moving Macros towards the periphery, and additionally on

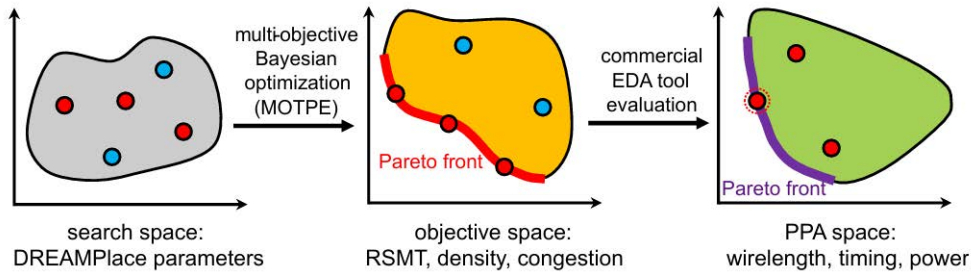


Figure 3.8: The flow of AutoDMP. Picture taken from [3]

preserving the positional relationship between Macros. Furthermore, it takes advantage of methods such as gradient shifting and graph-based linear programming.

Last but not least, an algorithm based on Black-Box Optimization [17] separates the canvas into grids, also referred to as partitions, and refines the existing placement by moving each Macro to the grid that causes the smallest HPWL increase.

3.3.3 Packing-based Macro Placement

This section features more traditional solutions than the previous one, as packing-based algorithms originate from the already mentioned Floorplanning representations. In fact, the following algorithms are all based on some Floorplanning structure, but they manage to enhance it in order to handle the increasing demands of modern chip design. Usually, Simulated Annealing boosts their flexibility and enables them to explore a vast solution space.

MP-trees [4] is a packing structure that places Macros on the four corners of the chip and one next to another. It is based on the previously mentioned B*-Trees representation, expanding it to handle four corners instead of only one. Therefore, four packing trees are required, and each one translates the tree representation to a placement representation, as shown in Figure 3.9.

Another structure that relies on the tree structure is the CPtree-based Macro Placement [5]. An expansion of MP-Trees, the CPtree represents the entire chip floorplan into a single, circular structure that also considers pre-placed Macros or blockages and includes them in this structure, as shown in Figure 3.10.

Finally, Circular Contour [6] operates on a circular list, which represents the available corners of the chip layout. It is based on the Corner Sequence, expanding it to apply to every available corner in the entire floorplan. Geometrical methods are deployed to detect viable corners on the existing contour, which is updated constantly as displayed in Figure 3.11. The

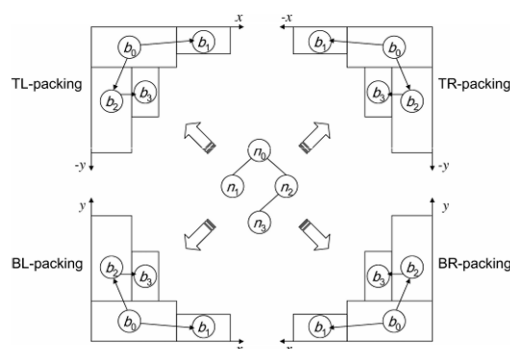


Figure 3.9: An example of an MP-Trees Macro Placement. Picture taken from [4]

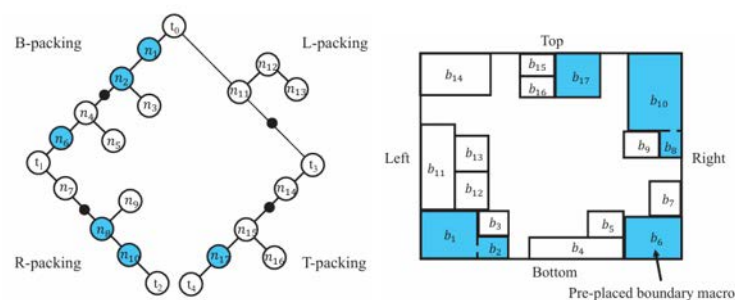


Figure 3.10: An example of an CPTree Macro Placement. Picture taken from [5]

solution is evaluated based on the wasted area or dead space, which is the inaccessible area left after a Macro shifting, as well as the shape of the final contour.

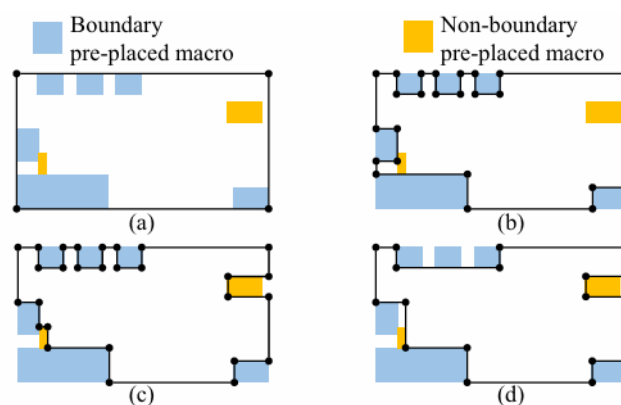


Figure 3.11: An example of a Circular Contour construction. Picture taken from [6]

Chapter 4

Our Macro Placement Contribution

4.1 Revisiting the existing algorithms

The algorithm that was developed for this work combines two of the packing algorithm that were presented in the previous chapter. The first algorithm is the CPtree-based Macro Placement [5] and the second algorithm is the Circular-Contour-based Macro Placement [6]. For simplicity, these two algorithms will be referred to with the names of their corresponding structures.

They were deemed suitable algorithms to build on, because the experiments performed on them were for a relatively small number of Macros, so the idea of combining and expanding them to be able to handle hundreds of Macros is a strong motivation for this work. Furthermore, these two algorithms complement each other. While CPtree focuses on filling the boundaries and placing the Macros based on them, Circular Contour does not use the concept of the chip boundaries at all, and focuses on placing the Macros on the available corners.

The CPtree structure, contrary to the Circular Contour, is a complicated structure. Therefore, it is essential to analyse in detail, so that its purpose and its details are elucidated.

4.1.1 Explanation and Refinement of the CPtree structure

The basic structure utilized is the CPtree, introduced in [5]. In this structure, the Macro Placement is represented as a binary tree. As shown in Figure 4.1, in the original work, there are four discrete sections of the tree that represent the four boundaries, and they are joined together with the help of some auxiliary nodes, called “terminal nodes”. It is clear that these

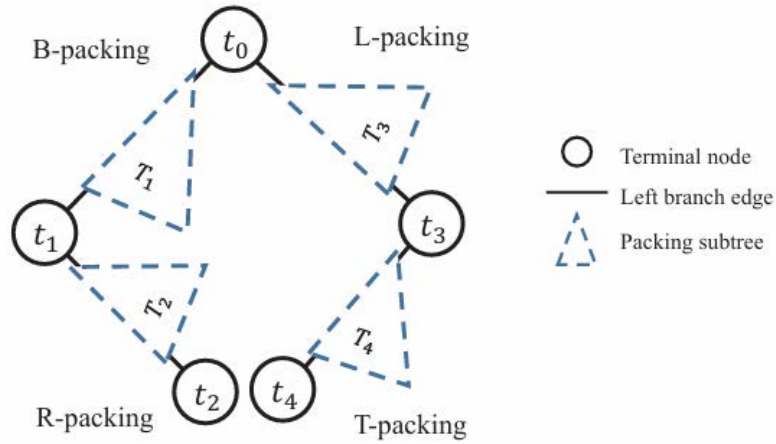


Figure 4.1: A top-level view of the CPtree structure. Picture taken from [5]

nodes represent the four corners of the core area, and that each terminal node matches the respective corner that the two respective boundary branches have in common.

Before proceeding to the finer details of the structure, it is essential to clarify a few points which may seem hazy at first glance. In Figure 4.1 there are five terminal nodes depicted for four corners, and all boundary branches are depicted as left branches, even though it is not stated clearly which node is the root of the binary tree. These ambiguities are clarified in this work, using a circular graph with four terminal nodes representing the corners, and four sections representing the boundaries that are all connected with the aid of the terminal nodes, as shown in Figure 4.2. Hence, our structure is, theoretically, a graph instead of a binary tree, due to the circular nature of it. However, we will take advantage of the tree terminology to describe our implementation, for practical reasons. So, the root of the tree will be the top left corner, the boundary branches will also be referred to as “left branches”, and every Macro that belongs to a left branch will be a “boundary child” or a “left child” (even though other nodes can also be left children, as we will shortly discuss). The union of all the boundary packing subtrees and the terminal nodes will be called the “left branch”.

An essential part of the CPtree structure are the packing anchors. These form a superset of the four terminal nodes that represent the corners, and also include the pre-placed Macros or rectangular blockages located on the chip boundaries, which are also represented as nodes in the left branch. Each of these nodes is the root of a packing subtree that starts from one packing anchor and ends at another, and every placeable node must have exactly one packing anchor. In other words, the packing anchor is the element that separates one packing subtree

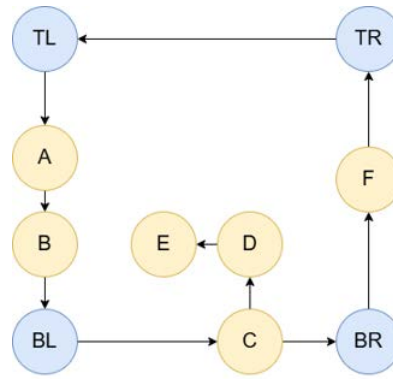


Figure 4.2: An example of a CPtree represented as a graph

from the other. It is important to note that, whether an anchor corresponds to a pre-placed Macro or a corner, it will always be affiliated with the boundary of the chip area. In the example of Figure 4.2, the four packing anchors are represented differently from the nodes representing the placeable Macros, while in Figure 4.3, the addition of pre-placed Macros is shown.

Of course, the structure does not consist purely of packing anchors and boundary nodes. With the exception of the four corner nodes, each boundary node can have a right child, in addition to the left child that they must have by definition, in order for the circular structure to not come across a dead end. This right child, which represents a Macro that will be placed, can have left and right children, and so on, just like every binary tree. All these left and right children, which are referred to as “non-boundary children”, are a part of their respective packing subtree, and their packing anchor is the same as every other node in the subtree.

My algorithm can also handle non-boundary pre-placed Macros, albeit differently. If such a Macro is “close enough” to one of the four chip boundaries, we assume that no object can fit between the Macro and the boundary, and so we consider it a boundary pre-placed Macro. As shown in Figure 4.3, the Macro is artificially stretched to be considered a boundary Macro. If the Macro is not close enough, it is considered a “non-boundary pre-placed Macro” and is treated as an obstacle which we have to avoid. In other words, we do not purposefully pack Macros next to a non-boundary pre-placed Macro, but we pack them so that they do not overlap with these kinds of Macros. Aside from this interaction, however, they are completely disconnected from the rest of the CPtree structure.

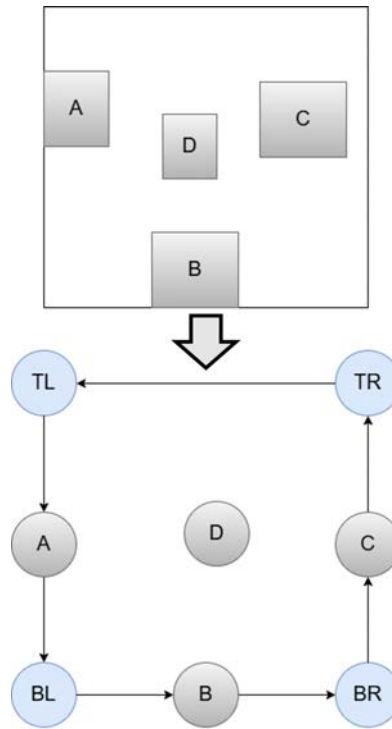


Figure 4.3: Pre-placed Macros and their position in the CPtree

4.1.2 Packing information of the CPtree structure

In [5], it is stated that each packing anchor contains the following information:

1. Anchor Distance
2. Number of Boundary Children
3. Switching Point
4. Interval Space
5. Anchor Width Constraint

We will describe each of these terms in detail. The Anchor Distance is the physical distance between an anchor and the very next anchor in the left branch. Since all anchors are, as was previously clarified, affiliated with the chip boundary, and the four chip corners are a subset of them, every pair of consecutive anchors will share the same boundary, so the distance will correspond to either the horizontal or the vertical distance between the two anchors.

The Number of Boundary Children is self explanatory and corresponds to the number of nodes between the respective anchor and the following one.

The Switching Point is a critically important element of the structure. It is not a node, and has no physical substance or coordinates, but rather a point in the scope of the tree. It splits the packing subtree in two parts. Nodes that belong to the part on the side of the subtree anchor will be packed backwards, or in other words, towards that anchor, while nodes that belong to the other side will be packed away from the packing anchor. This concept applies mainly to the boundary Macros, but it institutes a set of packing rules that must be followed by the child Macros as well. The packing details will be analyzed in more detail in another section.

The Interval Space describes the boundary segment that, after the placement of all boundary nodes, remains blank. It is, in theory, computed by calculating the sum of the width or height of all the Macros of one boundary branch, and subtracting it from the Anchor Distance. However, the boundary branches inevitably interact with each other physically, so this formula may not be used in practice.

The Anchor Width Constraint is, as its name suggests, a constraint, specifying that the Interval Space must be greater or equal than zero, to prevent boundary overlaps or out-of-bounds Macros.

4.2 Filling the CPtree

The construction of the CPtree is a crucial part of the algorithm, since it needs to represent the desired Macro Placement as successfully as possible. It is one of the aspects that were covered inadequately in the original work, therefore a need for an original method emerges. In contrast with the packing of its contents, the tree building is carried out only once. Any optimization that may happen afterwards only alters the tree partially, swapping or moving one or two nodes at a time.

At first, we need to specify the definition of a Macro, in order to select them out of all the cells that appear in the design. Based on the experimentation that was performed, we set the minimum number of rows that a Macro must be taller than or equal to, to four rows. This excludes, aside from the single-row Standard Cells, all the multiple-row-height cells whose height is equal to two or three rows, a height that leads to the cell being considered more of a Standard Cell and less of a Macro. We decide that the width of a cell does not affect its classification, because it is extremely rare for a Standard Cell to be wider than the height of

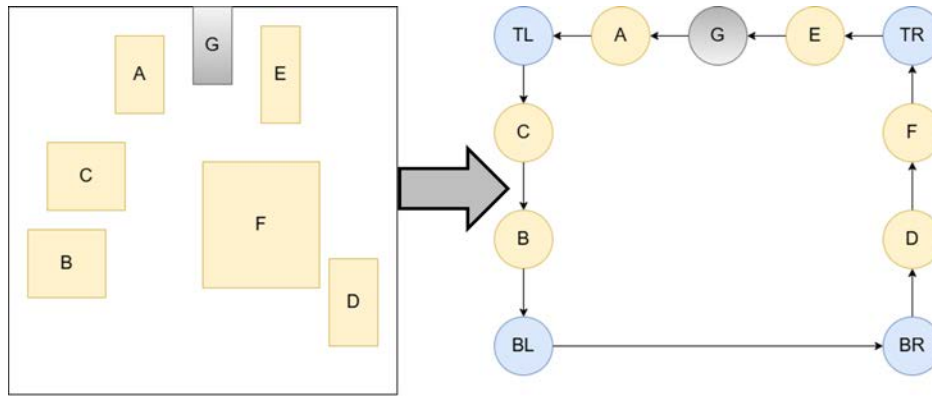


Figure 4.4: Construction of the CPTree boundaries, based on the initial placement

four rows.

The next step is to assign the Macros to the correct boundary branch. For this process, we take advantage of the information provided by the initial placement. Each Macro is assigned to the boundary that corresponds to its closest chip side, based on its coordinates. In case of a tie, we favour the left and right boundaries over the top and bottom. We also sort the Macros in each list in order of ascending distance from the respective chip side. There is an exception when it comes to large Macros, i.e. Macros whose width or height is twice as large as the average, or Macros whose area is at least 1.5 times larger than the average. These Macros are artificially moved closer to their respective boundary, the reason being that such a large object would hinder the packing procedure if it was packed as a child Macro.

Afterwards, we insert all the boundary nodes in the left branch of the CPTree structure, beginning with the left and right boundary nodes and obeying the order of the previously sorted lists. Once again, the position of each Macro in the tree depends on the coordinates produced by the initial placement. The goal is for the nodes belonging to a boundary branch to be physically arranged in the same order as the initial placement, as displayed in 4.4. We explored multiple approaches as to whether or not the boundaries should be filled as much as possible. We observed that, if we only added Macros that were close enough to the boundary, for example having a boundary distance less than 20% of the chip width or height, the packing was inefficient, due to the Macros forming isolated sections near the corners. Therefore, the current approach is to fill the boundaries as much as the anchor width constraint allows.

After the left and right boundaries are concluded, before continuing with the top and bottom boundaries, we consider potential issues near the corners of the chip. If the top and bottom packing is carried out independently from the rest of the design, the eventual packed

placement would look similar to 4.5 α' . However, the horizontal and vertical sides inevitably affect each other, so this approach would yield overlaps, as shown in 4.5 β' . This causes a problem when the boundaries are almost full, because there would be no way to solve the overlaps, even with shifting. To resolve this conflict, we reduce the interval space of the top and bottom anchors in a pessimistic way, depending on the max left and right widths of the boundary branches, forcing the packing of boundary branch to look like 4.5 γ' . It is important to be reminded that the packing is performed after the whole tree is filled, so this step cannot be dealt with dynamically. The potential gaps shown in 4.5 γ' will be fixed during the packing phase.

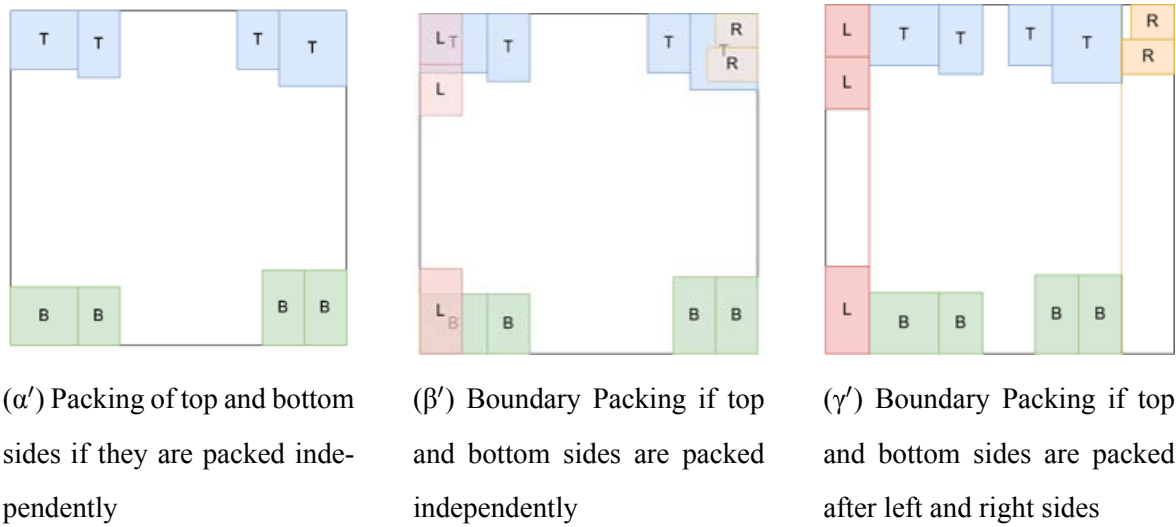


Figure 4.5: Boundary Packing and potential issues

The tree filling is completed with the list of the child Macros, which is constituted by all Macros that could not fit into the boundary. Each remaining Macro must be matched to its optimal parent. We decide to rely to the initial placement information once more, therefore considering the ideal parent of each Macro to be its closest neighbor based on the original coordinates, out of all the nodes that are already positioned in the CPtree and have at least one vacant child spot. So, if either the left child or the right child of the nearest neighbor of a Macro is empty, that neighbor is the new parent of a Macro. The only exception that can occur is attributed to the constraint that there cannot be more than two right children one after another in a right branch, in which case the Macro must be assigned as the left child, if that position is vacant.

At this point, all nodes have been added in the CPtree. The only information that has not yet been filled is the switching point position of each packing anchor, which determines the

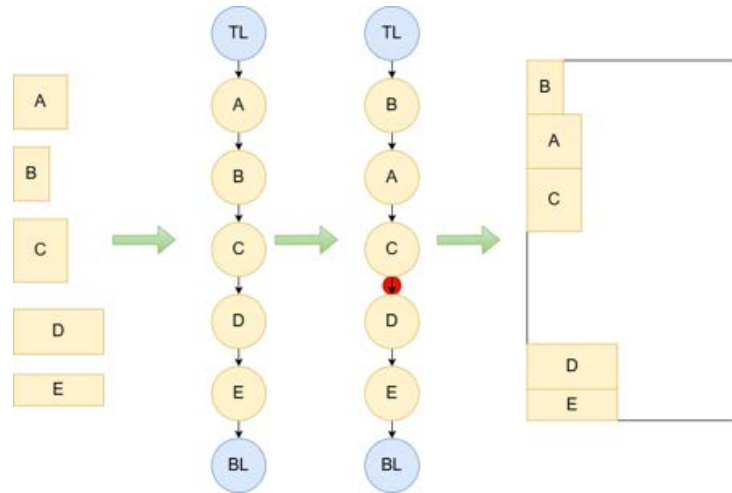


Figure 4.6: Post-processing of the CPTree boundaries

packing direction of each node. The initial approach was to insert it in the middle of each boundary subtree. However, a new criterion was implemented, to place the switching point more sensibly, considering the boundary irregularities. We perform a slight postprocessing of the nodes of each boundary subtree, by swapping nodes in order to group those with equal width or height together. We limit this postprocessing routine to swaps between Macros with no more than two nodes separating them, preserving the original idea based on which the tree was constructed. The switching point is then placed in the position with the highest irregularity, as the gap that will be caused during the packing phase will reduce that irregularity. An example is displayed in Figure 4.6.

4.3 Boundary Packing

After the tree is constructed in its entirety, the calculation of the coordinates of each Macro can begin, starting with the boundary branches. This step, compared to the placement of the child Macros, is relatively simple, since it is guaranteed that all Macros will fit in the boundaries, so algorithm 1, which was proposed in [5] can be used. The included algorithm refers to left-side Macros, but the logic is identical for the other three sides. It is important to note that terminal nodes are considered to have zero area.

For each of the four boundaries, either the x or the y coordinate is independent from the rest of the placement and only depends on the boundary. For example, the x coordinate of the Macros that sit on the left side is always equal to the start of the core area. Therefore, we only have one coordinate to calculate, which depends on the interval space of the boundary

Algorithm 1 Calculate Coordinates for List of Boundary Nodes

```

1: Input: Packing anchor  $p$ , list of nodes starting from  $p.next$ 
2: Output:  $x$  and  $y$  coordinates for each node
3:  $n \leftarrow p.next$ ;
4: while  $n \neq anchor$  do
5:    $n.x \leftarrow p.x$ ;
6:   if  $n$  is after switching point then
7:      $n.y \leftarrow n.parent.y + n.parent.h + p.intervalspace$ ;
8:   else
9:      $n.y \leftarrow n.parent.y + n.parent.h$ ;
10:  end if
11:   $n \leftarrow n.next$ ;
12: end while

```

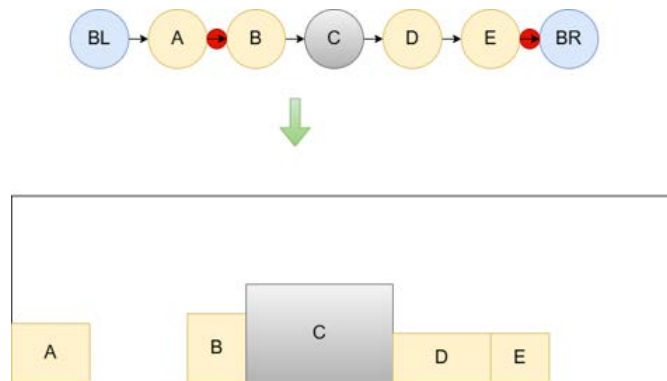


Figure 4.7: An example of boundary packing for the bottom side

packing subtree, the position of the switching point, and recursively, on the position of the other Macros. In case of pre-placed Macros, each packing subtree is packed regardless of the other anchors. An example of a boundary packing for the bottom side can be observed in Figure 4.7.

As was previously mentioned, the interval spaces of the top and bottom packing subtrees have been reduced depending on the maximum left or right boundary width, following a pessimistic approach. During the boundary packing, this interval space value is used. However, after all four sides have been packed, the top and bottom boundary Macros are shifted towards the boundary if possible, to make sure that no boundary space is wasted, except for the gap caused by the switching point.

This boundary shifting is executed using the first overlap check of the algorithm. An overlap check verifies that a Macro does not overlap with any of the placed Macro of the design. To speed up this check, we store and consistently update the anchor bounding box of each packing anchor and the boundary bounding box of each boundary Macro, as shown in Figure 4.8. It is important to note that non-boundary pre-placed Macros are considered in every overlap check, even though they are not added in the CPtree.

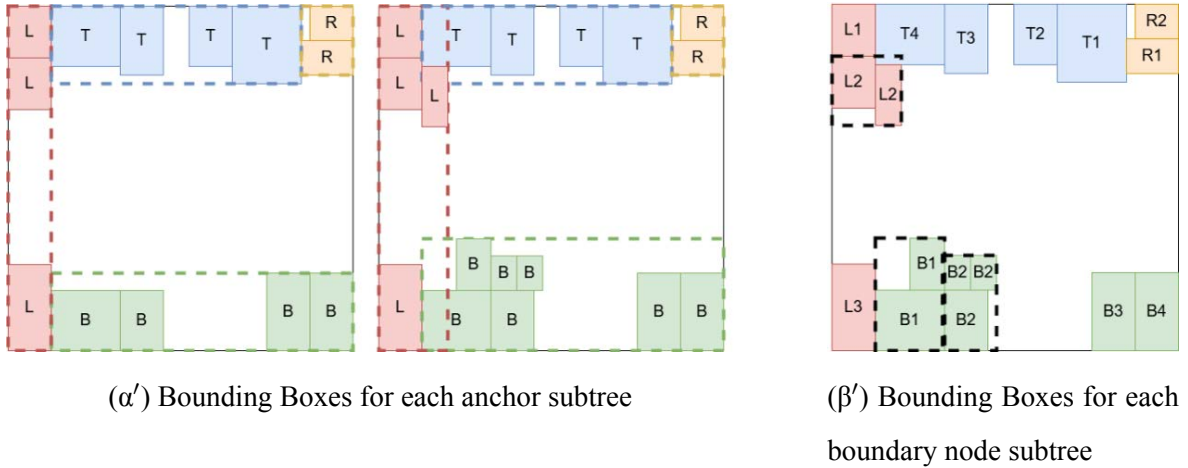


Figure 4.8: An example of Bounding Boxes

The concluding operation, regarding the boundary packing, is the formation of the contour, which will be used excessively in the next steps. Algorithmically, this operation is straightforward, as we simply trace a line across all placed Macros. We deviate from this rule only in case of a narrow opening, whose width is smaller than the width or height of the smallest Macro, meaning that no Macro can benefit from this contour pattern. An example is displayed in Figure 4.9.

4.4 Children Packing

In contrast to the boundary packing, it is unknown how many or which nodes will be successfully packed during the children packing phase. The strategy followed revolves around finalizing the position of all Macros that do not yield challenging overlaps. These Macros are packed using the original algorithm, when they are allowed to by the existing floorplan. In case of minor overlaps, we resort to shifting the current Macro.

The nodes are visited in Depth-first, pre-order traversal, skipping the boundary branches. The reason is that we want the parent of a node to be placed before calculating the location of

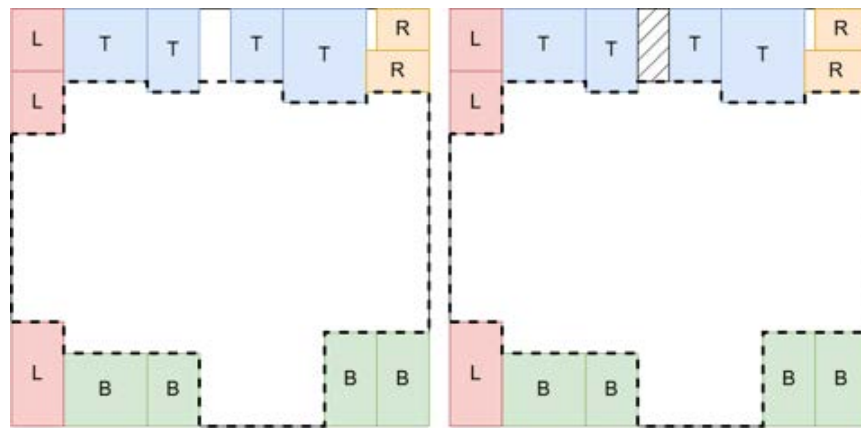


Figure 4.9: An example of contour tracing

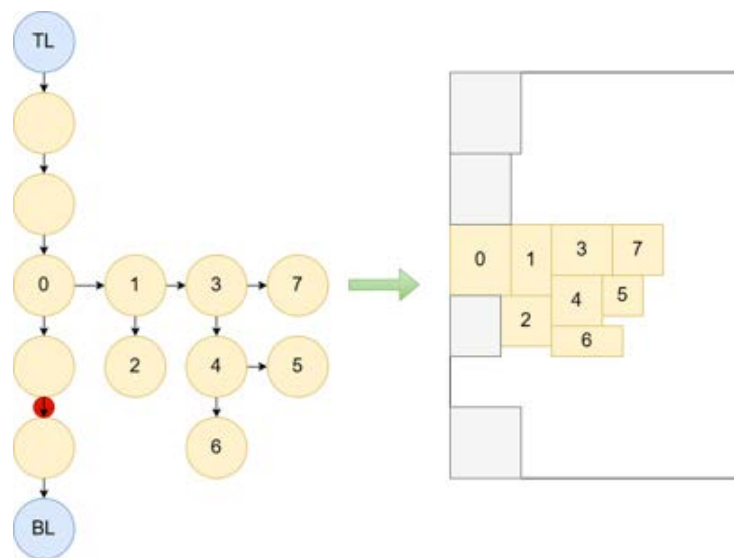


Figure 4.10: An example of a packing subtree and its respective placement. Numbers indicate the order of initial position computation.

his children. So, the pre-order approach applies mainly to the children subtrees. An example is displayed in Figure 4.10

A starting position is computed for each node, based on algorithm 2. An example of a child packing when all starting positions are overlap-free is shown in Figure 4.11 .Although this algorithm is fairly uncomplicated, it is largely inefficient for a large number of Macros if the objective is to exterminate all overlaps. Therefore, we take advantage of its simplicity to generate a starting position, but we apply a different method for determining the final coordinates. Specifically, using the already mentioned bounding boxes for the anchor and boundary subtrees, we identify the overlap for the current child Macro. If the initial position of not clear of overlaps, the Macro is temporarily rotated, and we repeat the check. If either

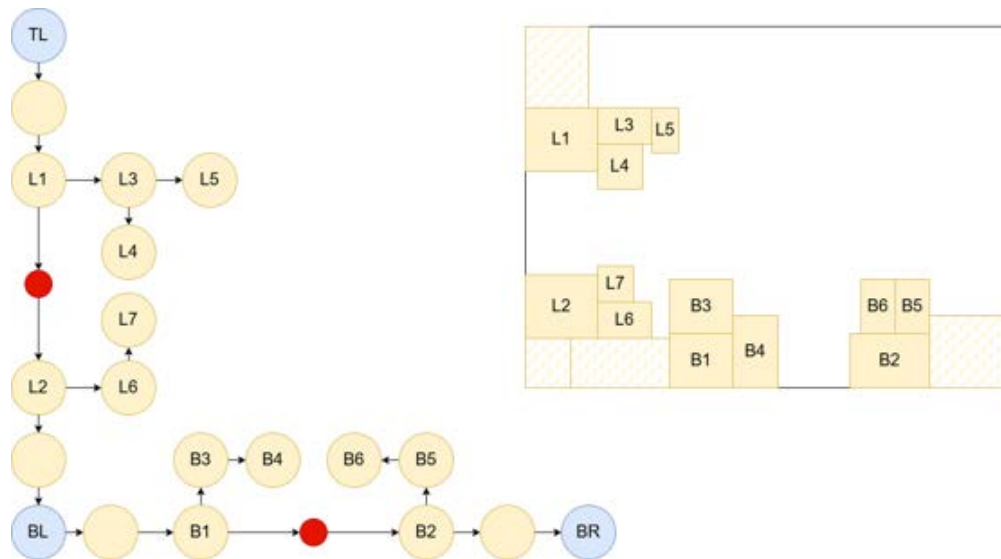


Figure 4.11: An example of child packing when no overlaps exist for the starting positions.

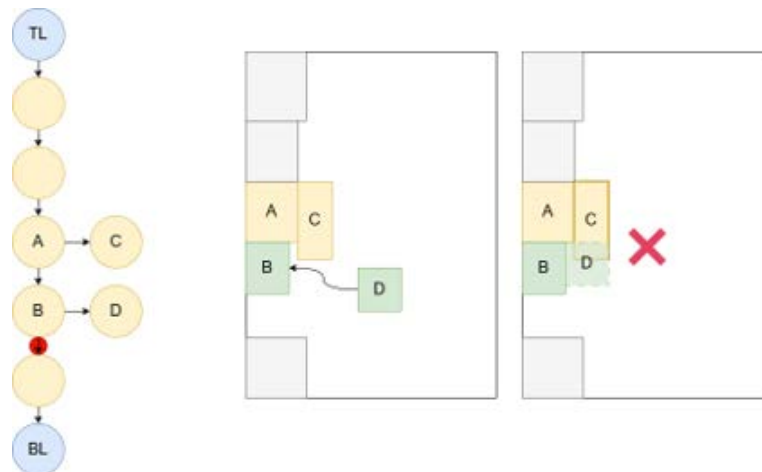


Figure 4.12: An example of a starting position yielding overlaps.

the original or the rotated Macro is found to be clear of overlaps after the initial position computation, we proceed using the respective orientation. Otherwise, we collect the overlaps for the original orientation and attempt to clear them. On condition that this process fails, we repeat for the rotated Macro, and if the result is still unsuccessful, this Macro is deemed unable to be packed in this step.

In Figure 4.12, a starting position that yields overlap is displayed, while in Figure 4.13, an overlap is overcome after rotating.

If there is at least one overlapping Macro for the respective orientation, we attempt to clear all overlaps, using different methods depending on whether the Macro is a left or a right child. In case of a right child, we implement a geometrical approach, shifting the Macro according

Algorithm 2 Calculate Coordinates for a non-boundary node

```

1: Input: Non-boundary node  $n$ 
2: Output:  $x$  and  $y$  coordinates for node  $n$ 
3:  $n \leftarrow p.next$ ;
4: while  $n \neq anchor$  do
5:   if  $n$  is a left child then
6:      $n.x \leftarrow n.parent.x + n.parent.w$ ;
7:     if  $n$  packing direction is backward then
8:        $n.y \leftarrow n.parent.y$ ;
9:     else
10:       $n.y \leftarrow n.parent.y + (n.parent.h - n.h)$ ;
11:    end if
12:  else
13:     $n.x \leftarrow n.parent.x$ ;
14:    if  $n$  packing direction is backward then
15:       $n.y \leftarrow n.parent.y + n.parent.h$ ;
16:    else
17:       $n.y \leftarrow n.parent.y - n.h$ ;
18:    end if
19:  end if
20: end while

```

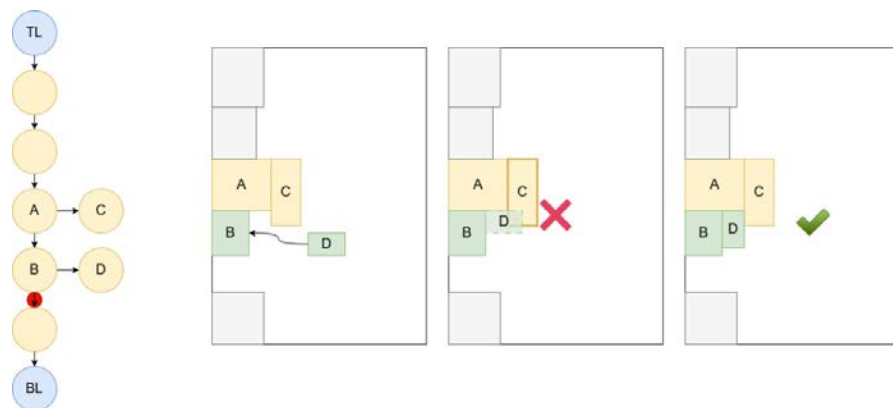
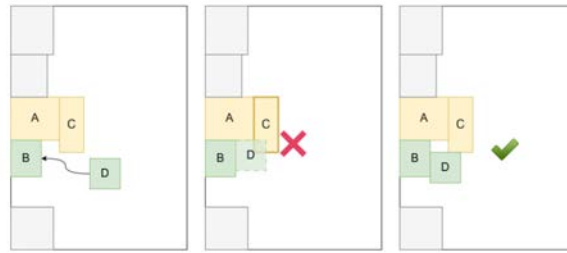
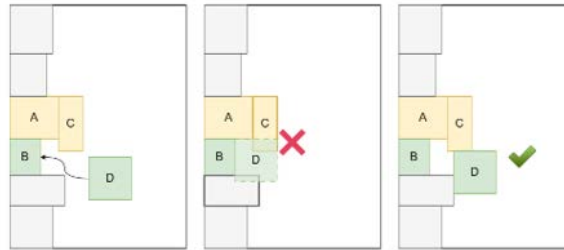


Figure 4.13: An example of a rotation fixing a starting position overlap.

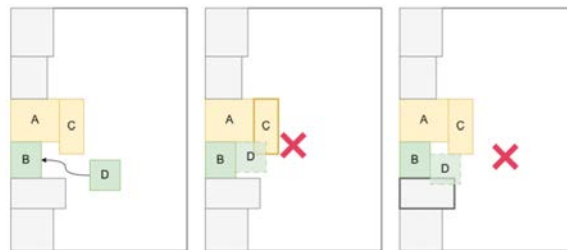
to the rectangles with which it already overlaps, as displayed in Figure 4.14 α' . When this procedure is finalised, we check for existing overlaps again, since this new position has not been explored. If new overlaps emerge, or if the current position is partially outside the core area, we discard this solution. In Figure 4.14, one unsuccessful and two successful shifting attempts are displayed.



(α') An example of a successful shifting



(β') An example of a successful shifting



(γ') An example of an unsuccessful shifting

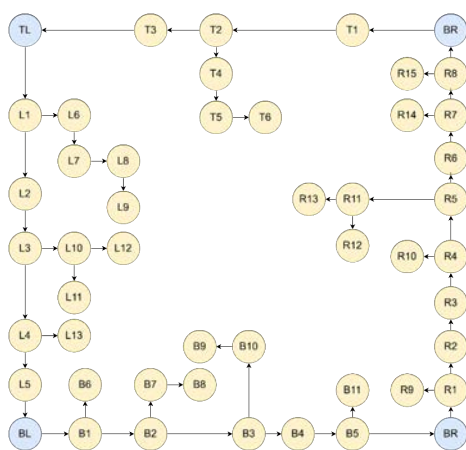
Figure 4.14: Examples of shifting for overlap resolving

In case of a left child, we obey the idea that left packing is parallel to the corresponding boundaries, so we attempt to place the Macro directly above, below, left or right of its parent, depending on the boundary. However, if we follow the original algorithm, huge stacks of Macros that affect the rest of the placement negatively may occur. Therefore, we attempt to find a corner near the original position to place the left child on, so that no overlaps occur. If no such corner is found, this solution is discarded.

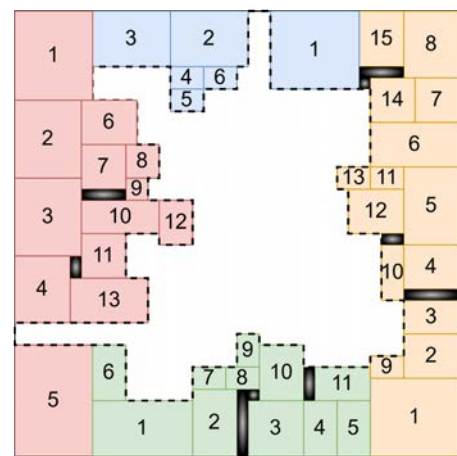
If the child is packed with success, the contour and bounding boxes are updated and we

proceed to the next Macro. If the process failed, we add the Macro to a list, called the “contour pool”, which contains all Macros whose position could not be determined in the child packing phase. There may exist nodes whose parents could not be packed. In this occasion, their children are also deemed unable to be packed during this step, and are also added in the contour pool. This explains why the Circular Contour algorithm is inseparable and equally important to the CPtree algorithm when it comes to our combination.

In Figure 4.15 α' , an example of a complete CPtree is shown, while in Figure 4.15 β' , the corresponding Macro Placement is displayed, along with the traced contour and the wasted area collected.



(α') An example of a complete CPtree



(β') The corresponding Macro Placement

Figure 4.15: A CPtree and its Macro Placement after the end of the child packing

4.5 Contour Packing

This step aims to complete the packing of child nodes by finding optimal corners for the remaining Macros. A corner can be classified as either convex or non-convex, as shown in Figure 4.16. Macros can only be placed on convex corners. The set of available corners is formed from the existing contour, excluding the non-convex corners, and consistently updated after any Macro position has been finalised.

Corners can also be categorized based on their orientation, as top-left, bottom-left, top-right and bottom-right. For each node, only the corners that match its original side are considered. For example, if a node is a member of the top boundary branch, it can only be placed

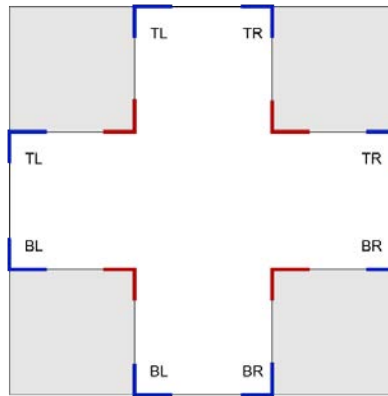


Figure 4.16: Convex (blue) vs non-convex (red) corners.

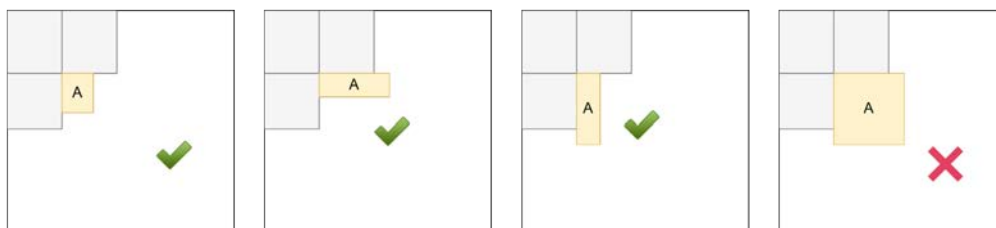


Figure 4.17: Examples of the fitting criterion for contour Macros.

on a top-left or top-right corner. For each node, all convex corners that match its side are considered. The two criteria used to determine the optimal corner are:

- Whether a Macro “fits” in the corner.
- The distance from the starting position, computed during the child packing phase.

A Macro “fits” in a corner when either its width or its height does not exceed the length of the respective contour side. An example is shown in Figure 4.17. If a Macro fits in multiple corners, the one that produces the least distance is preferred. Alternatively, if a Macro does not “fit” in any corner, only the distance is measured. But if there is at least one available corner in which the Macro “fits”, it is impossible for it to be placed on a non-fitting corner. This criterion is implemented to reduce large contour anomalies, especially for designs with varying sizes of Macros, while the distance criterion helps to minimize the divergence from the originally intended packing.

In Figure 4.18, we can observe the various candidate corners for a Macro. Rotating a Macro can unlock new corners. Ultimately, since there are multiple corners in which the Macro “fits”, the corner closest to the starting position is chosen, as shown in Figure 4.18γ’.

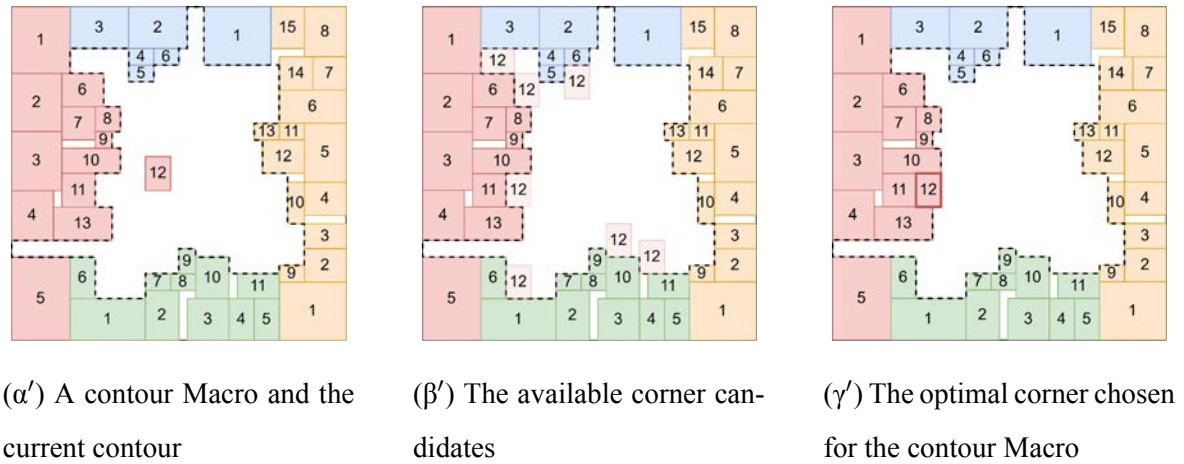


Figure 4.18: An example of the steps followed during the Contour Packing

It is essential to address the possibility of a lack of available corners. This scenario is highly possible if a Macro that has been inserted in the contour pool is significantly large. To tackle this issue, a smoothing process is performed before this step. During this process, gaps are cleared, similarly to what is already performed, but at this stage the margin is larger, in order to create more corners. This can potentially yield worse results for designs with uniform Macros, since it clears openings in which a Macro can fit, but it is notably effective for designs with Macro diversity. If, even after this adjustment, a Macro still cannot find an available corner, then this Macro placement is deemed unsuccessful.

Chapter 5

Macro Placement Evaluation and Flow Settings

5.1 Top-level Macro Placement flow

In Chapter 4, our Macro Placement algorithm was presented, starting from the CPtree construction and ending with a final, overlap-free Macro Placement. However, the resulting Macro Placement is rarely the optimal one. In order to detect a superior formation of Macros, each Macro Placement needs to be evaluated, and a new attempt must begin. Only after multiple Macro Placement attempts can one conclude that a result is optimal, or at least approaches the optimal.

As shown in Algorithm 3, the tree construction is carried out only once. Then, a Macro Placement is extracted from the packing procedure. Afterwards, the Macro Placement is evaluated, and if it is deemed to be an improvement, it is stored as the best so far, else it is discarded. For all subsequent iterations, a perturbation is performed on the tree, meaning that the new Macro Placement will differ as well. This is repeated until the max iteration count is met.

5.2 Results Evaluation

It is essential to evaluate each Macro Placement, in order to decide whether it is superior to a different placement of the same set of Macros. Therefore, we need to determine a suitable set of parameters that will be used for this process.

Algorithm 3 Macro Placement Optimization Flow

```

1: Input: macro_list
2: Output: best_macro_placement
3: current_tree  $\leftarrow$  build_tree(macro_list);
4: while i < max_i do
5:   if i > 0 then
6:     apply_perturbation(current_tree);
7:   end if
8:   current_macro_placement  $\leftarrow$  pack_tree(current_tree);
9:   score  $\leftarrow$  evaluate_macro_placement(current_macro_placement);
10:  if i > 0 then
11:    best_tree  $\leftarrow$  current_tree;
12:    best_macro_placement  $\leftarrow$  current_macro_placement;
13:  else
14:    current_tree  $\leftarrow$  best_tree;
15:  end if
16:  i  $\leftarrow$  i + 1;
17:  if i = max_i then
18:    break;
19:  end if
20: end while
21: return best_macro_placement;

```

The first parameter is the Half-Perimeter Wire Length (HPWL), which is a widely applied criterion throughout the entire ASIC flow. It is an approximation for the length of the interconnections between the connected nodes, and it is deemed necessary to try minimizing it. When it comes to a Macro Placement, we do not have control of the entirety of the placeable instances, but only those connected to the Macros that we deal with. Consequently, the minimization only applies to the fraction of the HPWL that is affected by the Macro Placement.

The second parameter is the total wasted area of the design. This includes:

- The narrow openings that are consistently smoothed.
- The areas smoothed before the contour pool packing.
- The areas completely isolated from the rest of the design.

For the first two, the computation is exact, and the area of each rectangle is added to the total area. For the third, the computation is not exact, since the isolated area is, in the general case, not a rectangle but a rectilinear polygon, and computing its area is a challenging task. Therefore, we use the upper bound as an approximation. The sum of all three sections is added and represents the total wasted area. This parameter is also attempted to be minimized.

The third and final parameter is the rectangle that a global placer theoretically sees when trying to place all Standard Cells. Even though such an algorithm is more complicated, we make this assumption to simplify things. An example of a placement rectangle is shown in Figure 5.1. As displayed, this rectangle approximately represents the free area of the design, and, in contrast to the previous two parameters, must be maximised. It is important to note that the extraction of the optimal rectangle is quite complicated, so the result may not always reflect the ideal solution. Under the circumstances that there exists at least one non-boundary pre-placed Macro, we simply ignore that, following the assumption that this is an outlier.

The equation employed to calculate the score of a Macro Placement is:

$$\text{score} = - \left(\frac{\text{curHPWL} - \text{prevHPWL}}{\text{prevHPWL}} + \frac{\text{curWastedArea} - \text{prevWastedArea}}{\text{maxWastedArea}} - \frac{\text{curRectangleArea} - \text{prevRectangleArea}}{\text{maxRectangleArea}} \right) \quad (5.1)$$

We want this score to be as high as possible. It is important to highlight the use of percentages as a form of scaling, since each metric is measured in different units. Furthermore, the

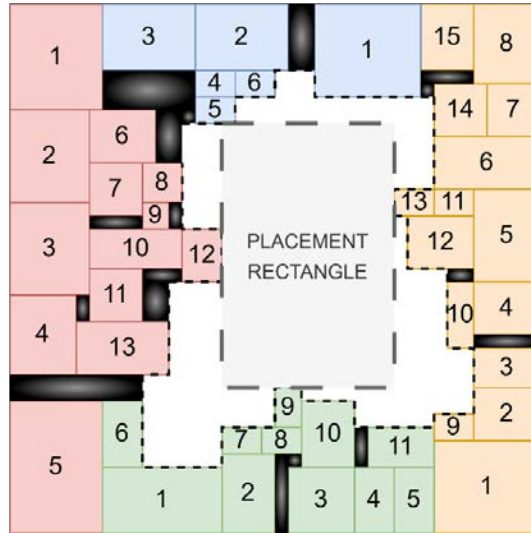


Figure 5.1: Illustration of the Wasted Area collected, as well as the optimal Placement Rectangle computed after a Macro Placement

use of the maximum values, instead of the previous values, as the denominators of the second and third terms, is preferred due to the large change rates of the wasted area and rectangle area parameters.

5.3 CPtree Perturbations

All of the above procedures aim to translate the initial CPtree into its respective packing formation. Of course, the optimality of the initial CPtree is not guaranteed, which raises the need for subsequent, successive optimizations to be performed, with the aim of reaching an ideal solution.

Since the packing is entirely dependent on the formation of the CPtree, if this structure changes even slightly, then the result after the packing algorithm will be different. Therefore, we will perform such a change, which is called an “action” or “perturbation” after every packing iteration finishes. This scheme is inspired by the Simulated Annealing method [7] followed in multiple Floorplanning approaches. In this method, a discrete set of perturbations is determined, and each perturbation is given an initial probability. Generally, perturbations that are more likely to improve the results are given higher probabilities.

The set of perturbations that will be utilized in this work consists of:

- Rotation of a node

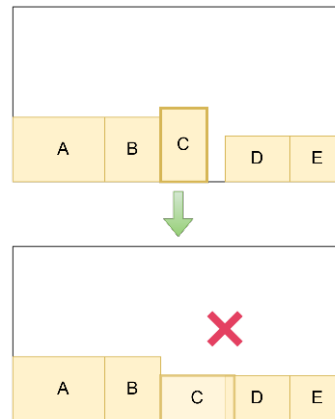


Figure 5.2: An example of the Rotation perturbation failing due to the Anchor Width Constraint.

- Move of a node
- Swap of two nodes
- Change of the Switching Point of a Packing Anchor branch

The first perturbation is the rotation of a Macro by 90 degrees. Even though rotation occurs during the packing process as well, this perturbation changes the default orientation of the Macro. We limit ourselves to two orientations, which correspond to the two unique aspect ratios of a rectangle. The only limitation regarding this action is related to the Anchor Width Constraint, as shown in Figure 5.2.

The next perturbation is the move of a node, also called “insertion”. During this action, we remove a node from one part of the CPtree and insert it to another part. This action can be separated into three kinds of moves:

- Boundary Insertion
- Left Child Insertion
- Right Child Insertion

During the boundary insertion, we insert a move to one of the boundary subtrees, if there is available space. As for the left and right child insertions, we insert the node as a left or right child of a node whose respective child spot is available. This insertion can also fail due to the Anchor Width Constraint.

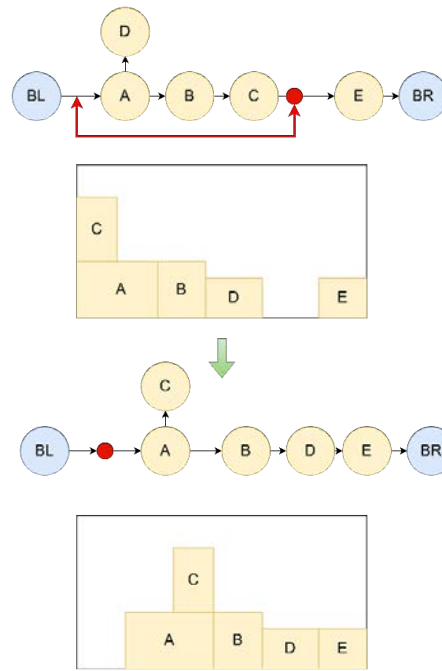


Figure 5.3: An example of the impact of a Switching Point change.

Another perturbation is the swap between two nodes. During this action, the two candidates exchange their position in the CPtree, which includes their parents and children, as well as their packing behaviour, but not their orientation or other physical characteristics. This perturbation can, just like the previous two, fail if the Anchor Width Constraint is violated while attempting to swap a boundary node with another node that does not fit in the respective boundary.

The final perturbation is the change of the switching point. This action changes the position on a boundary branch, on which the packing direction changes from backward to forward, applying to boundary as well as non-boundary nodes. Even though this action does not target one specific Macro, it is guaranteed to change the position of multiple Macros. Therefore, it is useful if we desire to stir the current packing formation. An example of how a change in the switching point affects the packing is shown in Figure 5.3.

5.4 Optimization Settings

The core parts of the Macro Placement optimization flow are completed, but there is still room for configuration by the user. It is possible to apply different settings to the optimization flow, with the aim of achieving improved results. These options are:

- **Probabilities:** The initial and default probability of each of the four action is equal, i.e. 25%. A potentially beneficial optimization would be to dynamically change these probabilities during the algorithm, depending on the success rate and effectiveness of each one. Thus, when a move is successful, its probability is increased. The ratio of the increase is dependent on how much it improves the packing based on the three metrics that were previously discussed.
- **Macro Choice:** One approach is to apply full randomness and pick a random Macro or position in each iteration. Another, more intelligent approach, is to attempt determining the most “critical” Macros, whose current placement is either far from its optimal position, or it prevents the rest of the Macros from being packed optimally. Therefore, we introduce four parameters which apply to placeable nodes, and for which we measure a “score” during every iteration. The Macro with the highest summation of the four scores, after appropriate scaling, is deemed to be the most critical. The four parameters are:
 - Distance from the initial position
 - Inducing Wasted Area
 - Effect on Contour
 - Effect on the optimal Placement Rectangle

An example is displayed in Figure 5.4. It is clear that Macro C causes Wasted Area, E affects the contour negatively, and G serves as an obstacle to the Placement Rectangle. Therefore, these three nodes are deemed more “critical” than the rest.

- **Solution Exploring:** The current optimization method continuously searches for an improved formation based on the present one, which has been found to be the optimal so far. One drawback of this is that it does not explore the entire space of the possible formations. It is possible that, in order to uncover the optimal placement, one “bad” action has to be performed, which is not currently allowed. Therefore, we implement a Simulated Annealing-like algorithm, which can accept moves that yield worse results, depending on how bad the negative score is and how far on the iteration count the current iteration sits. We can implement various functions to control the acceptance, but

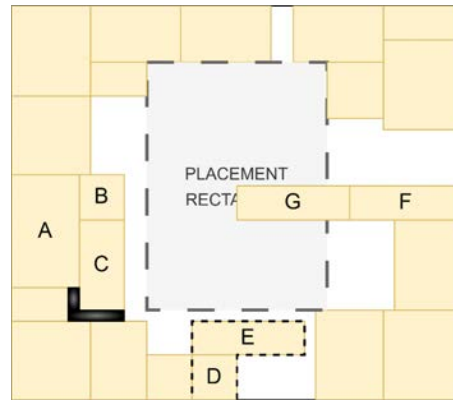


Figure 5.4: An example illustrating the criticality of Macros C, E and G.

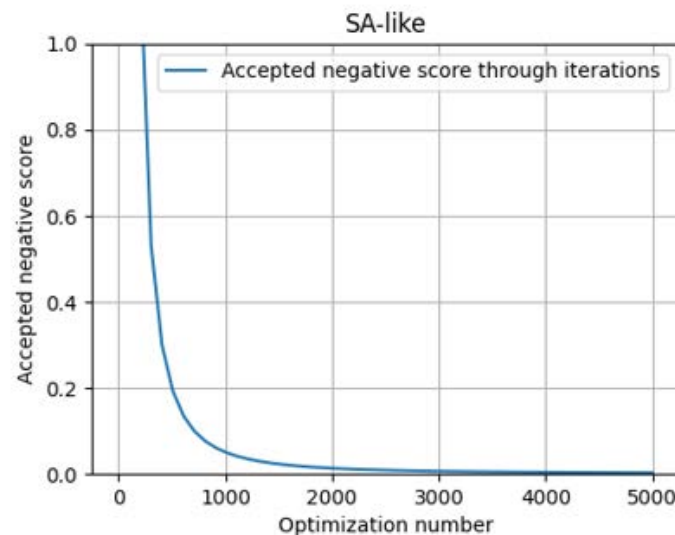


Figure 5.5: A graph displaying the function used for the Simulated Annealing-like algorithm.

a function such as this shown in Figure 5.5 is found to be efficient. If many consecutive solutions yield bad results, we revert to the best solution and continue.

- **Action Choice:** By default, the next perturbation in each iteration is chosen based on the probabilistic model that was previously discussed. Alternatively, two custom flows are developed, which ensure that, for the highest scoring Macro, multiple possible actions are explored. It is important to be reminded that, even if the target Macro of an operation is known, the position to be inserted in, or the Macro with which it is swapped, remain random. The two flows are shown in 5.6. The first one focuses on performing the straightforward operations first, which do not include randomness excluding the Macro to rotate on the position to change, while the second one begins with the swaps and moves. Of course, many other custom combinations may be developed.

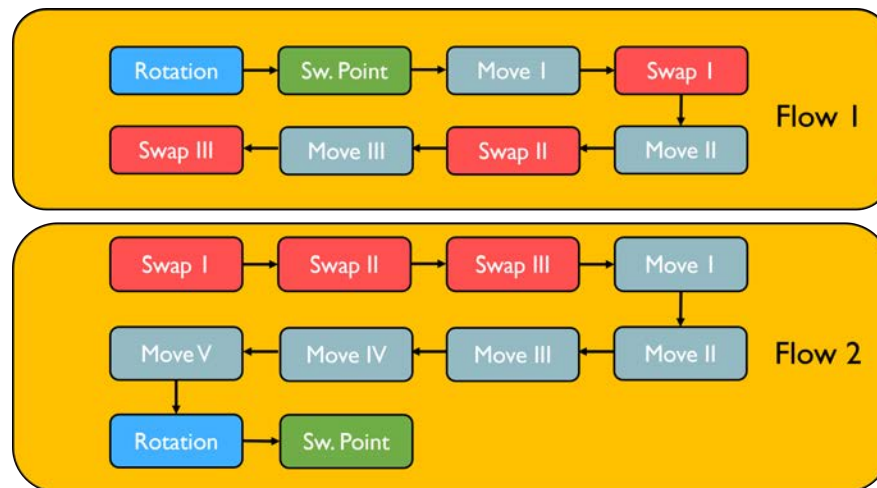


Figure 5.6: The two alternative flows which we experiment on.

5.5 Top-level Placement flow

In the previous sections, we analyzed the Macro Placement flow and its multiple steps. It is essential for this flow to be accompanied by preceding and succeeding steps that support it. Currently, the only necessary preceding step is the Initial Placement that, as was presented, is thoroughly used for the computation of the Macro positions. As for the succeeding steps, a Macro Legalization must follow, which in turn must be followed by a Macro Fixing. These two actions are vital for the subsequent Standard Cell Placement, for which the Macros must serve as fixed objects. Finally, the Standard Cells are legalized, and the exact position of all components is known.

However, there is still room for improvement. Instead of relying on a single initial placement, which may not always contain useful or representative information, we can perform a single iteration of Macro Placement, followed by Macro Legalization and fixing, and a Standard Cell Placement, in order to obtain a more meaningful position for every Standard Cell, as well as to create a strong foundation for the standard multi-iteration flow of Macro Placement. This flow is displayed in Figure 5.7.

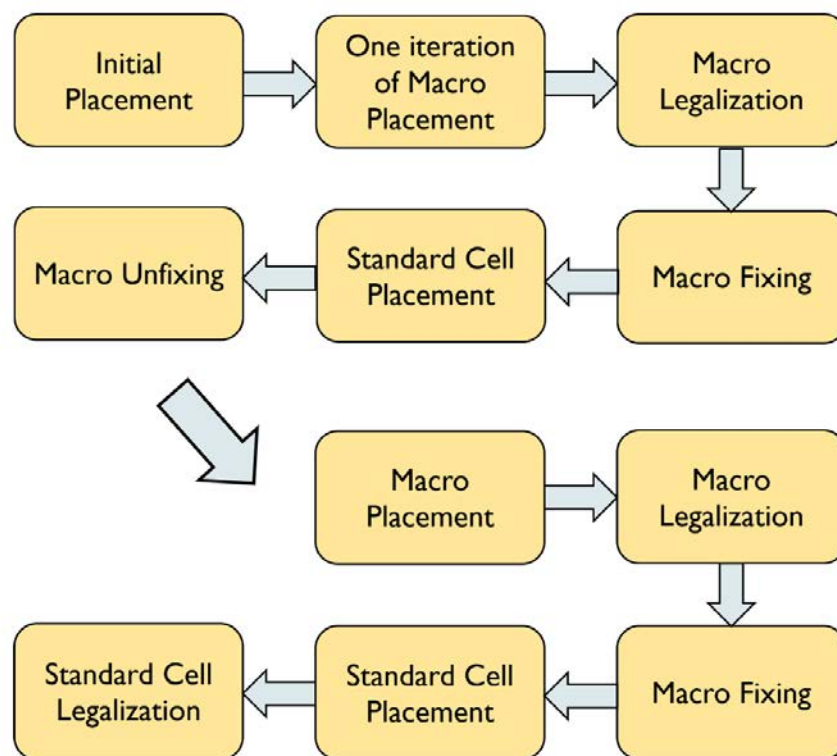


Figure 5.7: The top-level Placement flow, when an initial Macro Placement is performed.

Chapter 6

Experimental Results

The experiments are performed on nine designs, whose characteristics can be seen in Table 6.1. As displayed, some designs were slightly altered, to experiment on pre-placed Macros as well. While Ariane includes Macros of the same size, the other circuits consists of Macros of different size.

Table 6.1: Designs experimented on

Design	Cells	Macros	Boundary Fixed Macros	Non-boundary Fixed Macros
ariane	64916	133	0	0
ariane*	64916	0	8	0
ariane**	64916	0	0	4
mgc_fft_a	30625	0	0	0
mgc_matrix_mult_a	149650	0	0	0
superblue1	1209716	0	0	0
superblue16	981559	0	0	0
superblue16*	981559	0	5	0
superblue18	768726	0	0	0

6.1 Macro Placement Flow Experiments

In Table 6.2, the difference in the three evaluation metrics, between fixed and dynamic probabilities is displayed. It is important to note that these metrics do not form an accurate

representation of the quality of a Macro Placement on their own, since typically designs are evaluated after Global Placement. However, applying the Global Placement algorithm for each Macro Placement experiment would be time-consuming, therefore only the Macro Placement results are presented. Applying dynamic probabilities can potentially improve the result quality, although no clear and obvious improvement occurs.

Table 6.2: Increase in the evaluation metrics, for dynamic vs static probabilities

Design	HPWL Change	Wasted Area Change	Placement Rectangle Change
ariane	+0.56%	-51.98%	-6.79%
ariane*	-1.34%	-16.79%	+13.61%
ariane**	+1.47%	+16.75%	+2.52%
mgc_fft_a	+0.2%	+27.4%	+2.2%
mgc_matrix_mult_a	+1.01%	+9.66%	+0.04%
superblue1	-0.63%	+26.19%	-0.05%
superblue16	+1.64%	-33.11%	-0.07%
superblue16*	+4.07%	+3.15%	+1.83%
superblue18	+0.3%	+47.73%	-3.89%
Average	+0.81%	+3.22%	+1.04%

In Table 6.3, the increase in successful operations, as well as in the three metrics, between Macro Placement with random Macro choice and with the critical Macros scoring system is presented. Once again, no clear improvement can be displayed. The user can experiment with both flows to detect the optimal result. It is important to note that, for designs with very few Macros, there is no merit in applying the scoring system, hence the exclusion of two designs in the table.

On the other hand, in Table 6.4, it is evident that the Simulated Annealing-like approach did not improve the results produced by the Greedy approach. As displayed, the results actually became much worse, signifying the failure of this approach, and the need for better settings, or a different function.

The success percentage of each operation can be displayed in Table 6.5. We can observe that, on average, swapping is the best operation, but for the Superblue designs, which include Macros with different shapes, and most of which have a large aspect ratio, rotation is more

Table 6.3: Increase in the evaluation metrics, for critical Macros vs random Macros

Design	Increase in successful operations	HPWL Change	Wasted Area Change	Placement Area Change
ariane	+1.14%	+1.9%	+78.9%	+11.37%
ariane*	-25.41%	+1.89%	+19.83%	+4.17%
ariane**	+131.31%	-2.35%	-64.16%	+17.25%
superblue1	-25.83%	+2.82%	-11.72%	+11.24%
superblue16	-10.42%	-3.35%	-54.88%	+9.34%
superblue16*	-84.56%	-1.73%	+215.56%	-55.62%
superblue18	+15.37%	-1.19%	-36.52%	+6.83%
Average	+0.23%	-0.29%	+21%	+0.65%

successful. This makes sense, because for designs like Ariane, which include Macros whose shape approaches square, rotating usually changes the Macro Placement very little, in contrast to designs with rectangular, long and thin Macros. We can also conclude that the move or insertion is usually the second best operation, while the switching point change is the most ineffective operation.

In Table 6.6, the experimental results of the two custom flows presented earlier are displayed. Both flows can serve as a decent alternative to the probabilistic approach, while flow 2 produces slightly better results than flow 1.

In figure 6.1, the HPWL for various initial placement configurations is displayed. In order to obtain more meaningful results, Global Placement is applied, using a force-directed algorithm, followed by Legalization. We can observe that the quality of the Initial Placement is essential, and it significantly affects the final result. When the Initial Placement is a Quadratic Placement (QP), the final HPWL is, in most cases, worse than when it is a DEF with meaningful information. The QP uses arithmetical equations to place the components, without accounting for overlaps, so it sometimes leaves all Macros on the center of the chip, leading to an unsuccessful Macro Placement. Another observation that can be made is that the additional Global Placement does not show a significant difference compared to the standard flow.

Table 6.4: Increase in the evaluation metrics, for Greedy vs SA-like solution exploring

Design	Increase in successful operations	HPWL Change	Wasted Area Change	Placement Area Change
ariane	-95.00%	+9.56%	+344.04%	-3.63%
ariane*	-94.59%	+12.46%	+119.95%	-16.80%
ariane**	-85.74%	+11.16%	+123.62%	+0.14%
mgc_fft_a	0.00%	0.00%	0.00%	0.00%
mgc_matrix_mult_a	0.00%	0.00%	0.00%	0.00%
superblue1	-94.32%	+2.58%	+7.43%	-27.50%
superblue16	-68.75%	+6.79%	+12.26%	-13.77%
superblue16*	-93.07%	+17.20%	+197.22%	-0.02%
superblue18	-92.27%	+9.43%	+56.89%	-6.98%
Average	-69.30%	+7.69%	+95.71%	-7.62%

6.2 Top-level Placement Flow Experiments

Perhaps the most meaningful comparison that we can make, is that between our Macro Placement flow and a Mixed-size flow, which comes in the form of the Global Placement algorithm. When Standard Cells and Macros are placed simultaneously, using a force-directed algorithm, the results are on average significantly worse than our Macro Placement flow, as displayed in Table 6.7. In some cases, our flow manages to reduce the HPWL by more than three times.

To conclude the Experimental Results section, the flow stages of our Macro Placement algorithm are displayed in Figure 6.2. In 6.2 α' , the Initial Placement is displayed. In 6.2 β' , the Macros are placed based on the Initial Placement, while in 6.2 γ' , the Standard Cells are placed using a Global Placement algorithm, while the Macros are fixed. Finally, in 6.2 δ' , the cells are legalized.

In Figure 6.3, the difference between our Macro Placement flow and a Global Placement algorithm is displayed. While the Global Placer places the Macros without any alignment or special care, the Macro Placer manages to reserve adequate space for the Standard Cells, leading to a significantly smaller HPWL.

Design	def, mp, gp	qp, mp, gp	def, mp, gp, mp, gp	qp, mp, gp, mp, gp
ariane	1412673	7391747	1243213	2213668
ariane*	1256302	10282059	1256247	1305008
ariane **	1183949	7034056	1190315	1222648
mgc_fft_a	3157023	2972998	3198989	3132590
mgc_matrix_mult_a	12468727	12661618	11748065	17158543
superblue1	494251520	436173913	385078515	560117633
superblue16	163751961	334360034	168177585	232361557
superblue16*	162647049	241005408	174633212	218141422
superblue18	96957912	136925649	112386384	295874325

Figure 6.1: Comparison of HPWL between different flow approaches. The red colour indicates a worse result, while the green colour indicates a better result.

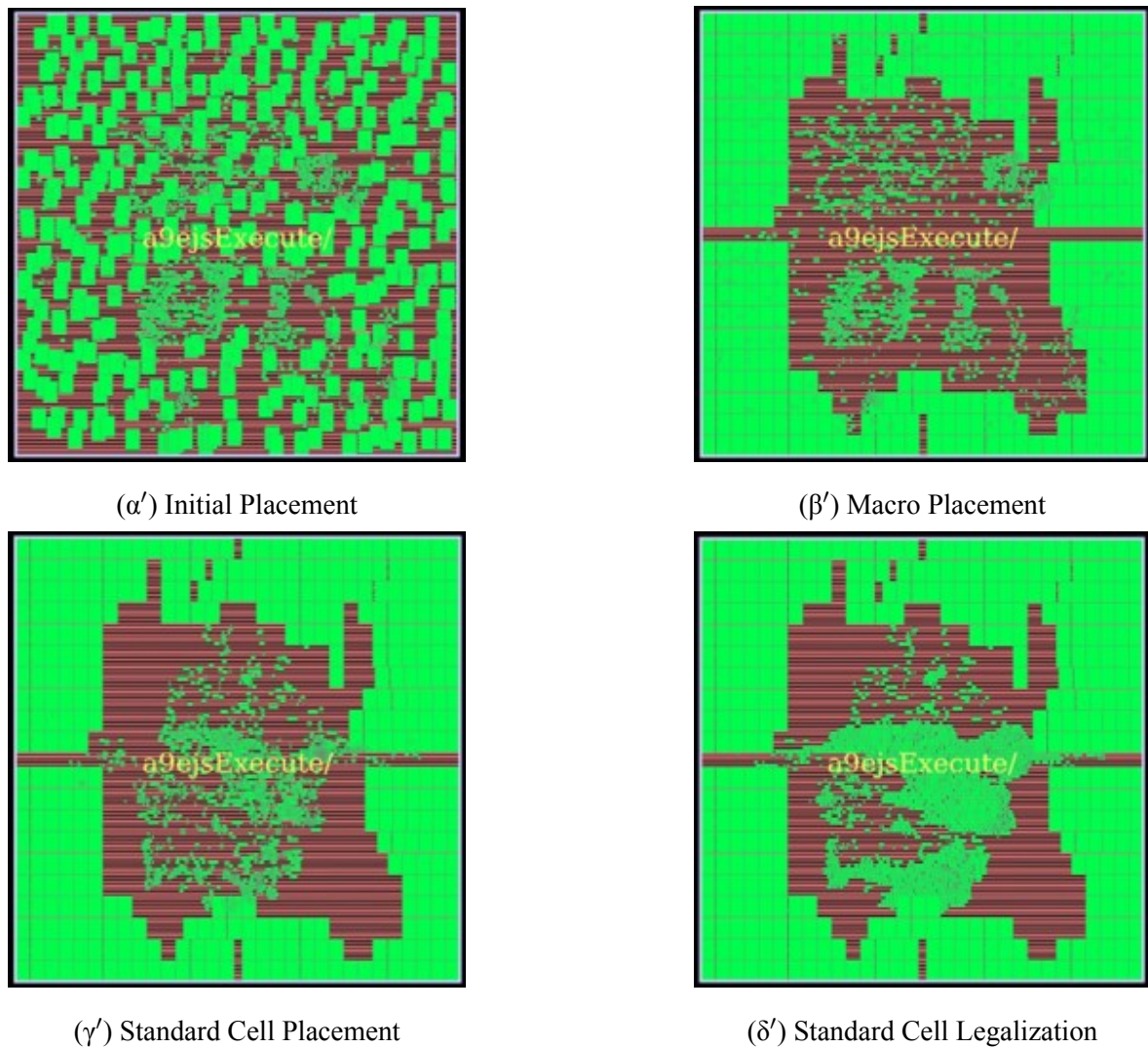


Figure 6.2: The discrete stages of the Placement flow for a design with Macros

Table 6.5: Success percentage for each perturbation type

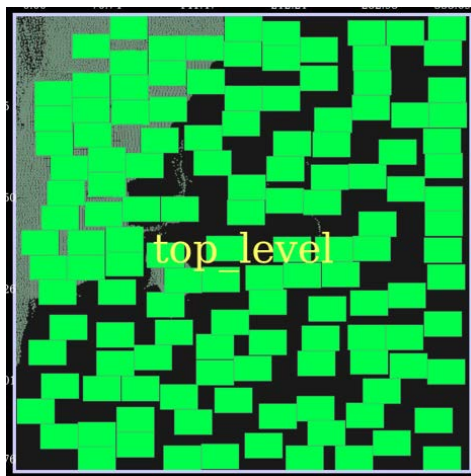
Design	Perturbation Success (%)	Rotation Success (%)	Move Success (%)	Swap Success (%)
ariane	16.42	19.70	56.77	7.11
ariane*	16.52	24.26	48.39	10.84
ariane**	21.14	21.14	45.61	12.10
mgc_fft_a	12.50	22.92	37.50	27.08
mgc_matrix_mult_a	19.44	38.89	22.22	19.44
superblue1	37.93	35.10	19.78	7.19
superblue16	35.62	31.19	25.66	7.52
superblue16*	39.06	27.36	21.32	12.26
superblue18	40.84	35.07	17.76	6.33
Average	26.61	28.40	32.78	12.21

Table 6.6: Increase in metrics between Probabilistic approach and Flow1/Flow2

Design	Flow 1			Flow 2		
	HPWL Change (%)	Wasted Area Change	Placement Area Change	HPWL Change (%)	Wasted Area Change	Placement Area Change
ariane	+2.12%	+16.73%	+2.34%	+1.50%	-38.42%	+1.71%
ariane*	-0.57%	-10.15%	+8.31%	+1.16%	+4.26%	-0.44%
ariane**	-0.28%	-5.19%	+16.09%	+2.21%	+11.00%	+16.04%
superblue1	+1.54%	-10.89%	+5.60%	+0.48%	-27.11%	-6.22%
superblue16	-0.72%	-24.17%	+8.43%	-3.74%	-26.35%	+12.10%
superblue16*	+0.75%	-3.46%	-50.65%	+3.60%	-1.82%	-12.35%
superblue18	-0.40%	-13.29%	+0.67%	+0.22%	-23.66%	+0.69%
Average	+0.35%	-7.20%	-1.32%	+0.78%	-14.59%	+1.65%

Table 6.7: Comparison of HPWL between Global Placement and the best Macro Placement result

Design	GP HPWL	Best MP HPWL	Change after MP
ariane	3933742	1243213	-68.40%
ariane*	3627777	1256247	-65.37%
ariane**	3933184	1183949	-69.90%
mgc_fft_a	3881851	2972998	-23.41%
mgc_matrix_mult_a	10125438	11748065	+16.03%
superblue1	341904930	385078515	+12.63%
superblue16	309243240	163751963	-47.05%
superblue16*	290514332	218141422	-24.91%
superblue18	109898257	96957912	-11.77%



(α') Result after Global Placement,
HPWL = 3933742



(β') Result after Macro Placement,
HPWL = 1256302

Figure 6.3: Comparison between Global and Macro Placement for Ariane

Chapter 7

Conclusions and Future Work

In this thesis, we have analyzed the problem of the placement of Macros. We have observed multiple challenges and corner cases, which require special handling. In order to tackle these challenges, we have presented a state-of-the-art packing-based algorithm for Macro Placement, which can handle pre-placed Macros, utilizes the CPtree [5] and Circular Contour [6] algorithms, and achieves a reduced HPWL compared to an approach utilizing a Global Placement algorithm for both Standard Cells and Macros.

There are several potential areas for future work. One of these is Timing-driven Macro Placement, meaning that Static Timing Analysis could be employed in order to consider the critical timing paths of the design and place the Macros accordingly. Furthermore, a routability criterion could be utilized, which could provide further insights on the Macro Placement results, aside from the HPWL. An additional feature that our algorithm could benefit from, would be the optimization of Macro Placement for Soft Macros. Since these objects can have variable aspect ratios, the packing process of our algorithm could be improved to support Soft Macros by finding the optimal shape.

Finally, a comparison with different Macro Placement algorithms would be particularly interesting. It could reveal the strengths and weaknesses of our algorithm compared to the other state-of-the-art approaches, highlight areas of improvement, and encourage further innovation and fresh ideas.

Bibliography

- [1] S. Sapatnekar C. Alpert, D. Mehta. *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications, 2008.
- [2] M. Yazgan et ali A.Mirhoseini, A. Goldie. A graph placement methodology for fast chip design. In *Nature* 594, 2021.
- [3] Anthony Agnesina, Puranjay Rajvanshi, Tian Yang, Geraldo Pradipta, Austin Jiao, Ben Keller, Brucek Khailany, and Haoxing Ren. Autodmp: Automated dreamplace-based macro placement. pages 149–157, 03 2023.
- [4] Tung-Chieh Chen, Ping-Hung Yuh, Yao-Wen Chang, Fwu-Juh Huang, and Denny Liu. Mp-trees: A packing-based macro placement algorithm for mixed-size designs. pages 447–452, 07 2007.
- [5] Yi-Fang Chen, Chau-Chin Huang, Chien-Hsiung Chiou, Yao-Wen Chang, and Chang-Jen Wang. Routability-driven blockage-aware macro placement. *Proceedings - Design Automation Conference*, 06 2014.
- [6] Chien-Hsiung Chiou, Chin-Hao Chang, Szu-To Chen, and Yao-Wen Chang. Circular-contour-based obstacle-aware macro placement. pages 172–177, 01 2016.
- [7] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science (New York, N.Y.)*, 220:671–80, 06 1983.
- [8] Chung-Kuan Cheng Pei-Ning Guo and T. Yoshimura. "an o-tree representation of non-slicing floorplan and its applications". In *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361*, New Orleans, LA, USA, 1999.

- [9] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. B*-trees: a new representation for non-slicing floorplans. In *Proceedings 37th Design Automation Conference*, pages 458–463, 2000.
- [10] Jai-Ming Lin, Yao-Wen Chang, and Shih-Ping Lin. Corner sequence - a p-admissible floorplan representation with a worst case linear-time packing scheme. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(4):679–686, 2003.
- [11] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Rectangle-packing-based module placement. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pages 472–479, 1995.
- [12] Chung-Kuan Cheng, Andrew B. Kahng, Sayak Kundu, Yucheng Wang, and Zhiang Wang. Assessment of reinforcement learning for macro placement, 2023.
- [13] Andrew Kahng, Ravi Varadarajan, and Zhiang Wang. Rtl-mp: Toward practical, human-quality chip planning and macro placement. pages 3–11, 04 2022.
- [14] Andrew Kahng, Ravi Varadarajan, and Zhiang Wang. Hier-rtlmp: A hierarchical automatic macro placer for large-scale complex ip blocks, 04 2023.
- [15] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. pages 1–6, 06 2019.
- [16] Yuan Pu, Tinghuan Chen, Zhuolun He, Chen Bai, Haisheng Zheng, Yibo Lin, and Bei Yu. Incremacro: Incremental macro placement refinement. pages 169–176, 03 2024.
- [17] Yunqi Shi, Ke Xue, Lei Song, and Chao Qian. Macro placement by wire-mask-guided black-box optimization, 06 2023.