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

Sequence Pair Floorplanning Optimisation for 2D/3D ASICs

Diploma Thesis

Spyridon Kyritsis

Supervisor: Dr. Christos Sotiriou

September 2023

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

Sequence Pair Floorplanning Optimisation for 2D/3D ASICs

Diploma Thesis

Spyridon Kyritsis

Supervisor: Dr. Christos Sotiriou

September 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Βελτιστοποίηση Χωροθέτησης με Χρήση του Αλγορίθμου Sequence Pair για 2D/3D Ολοκληρωμένα Κυκλώματα

Διπλωματική Εργασία

Σπυρίδων Κυρίτσης

Επιβλέπων/πουσα: Δρ. Χρήστος Σωτηρίου

Σεπτέμβριος 2023

Approved by the Examination Committee:

Supervisor **Dr. Christos Sotiriou**
Professor, Department of Electrical and Computer Engineering,
University of Thessaly

Member **Dr. Fotios Plessas**
Professor, Department of Electrical and Computer Engineering,
University of Thessaly

Member **Dr. Georgios Stamoulis**
Professor, Department of Electrical and Computer Engineering,
University of Thessaly

Acknowledgements

I would like to wholeheartedly thank Nikolaos Sketopoulos for his contribution to this work. His insight throughout this project has been priceless and his patience beyond measure.

I would also like to thank Prof. Christos Sotiriou for his trust over the years as, well as my CAS Lab colleagues for the incredible work environment and invaluable brainstorming sessions. Special thanks to Georgios Gkountroumanis for his help with the leon design and his expertise with the partitioning tool.

I also want to thank my wonderful friends for always caring for me and tolerating me. Vaios, Dimitra, Giannis, Margarita, Thanos and Konstantina thanks for all the great memories and for shaping me into who I am today.

Finally, I want to express my gratitude towards my parents, whose endless support and love accompanied me, through my 5-year academic journey.

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

Spyridon Kyritsis

Diploma Thesis

Sequence Pair Floorplanning Optimisation for 2D/3D ASICs

Spyridon Kyritsis

Abstract

Floorplanning has tremendous effects on the final qualities of the target Application Specific Integrated Circuits (ASIC), such as its Power Consumption, Performance and Total Area. Since floorplanning is one of the earliest stages in the ASIC design flow, it is governed by various degrees of design freedom, ranging from the shape of the chip and its sub-units to pin placement and more. However, such freedom comes at a cost. Missing the mark early on in the process can cause irreparable damage further down the road, which can lead to costly design cycles. In this work, we will tackle the floorplanning step from the scope of area minimisation, specifically Rectangle Packing. In particular, we will focus in a packing algorithm called Sequence Pair. Sequence Pair is a packing methodology, that relies in a small set of permutations, aiming to optimise the whitespace of a design, by utilising an iterative approach. We will analyse in great detail the intrinsics and the nuances of the algorithm, and we will provide insight into how it operates behind the curtain. By observing this behaviour, we will introduce novel optimisation moves and starting solutions for the algorithm, that massively accelerate improve its Quality of Results (QoR). The literature has long praised the efficiency of Simulated Annealing (SA) as an optimisation method for problems like the one Sequence Pair handles, but it has been incredibly secretive about the internals of their methods. Because SA is a very parameter-sensitive method, we developed a mathematical model for it and extensively experimented to find near-optimal parameters for different uses. The SA knowledge we obtained, along with our novel strategies, were incorporated into a highly customizable optimisation loop, that also featured a dynamic probability adjustment system for the move selection, and produced great results. We will be comparing each update we developed in an additive matter to determine their effectiveness. Finally, we will examine what the literature has to offer for the inverse problem, i.e. how given a floorplan we can obtain its Sequence Pair representation.

Keywords:
Floorplanning, Sequence Pair, Simulated Annealing, ASIC

Διπλωματική Εργασία

Βελτιστοποίηση Χωροθέτησης με Χρήση του Αλγορίθμου Sequence Pair για 2D/3D Ολοκληρωμένα Κυκλώματα

Σπυρίδων Κυρίτσης

Περίληψη

Η Χωροθέτηση έχει φοβερές επιπτώσεις στα τελικά χαρακτηριστικά των Ολοκληρωμένων Κυκλωμάτων, τόσο ως προς την Κατανάλωση Ισχύος, την Επίδοση αλλά και την Συνολική Επιφάνεια. Η Χωροθέτηση, όντας ένα από τα αρχικά στάδια της ροής σχεδιασμού Ολοκληρωμένων Κυκλωμάτων, ενέχει πολλούς βαθμούς ελευθερίας, από το σχήμα του τελικού κυκλώματος και των υπομονάδων του έως την τοποθεσία των ακροδεκτών Εισόδου-Εξόδου και άλλα. Όμως, μια τέτοια ελευθερία έχει ιδιαίτερο τίμημα. Λανθασμένες σχεδιαστικές αποφάσεις σε αυτό το πρώιμο στάδιο, μπορούν να έχουν σοβαρές συνέπειες στα μετέπειτα βήματα της ροής, γεγονός που μπορεί να επιφέρει κοστοβόρους σχεδιαστικούς κύκλους. Σε αυτή την εργασία θα εξετάσουμε τη Χωροθέτηση από την σκοπιά της ελαχιστοποίησης επιφάνειας και ειδικότερα από αυτή του πακεταρίσματος. Συγκεκριμένα θα επικεντρωθούμε σε έναν αλγόριθμο πακεταρίσματος, το λεγόμενο Sequence Pair. Το Sequence Pair είναι μία μεθοδολογία πακεταρίσματος, που βασίζεται σε μία μικρή ομάδα κινήσεων, και απσκοπεί στην ελαχιστοποίηση του νεκρού χώρου μέσω μιας επαναληπτικής διαδικασίας. Θα αναλύσουμε σε βάθος τις ιδιαιτερότητες του αλγορίθμου και θα παρέχουμε ενόραση στο πως αυτός λειτουργεί εσωτερικά. Παρατηρώντας τη συμπεριφορά του, θα εισάγουμε νέες κινήσεις και καινοτόμες αρχικές λύσεις στο ρεπερτόριο του αλγορίθμου, οι οποίες βελτιώνουν σημαντικά την ποιότητα των αποτελεσμάτων του. Η βιβλιογραφία έχει εκθειάσει την αποτελεσματικότα του Simulated Annealing ως μεθόδου βελτιστοποίησης για προβλήματα όπως το Sequence Pair, αλλά παραμένει κρυπτική ως προς τις ακριβείς παραμέτρους του. Καθώς το SA είναι μία ιδιαίτερα αριθμιτικά ευαίσθητη μεθοδολογία, αναπτύξαμε ένα μαθηματικό μοντέλο για το SA και μέσω εκτενών πειραμάτων, εντοπίσαμε ομάδες παραμέτρων για διάφορες χρήσεις. Η αποκτηθείσα γνώση πάνω στο SA, σε συνδυασμό με τις καινοτομές νέες κινήσεις και αρχικες λύσεις συνδυάστηκαν σε έναν ευέλικτο επαναληπτικό βρόχο βελτιστοποίησης, ο οποιός χρησιμοποιεί ένα σύστημα δυναμικής ρύθμισης πιθανότητας για την επιλογή κινήσεων, επιφέροντας εξαιρετικά αποτελέσματα. Θα συγκρίνουμε όλες τις νέες βελτιστοποιήσεις με αθροιστικό τρόπο για να αξιολογήσουμε την αποτελεσματικότητά τους. Τέλος, θα αναλύσουμε τη βιβλιογραφία ως προς το αντίστροφο πρόβλημα, δηλαδή δεδομένης μίας Χωροθέτησης, πώς μπορούμε να εξάγουμε την αναπαράστασή της κατά Sequence Pair.

Λέξεις-κλειδιά:
Χωροθέτηση, Sequence Pair, Simulated Annealing, Ολοκληρωμένα Κυκλώματα

Table of contents

List of figures

List of tables

List of Algorithms

Acronyms

ASIC Application Specific Integrated Circuits. xii, 58

DPA Dynamic Probability Adjustment. 41, 43, 52, 53

ECO Engineering Change Order. 27

EDA Electronic Design Automation. 2, 20, 58

HCG Horizontal Constraint Graph. xvii, 14–16, 25

IC Integrated Circuit. 1, 13

LCS Longest Common Subsequence. 18, 24

PSO Particle Swarm Optimisation. 20

QoR Quality of Results. xii, 25, 43, 53, 58

SA Simulated Annealing. xii, xiii, 1, 22, 31, 34, 39, 42, 43, 51–53, 58

SoC System on Chip. 21

TSV Through Silicon Via. 20

VCG Vertical Constraint Graph. xvii, 14–16, 25

Chapter 1

Introduction

The journey of modern electronics is truly fascinating. From the invention of the vacuum tube and later that of the transistor and the Integrated Circuit (IC), the leap humanity has made in just over a century is extraordinary. Such is the sheer scale of modern ICs that the only way to design and optimize them is to partition them into smaller sub-chips. Thus, the need for fast end efficient floorplanning schemes is paramount, as it effects multiple parameters of the final product from its total area, to its power consumption and performance.

1.1 Aims of this work

In this work, we will explore the research field IC floorplanning. More specifically we will focus on a famous floorplanning algorithm called Sequence Pair and we will expand its core operations by with new moves. Furthermore, we will showcase how the Sequence Pair relations between the floorplanning blocks can be manipulated to create both better starting positions, new optimisation moves as well as solve problems with blocks that may end up out of core.

To test the quality of results from the Sequence Pair algorithm, as well as the proposed new moves, we devised a novel, multi-phase optimisation loop. Our proposed optimisation loop is completely modular, meaning it can be tuned and even expanded in the future.

The optimisation loop utilises a technique well known in the literature called SA. However, SA is a multi-parameter technique that is very number-sensitive, and even small changes in the input variables can have a large impact in the final solution. In addition to the optimisation loop, we will introduce a detailed analysis of how different SA configurations affect the loop from a mathematical standpoint and present a number of different viable configurations and where they are most useful.

Finally, a lot of effort was put in re-creating an algorithm that, given a floorplan without its Sequence Pair representation, would reproduce that representation. Such a process was described even in the earliest Sequence Pair paper [10]. But as we will show, that and other

approaches described in [8], [11], and [9], through a series of simple practical examples have been deemed ineffective.

1.2 Thesis' Structure

Our work will be divided into 4 parts. In chapter 2, we will present a brief history of chip design and Electronic Design Automation (EDA) along with how floorplanning fits into the ecosystem of modern chip design.

We will exhibit a brief taxonomy of floorplanning algorithms with their main advantages and disadvantages. After that in chapter 3, we will introduce the Sequence Pair algorithm and take a deep dive into the advances the research literature has offered over the years not only in algorithmic improvements but also in novel areas. Then in chapter 4, we will provide greater detail into our research contribution in the floorplanning areas of:

- New initial Sequence Pair solutions,
- Novel Sequence Pair moves (pure Sequence Pair moves and intermediate optimisation strategies),
- Simulated Annealing optimisation and ,
- Placement to Sequence Pair strategies.

Finally in chapter 5, we will discuss how we evaluated all our newly implemented methodologies and optimisation techniques.

Chapter 2

Theoretical Background

In 1965, Gordon Moore announced his prediction about the scaling of the integrated circuits. Three years later he will be one of the three founders of Intel, one of the biggest high-tech companies. His predicion still stands to this day.

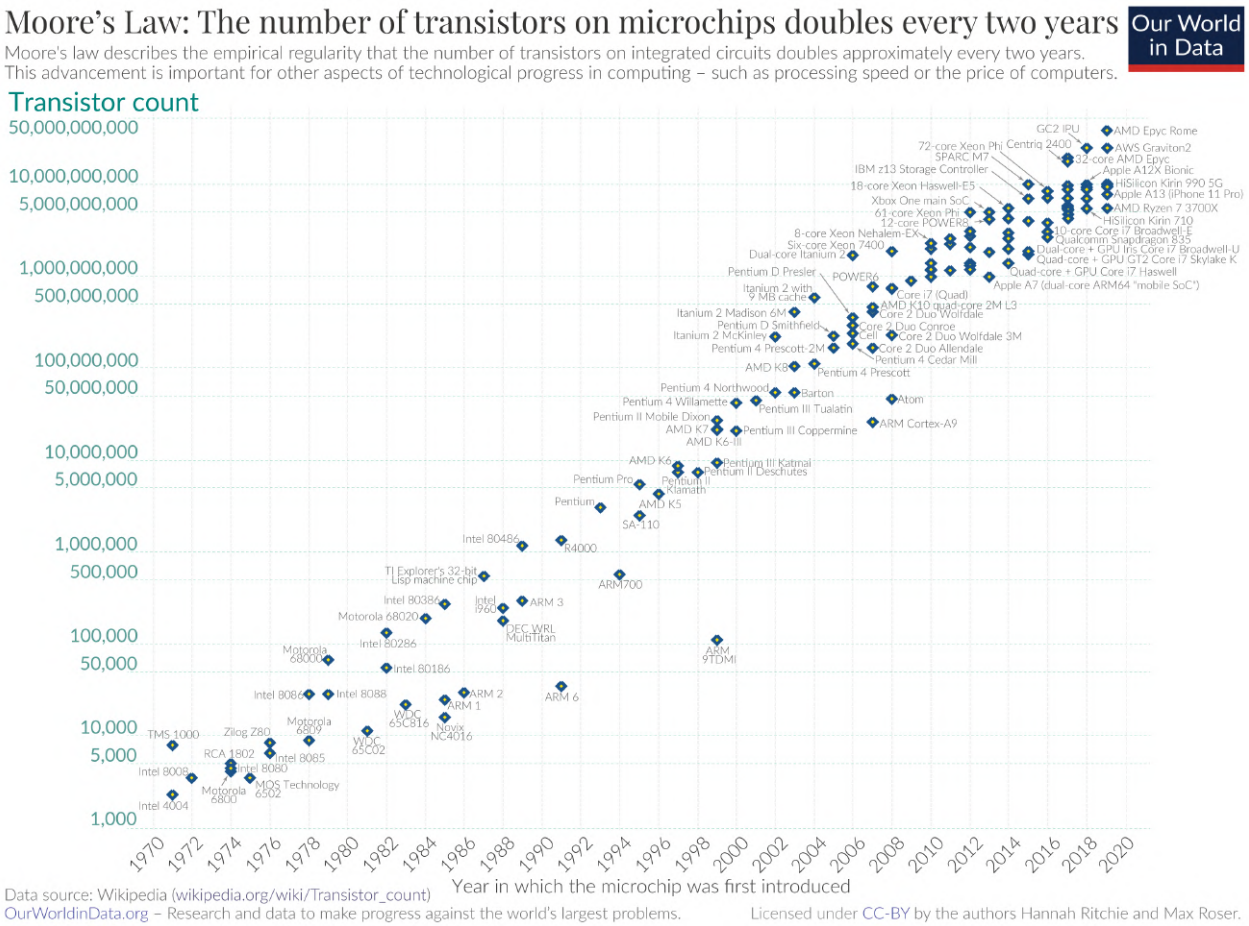


Figure 2.1: Moore's Law until today. [1]

2.1 Chip Design and EDA History

The history of chip design is a captivating journey through technological evolution. It all began in the mid-20th century when the earliest integrated circuits paved the way for groundbreaking innovations. In 1958, Jack Kilby at Texas Instruments and Robert Noyce at Fairchild Semiconductor independently invented the first integrated circuits, marking the birth of chip design. These early chips contained a modest number of transistors, but their significance was monumental. As time progressed, chip designers relentlessly pushed the boundaries of miniaturization, packing more and more transistors onto silicon wafers. This relentless pursuit led to the birth of microprocessors in the early 1970s, igniting the personal computer revolution. The 21st century brought forth the era of system-on-chip (SoC) designs, where entire systems were integrated onto a single chip, powering the modern era of smartphones, IoT devices, and more. Chip design history is a testament to human ingenuity and the relentless quest for smaller, faster, and more powerful electronic devices.

2.2 EDA Flow

The EDA flow is the backbone of modern chip design, orchestrating a complex symphony of processes that transform abstract concepts into physical silicon reality. It begins with high-level design and functional specification, where designers articulate the intended behavior and functionality of a chip. As the flow progresses, logic synthesis translates this high-level description into a netlist of gates and interconnections. The next stage, placement and routing, determines where each gate will physically reside on the chip and how they will be interconnected. This is followed by physical verification to ensure that the design adheres to manufacturing rules and standards. The process is further refined through simulations, verifying the chip's performance, functionality, and power consumption. Once all aspects are finely tuned and validated, the final design is sent to fabrication. The EDA flow is a meticulously orchestrated sequence of steps, ensuring that chip designs meet the ever-increasing demands of performance, power efficiency, and miniaturization in today's technology-driven world.

The EDA flow has evolved massively over the years as it can be seen in Figure 2.2.

Time period	Circuit and physical design process advancements
1950–1965	Manual design only
1965–1975	First software to generate the tapes for the layout photoplotter by copying digital recordings of manually drawn components. Layout editors, e.g., place and route tools, first developed for PCBs
1975–1985	More advanced tools for ICs and PCBs, with more sophisticated algorithms. First annealing algorithms
1985–1990	First performance-driven tools and parallel optimization algorithms for layout; better understanding of the underlying theory (graph theory, solution complexity, etc.)
1990–2000	First over-the-cell routing, first 3D and multilayer placement and routing techniques developed. Automated circuit synthesis and routability-oriented design become dominant. Start of parallelizing workloads. Emergence of physical synthesis. Quadratic algorithms. Layout hotspot detection
2000–2010	Design for manufacturability (DFM). Design for test (DFT). Optical proximity correction (OPC) and other techniques emerge at the design-manufacturing interface. Increased reusability of blocks, including intellectual property (IP) blocks. Low-power methods such as voltage islands and clock gating
2010–2020	Built-in self-test (BIST). Emergence of layout post-processing and migration-mitigating design methodologies, monolithic 3D design, and more-than-Moore integration. Renewal of procedural automation (generator) methods. Hyperoptimization
2020–now	Machine learning (ML) boosting of algorithms. Loss of orthogonalizations and abstractions. Fusions. Co-optimizations

Figure 2.2: Timeline of EDA progress with respect to circuit and physical design [2].

The major steps of the EDA flow are presented through Figure 2.3.

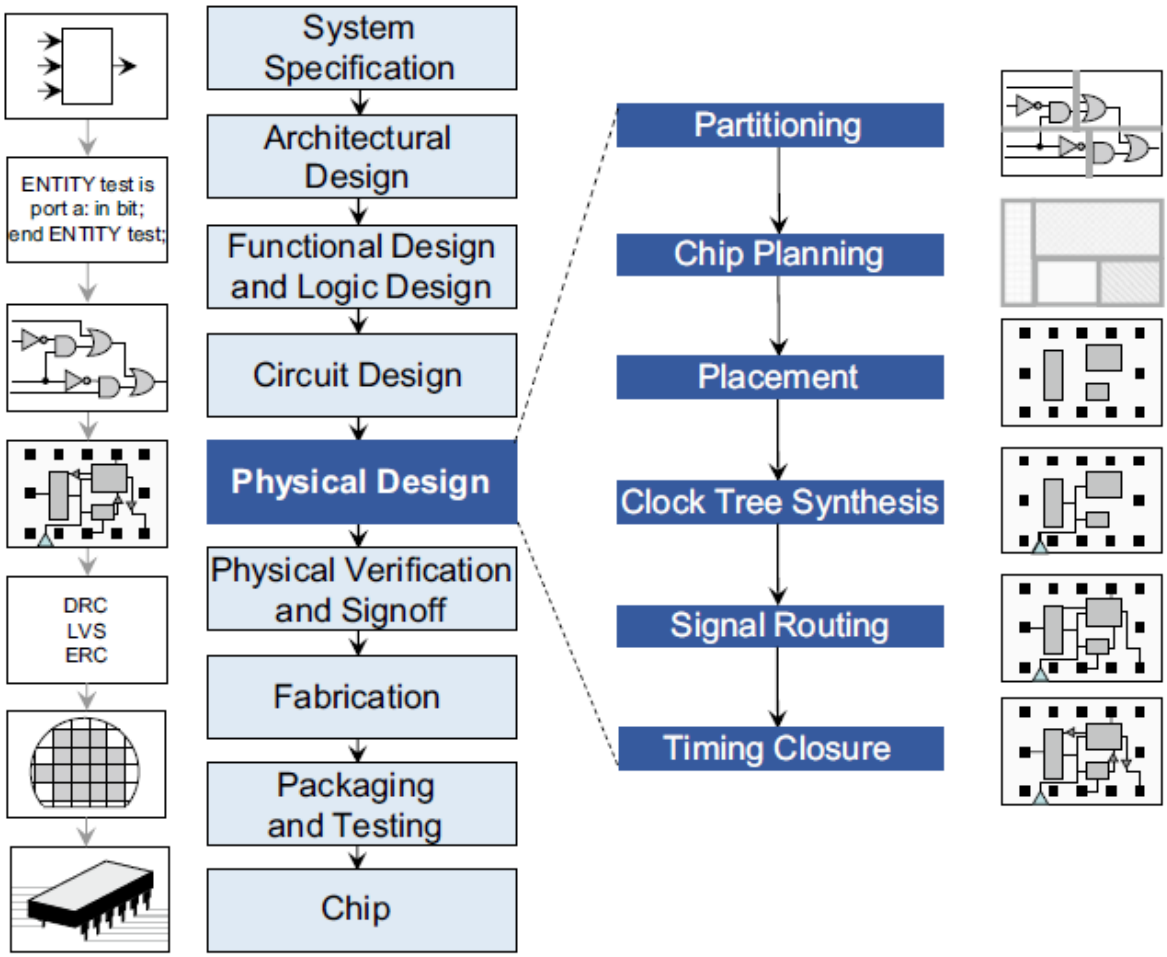


Figure 2.3: Major steps in VLSI circuit Design [2].

We will focus on the physical design flow as it can be seen in Figure 2.4. Here we can see the the different types of constraints that typical floorplanning algorithms usually deal with.

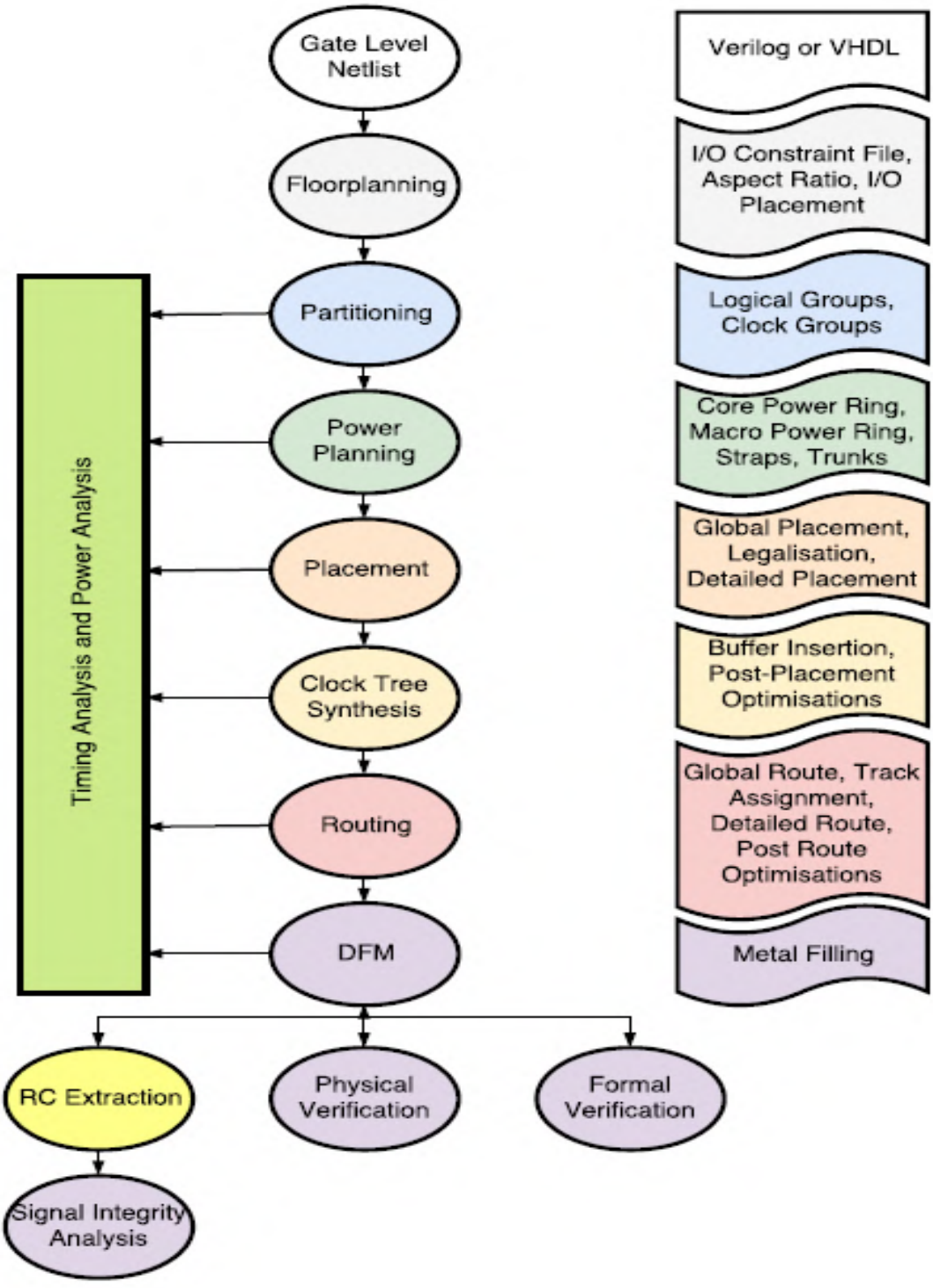


Figure 2.4: Physical Design Flow [3].

2.3 Floorplanning Background

Floorplanning stands as a crucial stage within the VLSI design process, holding a critical responsibility in shaping the physical arrangement of intricate integrated circuits.

Floorplanning plays a pivotal role in defining the layout's structure, specifically the positions of modules within the chip. It relies on the circuit's interconnection needs and area estimations to create this topology. A floorplan serves as a valuable reference during the detailed design of functional modules or blocks, especially when certain modules on the chip have undefined aspect ratios and pin positions [12].

To execute this phase efficiently, a multitude of input factors come into play. At the forefront is the design netlist, functioning as the circuit's architectural blueprint, detailing the logical links between its elements. Furthermore, it is imperative to thoroughly assess factors such as space demands, power prerequisites, and timing limitations. This meticulous evaluation ensures that the ultimate floorplan aligns seamlessly with the desired performance and power efficiency objectives [13].

Floorplanning is incredibly complex as can be seen in Figure 2.5. Different algorithms have vastly different numbers of possible combinations, albeit this does not disallow possible duplicate solutions.

For our purposes, we can categorize floorplans into 2 main types: slicing floorplans and non-slicing floorplans. Slicing floorplans are generated by making repetitive horizontal or vertical cuts, while non-slicing floorplans are not amenable to such cuts and can even be separated into even smaller sub-categories like mosaic floorplans, which we will not analyse firther. It's essential to maintain the specified dimensions of each hard module, and no module is allowed to be rotated. If a module is rotated, its width and height must be swapped accordingly.

However, such a description, is far to narrow and is not able to fit all necessary needs of modern design. For that reason a lot of more general floorplanning algorithms have been developed over the years. Some of their traits as well as their complexities and representations can be seen in Figure 2.7, Figure 2.8, and Figure 2.9.

Nuber of blocks	Combinations of O-Tree	Combinations of Slicing floorplan	Combinations of Mosaic floorplan	Combinations of Sequence Pairs
1	1	1	1	1
2	2	2	2	2
3	5	6	6	6
4	14	22	22	24
5	42	90	92	120
6	132	394	422	720
7	429	1,806	2,074	5,040
8	1,430	8,558	10,754	40,320
9	4,862	41,586	58,202	362,880
10	16,796	206,098	326,240	3,628,800
11	58,786	1,037,718	1,882,690	39,916,800
12	208,012	5,293,446	11,140,560	479,001,600
13	742,900	27,297,738	67,329,992	6,227,020,800
14	2,674,440	142,078,746	414,499,438	87,178,291,200
15	9,694,845	745,387,038	2,593,341,586	1,307,674,368,000
16	35,357,670	3,937,603,038	16,458,756,586	20,922,789,888,000
17	129,644,790	20,927,156,706	105,791,986,682	355,687,428,096,000

Figure 2.5: Exact Number of combinations of different floorplan representations.

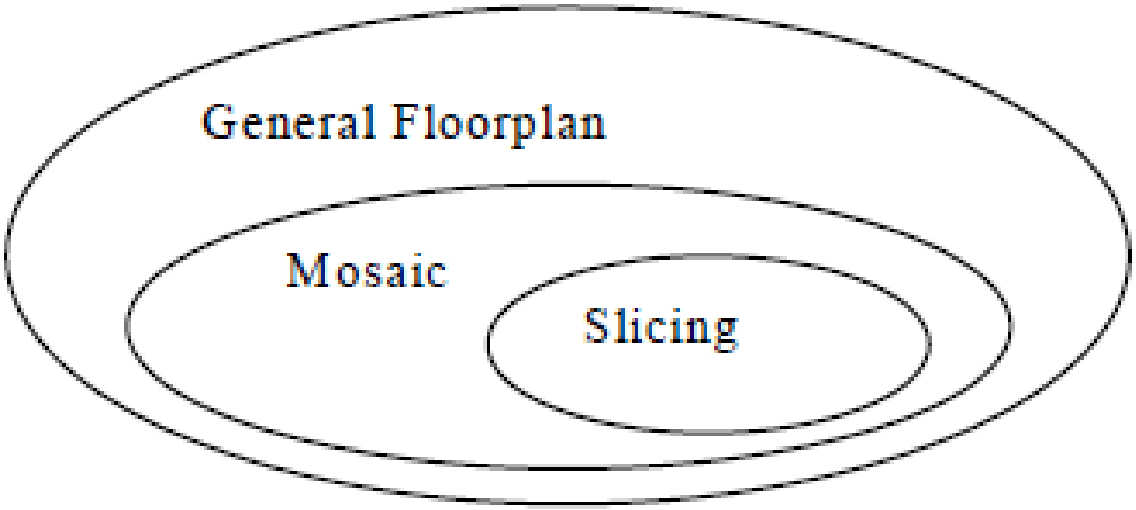
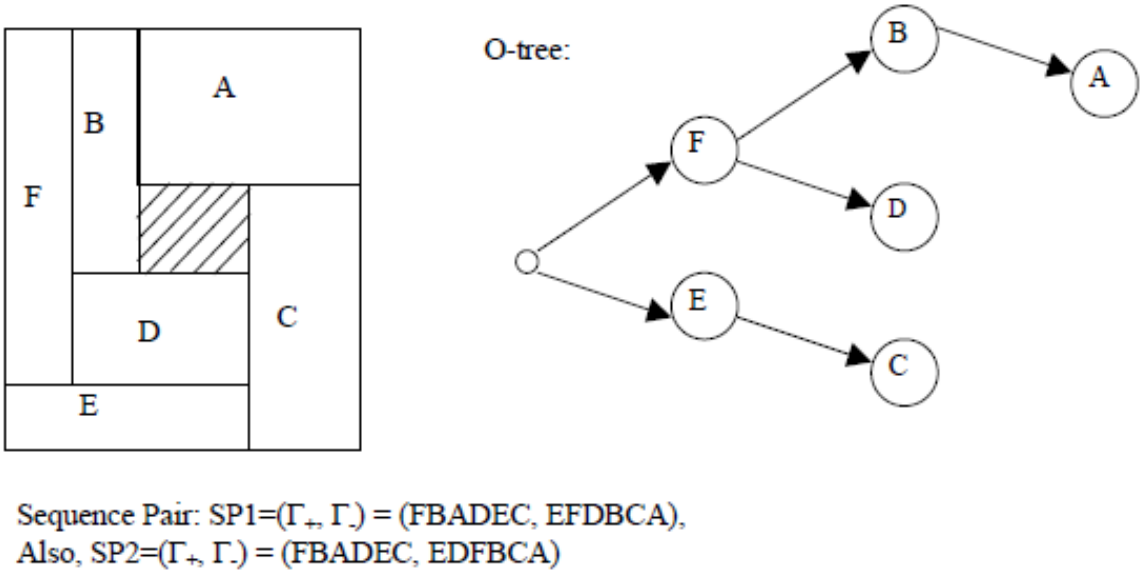
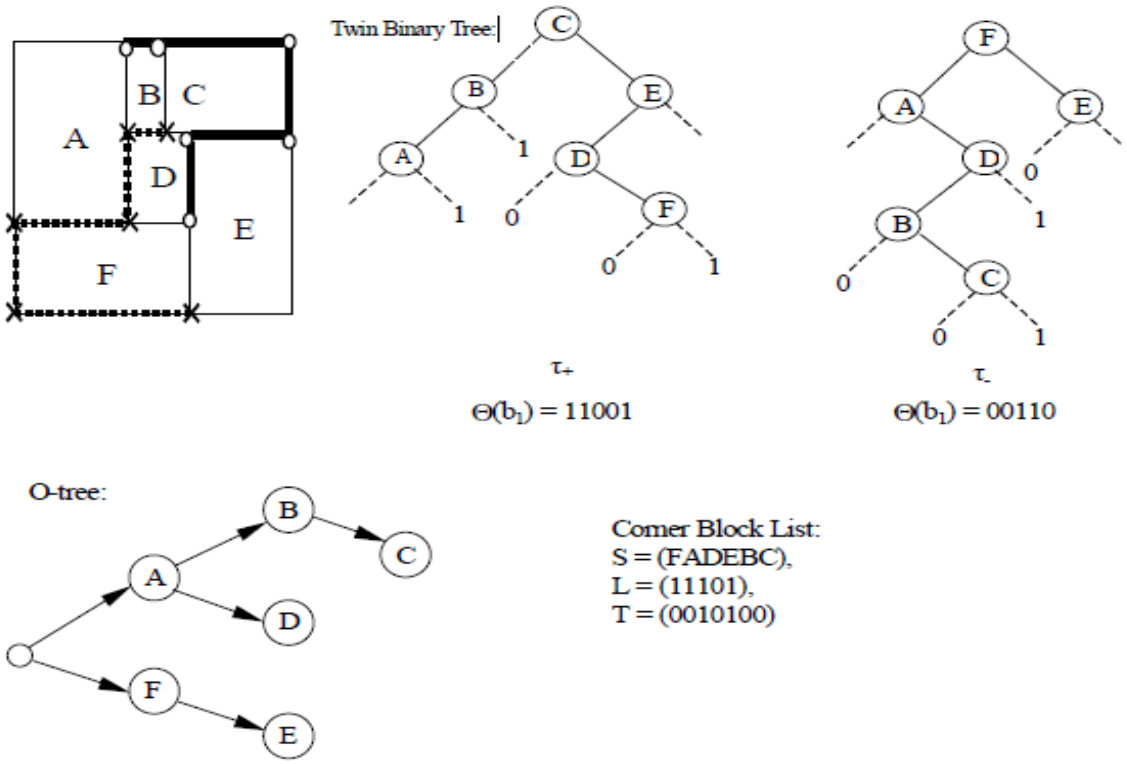


Figure 2.6: Different floorplan categories [4].



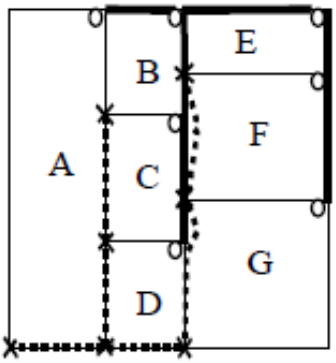
Sequence Pair: SP1=(Γ+, Γ-) = (FBADEC, EFDBCA),
Also, SP2=(Γ+, Γ-) = (FBADEC, EDFBCA)

Figure 2.7: General floorplan representations [4].

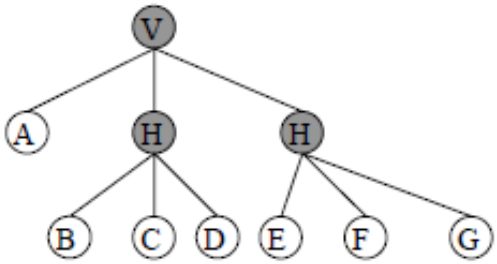


Sequence Pair: $SP_1 = (\Gamma_+, \Gamma_-) = (ABCDFE, FADEBC)$. Also $SP_2 = (\Gamma_+, \Gamma_-) = (ABCDFE, FADBEC)$ refers to the same floorplanning.

Figure 2.8: Mosaic floorplan representations [4].

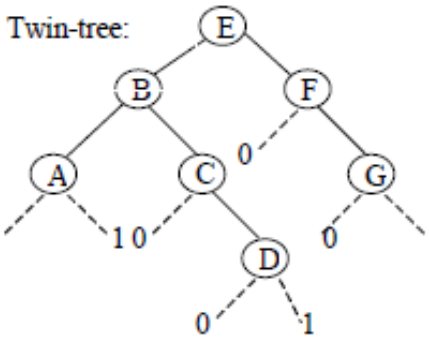


Slicing-Ordered tree:



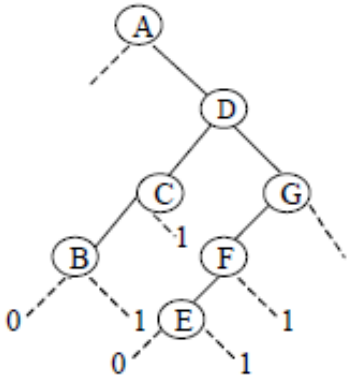
(A(BCD)(EFG))

Twin-tree:



$\Theta(b_1) = 100100$

τ_+



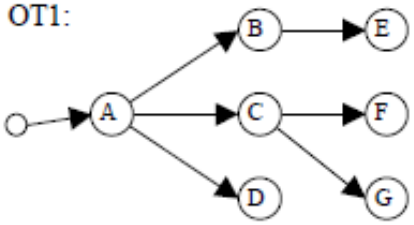
$\Theta(b_2) = 011011$

τ_-

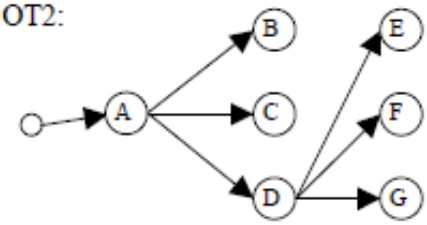
In-order sequence of τ_+ : I = ABCDEFG

Two different O-tree representations:

OT1:



OT2:



One SP representation of this floorplan is $SP_1=(\Gamma_+, \Gamma_-)$ = (ABECFDG, ADCGBFE).
Also the sequence pair $SP_2=(\Gamma_+, \Gamma_-)$ = (ABCDEFG, ADCGBFE) and
$SP_3=(\Gamma_+, \Gamma_-)$ = (ABCDEFG, ADCBGFE) represent the same floorplan.

CB: S = (ADCBGFE), L = (100100), T=(00011000)

Figure 2.9: Slicing floorplan representations [4].

Chapter 3

Sequence Pair Literature

In this chapter, we will present the Sequence Pair floorplanning algorithm, from its inception in 1995 to novel ideas that have, over the years, improved both its efficiency and expanded its applications.

3.1 Original Work

The Sequence Pair algorithm was first introduced by Murata et al. [14] and what it aimed to achieve at the time, was to solve the rectangle packing problem for the rectangular blocks of an IC. An important aspect of Sequence Pair, is the fact that the resulting packing is legal, in the sense that the floorplan blocks do not overlap with one another. To do that, the authors formulated, by using an identifier (e.g module name) for each floorplan block, two sequences of these identifiers, named the Positive and the Negative sequence respectively [1]. The core idea of the algorithm is, that for 2 blocks A and B the relative position of their identifier

[1] In Sequence Pair literature it is common notation that the first sequence is the Positive one. We will be using that notation and assuming that the first presented sequence is the Positive Sequence and the second is the Negative Sequence.

Table 3.1: Floorplan Representations, Computational Complexity, Search Space, and Flexibility [7].

Representation	Computational Complexity	Search Space	Flexibility
Polish Expression	$O(n)$	$O(n!2^{3n}/n^{1.5})$	Slicing
Corner Block List	$O(n)$	$O(n!2^{3n})$	Mosaic
Twin Binary Sequence	$O(n)$	$O(n!2^{3n}/n^{1.5})$	Mosaic
O-tree	$O(n)$	$O(n!2^{2n}/n^{1.5})$	Compacted
B*-tree	$O(n)$	$O(n!2^{2n}/n^{1.5})$	Compacted
BSG	$O(n^2)$	$O(n!C(n^2,n))$	General
Transitive Closure Graph	$O(n^2)$	$O(n!)^2$	General
Corner Sequence	$O(n)$	$O(n!)^2$	General
TCG-S	$O(n \log n)$	$O(n!)^2$	General

determined their relative positions in the final placement. Let us imagine a floorplan of these 2 blocks. Their relative positions depend on the blocks names (A and B), based on the following convention:

1. If A is after B in both sequences, (Sequence Pair is (BA, BA)) $\Rightarrow$ A is Right-of B.
2. If A is before B in both sequences, (Sequence Pair is (AB, AB)) $\Rightarrow$ A is Left-of B.
3. If A is after B in the Positive Sequence and A is before B in the Negative Sequence, (Sequence Pair is (BA, AB)) $\Rightarrow$ A is Below-of B.
4. If A is before B in the Positive Sequence and A is after B in the Negative Sequence, (Sequence Pair is (AB, BA)) $\Rightarrow$ A is Above-of B.

As is apparent from the above, this covers all possible combinations.

One of the biggest advantages that the Sequence Pair representation possesses against other floorplanning algorithms is its inherent flexibility. This is due to the fact that, through a small set of permutations it can handle a much larger number and a greater variety of floorplans. Hence, the main problem that Sequence Pair optimisation entails is to find and evaluate, as fast as possible, the best set of permutations, in order to obtain a floorplan as close to optimal.

This flexibility is an incredible characteristic, albeit it comes with a few side effects. A feature of Sequence Pair described in the original paper, is the fact that multiple Sequence Pair can produce identical floorplans. Such Sequence Pairs, when evaluated, will indeed produce identical floorplans, but the underlying HCG and VCG will be different.

This means that, when 2 Sequence Pairs that produce identical floorplans are subjected to a series of permutations, with the goal of optimising some cost function, as we will see in section 4.5, the final floorplan will most likely vary significantly. Another artifact of the Sequence Pair representation, is that the opposite not true. This means that given a Sequence Pair, evaluating it to a floorplan, as described in subsection 3.1.1, will only ever yield one result. This is logical due to the usage of the Longest Path algorithm which always produces the same result, during the Sequence Pair evaluation.

3.1.1 Main Evaluation Algorithm

As mentioned above the evaluation is one of the most important steps in Sequence Pair floorplanning. This is because in order to judge how good a permutation is, we always have to evaluate the new Sequence Pair it produces. And in a optimisation loop, where we are expected to tried hundreds of thousands of permutations, evaluation presents an incredible bottleneck.

If we expand the above example to a floorplan with multiple blocks we can see a grid of relations arising. For example, in [5], we can see how a Relation Table can be forged, given a

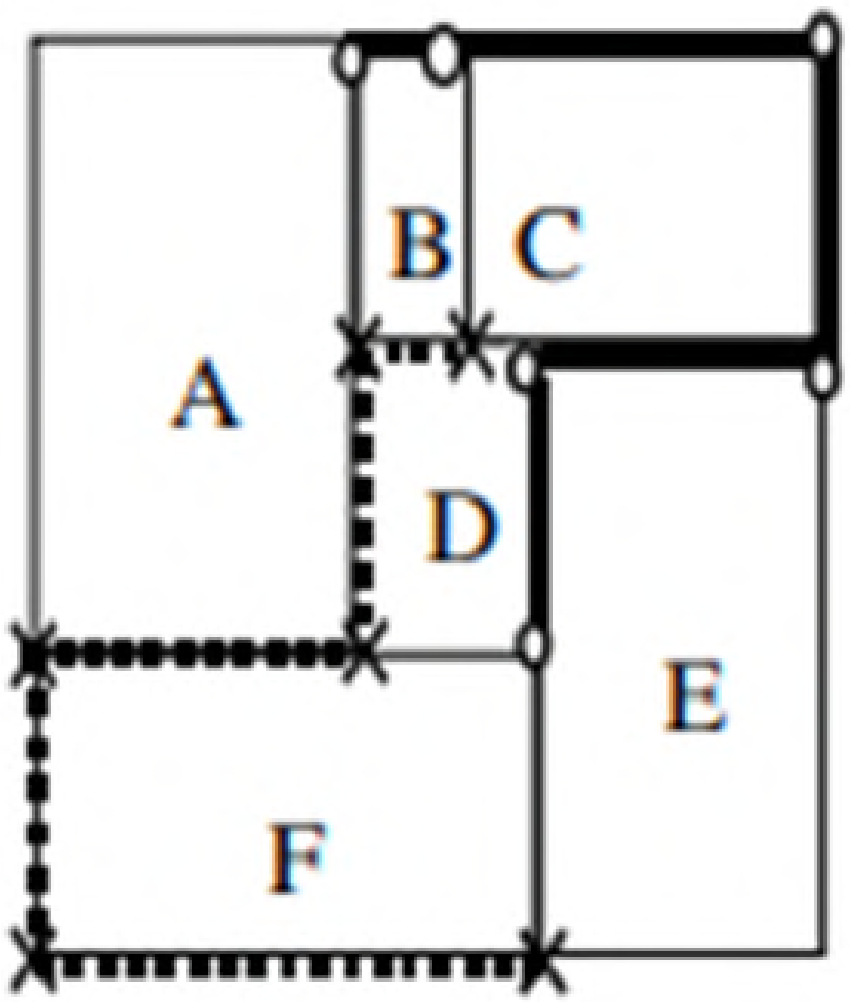


Figure 3.1: Both (ABCDFE, FADEBC) and (ABCDFE, FADBEC) produce the same floorplan.

Sequence Pair. The Relation Table denotes which relation the block has with the other blocks. In the following example, blocks {2, 3, 4, 5, 6, 7, 8} are Below-Of block 1.

However, up to this point we have only spoken of how the relations are formed and not about how, given a Sequence Pair, we can evaluate it and extract information about the packing it represents. Utilising the information in the Relation Table above, the authors construct 2 directed and weighted constraint graphs called HCG and VCG, respectively. These graphs are constructed in the following manner:

1. V: source s, sink t, and vertices labeled with the floorplan block names.

2. E: (s, x) and (x, t) for each module x, and if (x, x') and only if block x is Left-Of block x'.

Table 3.2: Relation Table for the Sequence Pair (17452638, 84725361) [5].

Block	Right-Of	Left-Of	Above-Of	Below-Of
1	∅	∅	∅	{2, 3, 4, 5, 6, 7, 8}
2	{3, 6}	{4, 7}	{1, 5}	{8}
3	∅	{2, 4, 5, 7}	{1, 6}	{8}
4	{2, 3, 5, 6}	∅	{1, 7}	{8}
5	{3, 6}	{4, 7}	{1}	{2, 8}
6	∅	{2, 4, 5, 7}	{1}	{3, 8}
7	{2, 3, 5, 6}	∅	{1}	{4, 8}
8	∅	∅	{1, 2, 3, 4, 5, 6, 7}	∅

3. Vertex-weight: zero for s and t, width of floorplan block for the other vertices.

Similarly, the vertical-constraint graph is constructed using Below-Of constraint and the height of each floorplan block. [2] After the 2 graphs have been constructed, to extract the coordinates of each block in the resulting packing, what we have to do is perform a Longest Path algorithm, with the X and Y coordinate originating from the HCG and the VCG, respectively. This has a complexity of $O(V + E)$, since the graphs are directed.

The usage of the Longest Path algorithm is a necessary one. Since we are using the width/height of each block as weights in the 2 graphs, the Longest Path algorithm ensures that there are no overlaps between the floorplan blocks.

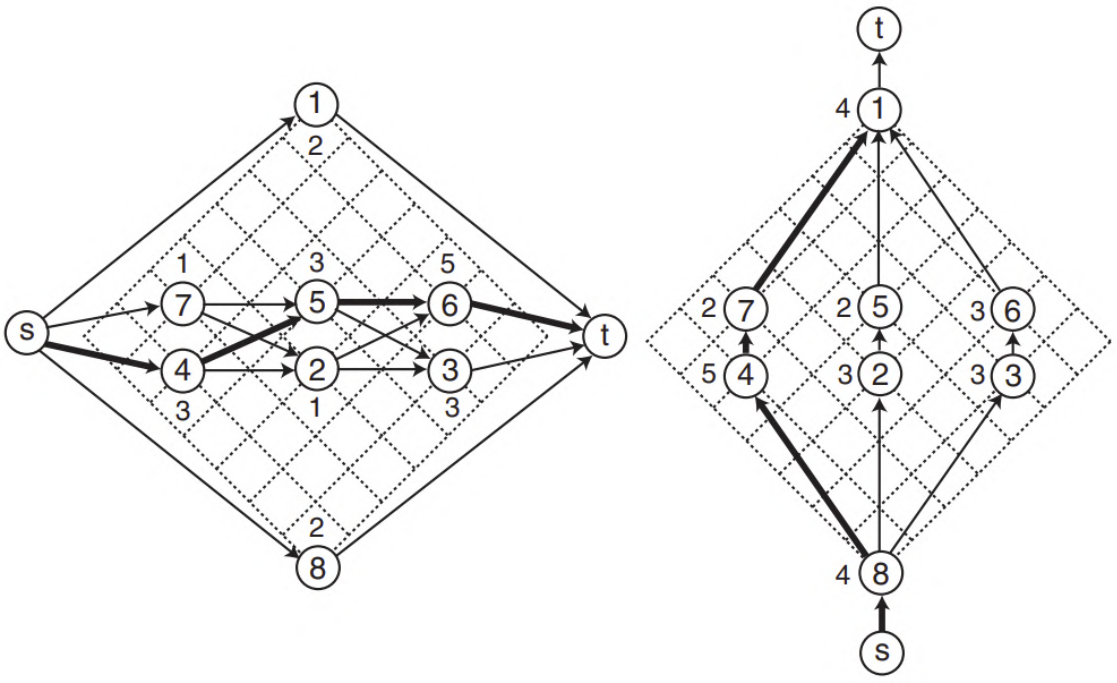


Figure 3.2: (a) HCG with longest s-t path length 11, (b) VCG with longest s-t path length 15. The numbers next to each node denotes its width (in HCG) or height (in VCG) [5].

[2] The dimensions (width, height) of the blocks 1 to 8 are {(2,4), (1,3), (3,3), (3,5), (3,2), (5,3), (1,2), (2,4)}.

3.1.2 Original Sequence Pair Moves

Now that we can evaluate any given Sequence Pair, we can begin to optimize a given packing by performing permutations in its Sequence Pair. In the original work by Murata et al. the following moves were defined:

1. Rotate a block by 90 degrees.
2. Swap a pair of blocks in both sequences.
3. Swap a pair of blocks in one sequence.

While the moves seem simple enough at first sight, their actual effect on the resulting floorplan, has, to our knowledge, never been explored in the literature. This will be explored in finer detail in subsection 4.1.1.

3.1.3 Original Placement to Sequence Pair

Furthermore, the original paper [14] and educational works that reference it such as [12], further provide a way to, given a placement, extract its corresponding Sequence Pair, via a process called Gridding, described below. For each block (denoted as ”x”), we use a dot, to draw lines. We place the dot initially at the upper-right corner of block x and make it move upwards. The dot’s movement alternates between right and up until it reaches the upper-right corner of the block. While moving, the dot should not cross the following boundaries:

1. Boundaries of other blocks.
2. Lines that have been drawn previously.
3. The boundary of the entire chip.

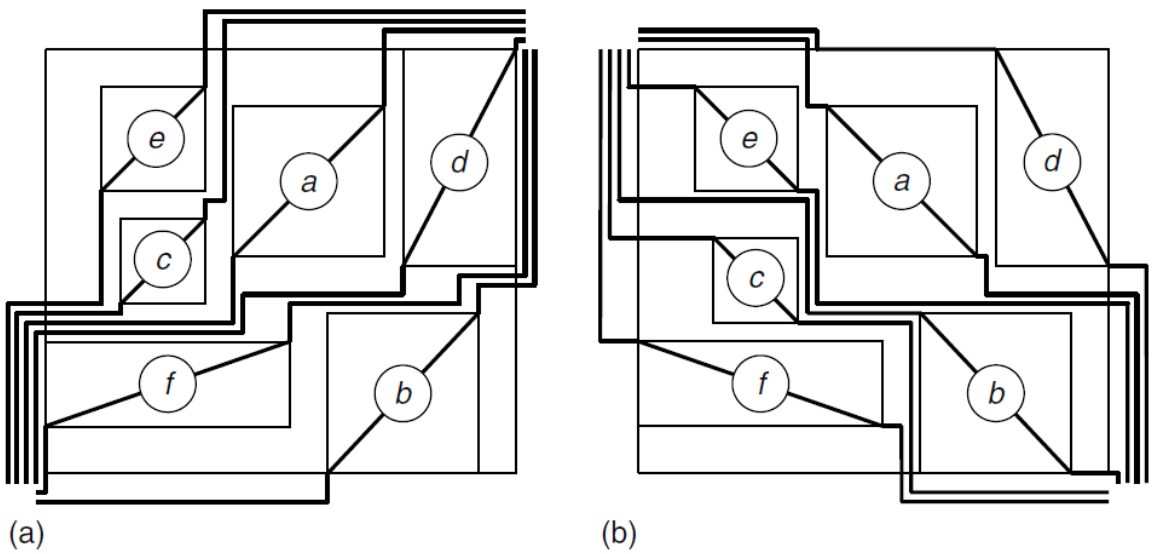


Figure 3.3: Example of Positive and Negative Loci created through the Gridding process [5].

The line that is created by this process is known as the ”up-right step-line” of block x. Similarly, we also draw a ”down-left step-line.” When we combine these two step-lines with the connecting diagonal line of block x, we get what is called the ”positive step-line” of block x. It's worth noting that it's always possible to create such a positive step-line for any given block, and they are commonly referenced by the names of the respective blocks.

The authors provide neither further intuition into this process nor any indication to what the process' complexity looks like. This allegedly is enough to create a Sequence Pair that represents the given placement, a claim that we will examine in section 4.6.

3.2 Upgrades to the original approach

In this section, we will tackle how the research community has innovated on the original Sequence Pair algorithm over the years, both in an algorithmic manner, to improve its overall complexity but also in finding new applications. We will see how different constraints have been imposed to the algorithm, so that it can handle more realistic problems, a few exotic applications of the algorithm and how the literature proposals on how to solve the Placement to Sequence Pair problem.

3.2.1 Execution time upgrades

As described in subsection 3.1.1 evaluation is the biggest bottleneck of Sequence Pair optimization and the literature discovered that early. In 2000 Tang et al. [15], developed an evaluation algorithm that converts the evaluation problem from a graph-theoretical one, to that of Longest Common Subsequence (LCS). This reduces overall evaluation complexity, from $O(V + E)$ to $O(n \log n)$. However they did not stop there, and in a subsequent work [16], they further improved the LCS methodology by using a very efficient Priority Queue developed by D. B. Johnson [17]. The efficient priority queue reduced the total evaluation complexity to $O(n \log \log n)$. It is a modified version of this algorithm, that we implemented for the purposes of this work. This version will be further analyzed in subsection 4.1.2.

However impressive was the work by Tang et al., the best overall complexity was achieved more recently. In 2013, Andrzej Kozik in [18], utilising dynamic programming, developed an algorithm transformed the LCS problem and used an improved data structure, such that, although still has a worst case complexity of $O(n^2)$, exhibits linear time behaviour in the average case, which significantly reduces the evaluation time.

3.2.2 Novel Sequence Pair expansion ideas

The literature was fast, to also improve on the capabilities of the original methodology. So far, we only discussed about how Sequence Pair can manipulate blocks of predefined

dimensions, hard-blocks as they are called by the scientific literature. Be that as it may, as discussed earlier, floorplanning, being one of the first steps in the VLSI design process, offers a lot of freedom for architectural exploration to engineers. More often than not, at this design stage floorplan blocks will have defined their final area and not their exact dimensions. Some may have one dimension defined. Such blocks are called soft-blocks. Other blocks may be required to have a specified location from the outset and even a lot more further parameters defined and thus, the literature adapted.

Further constraints considerations

Murata et al. [19] expanded the Sequence Pair representation to also include soft and pre-placed placed blocks by utilising Vaidya's algorithm.

As the performance of the design is not only dependent on area and wire length, timing-driven approaches for Sequence Pair soon followed. In 1999, Huang et al. [20] first introduced a timing-driven approach. In their work the authors used a multi-parameter cost function that incorporated total area, wire length as well as timing slack. Since the the numerical values of this metrics vary significantly, the authors used different weights for each one. For the timing slack in particular, they created weights for the designs' nets based on the timing slack. Considering that, the significance they attributed to each parameter is not transparent, because only some of their weight choices are mentioned. Nonetheless, their work showed significant timing improvements for the tested designs ($\sim$33.6%) with only a small increase in dead space ($\sim$5.3%). What must be mentioned, is that there was a non-negligible increase in execution time, due to the fact that timing operations are computationally expensive. Their execution time penalty came as high as 87.3%, compared to the non timing-driven approach.

Another important aspect of designing a chip are its thermal properties. This is explored further by Okada et al. [21], in a work solely based around Sequence Pair. Their approach to the problem is focused around the principle that the most energy dense blocks should be spaced apart. This is done by applying coefficients ("heat quantities") to constrain the block positions of a Sequence Pair placement based on its bottom-left and the top-right placements and averaging them to produce the final result. Another interesting work by Cong et al. [22], that although not strictly uses the Sequence Pair representation, but the TCG floorplanning scheme [23], that is very similar in concept and optimisation move repertoire. In this work, the authors formulate their solution around a new set of problems. The thermal problems that arise through heat flow in chips that have multiple layers.

Sequence Pair for 3D ICs

As mentioned section 3.2.2, Sequence Pair related research work has already been extended to the more exotic 3-dimensional space by [22]. In an incredible leap, even earlier than Cong et al., Yamazaki et al. [6] showcased a methodology to obtain a minimum volume

block packing (blocks with width, height and depth properties) by expanding Sequence Pair. What is incredible from this work is that creating a Sequence Triplet was not sufficient to encode such packings, because it resulted in non feasible solutions. Hence they devised, the so called Sequence Quintuple to help encapsulate all relations.

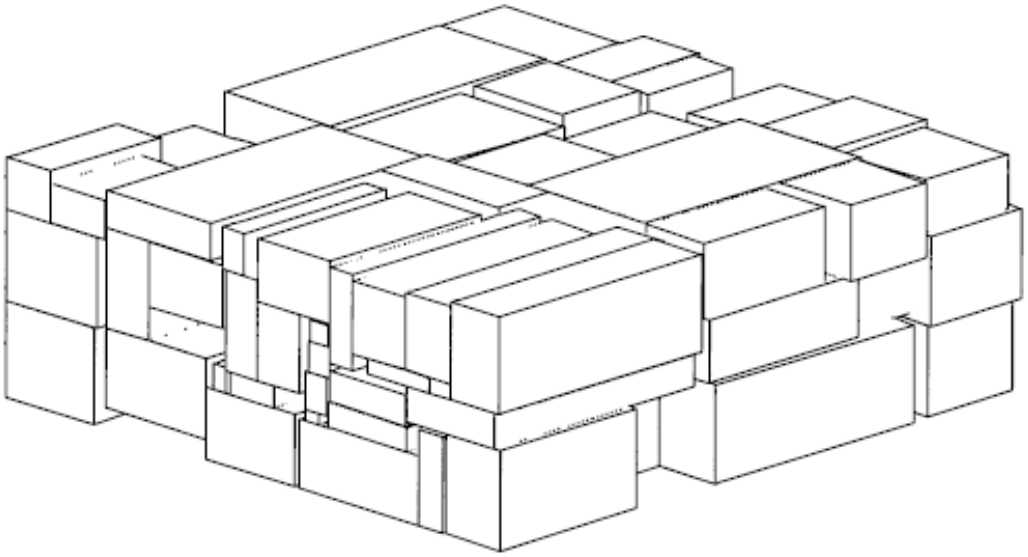


Figure 3.4: A packing of 100 3D blocks under Sequence Quintuple [6].

Since 3-dimensional and 2.5-dimensional design is gaining traction more recent Sequence Pair work has taken place recently aiming to solve some of the latest problems. Such chips require unique silicon constructions to communicated between different tiers, the most common of which is the Through Silicon Via (TSV). TSVs although necessary, are incredibly expensive in terms of area. In one such work 2022 Prakash et al.[24], presented a methodology that after partitioning the design into tiers utilises a Particle Swarm Optimisation (PSO) algorithm to minimise the number of TSVs along with the total wire length.

Exotic Sequence Pair Applications

Now we will present some of the works in the latest Sequence Pair literature. In one of the most impressive works on the matter, by Kahng et al. [2] integrate a their Macro Placer which internally uses the Sequence Pair representation. This macro placer is integrated to the OpenROAD project [25]. What is innovative about their work is the fact that they attempt, to mimic human placement behaviour by extracting information from the RTL of the given designs, with incredible results even against modern commercial EDA placement tools.

In [26], Sengupta et al. address another modern problem, that of excessive power consumption. This work focuses the Voltage Island design technique. This method involves utilizing multiple supply voltages on the same chip, with different blocks assigned to specific supply voltages. Due to implementation constraints, blocks with similar supply voltages need

to be placed adjacent to each other, creating what are known as ”islands.” Traditional floorplanning tools assume a single supply voltage for the entire System on Chip (SoC), requiring additional design steps to incorporate voltage islands. They introduce a novel floorplanning algorithm based on the Sequence Pair representation, which allows us to efficiently organize blocks as islands. Their approach considers the potential supply voltage choices for each block and simultaneously optimizes for both power and chip area. Their floorplanner streamlines the tasks of assigning blocks to different supply voltages and placing them within the chip.

In perhaps its most bizarre application yet, [27] use the Sequence Pair representation the layout design for Robotic Cellular systems. Via the Sequence Pair they determine the positions of components, a robot and a table on which components to be placed.

3.2.3 Expanding Placement to Sequence Pair

The Placement to Sequence Pair problem, did not remain stagnant either over the years. A number of works [8], [11], and [9], the first 2 by Huo et al. and the last by Egeblad were published. However, each of these methods contains massive flaws that will be examined in section 4.6.

Chapter 4

Our Contribution

In this chapter we will discuss our contribution to the Sequence Pair field. We will explain what Sequence Pair evaluation methods we implemented, as well as give insight into how the original moves operate behind the scenes. Moreover, we will present the innovative optimisation moves we introduced and showcase the effects that different starting positions have in the optimisation process. Then, we will have a slight detour to understand how SA works and introduce our novel optimisation loop. Lastly, we will investigate the field of Placement to Sequence Pair. All of the work described in this chapter was integrated in CASLab's [28] EDA tool, called ASP, as part of its floorplanning capabilities, and was entirely developed in C.

4.1 Sequence Pair Methodology

4.1.1 Description of main moves

As previously stated Sequence Pair is very minimalistic in its move-set. The original paper only describes what the moves are and not what they actually do. Here we will analyse the following 3 moves and their effects in depth.

1. Rotate a block by 90 degrees.

2. Swap a pair of blocks in both sequences.

3. Swap a pair of blocks in one sequence.

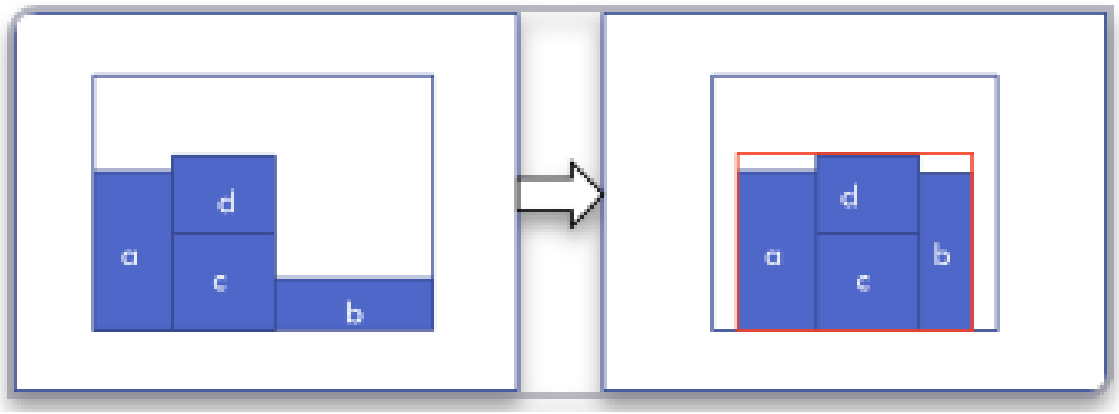


Figure 4.1: Rotating block b.

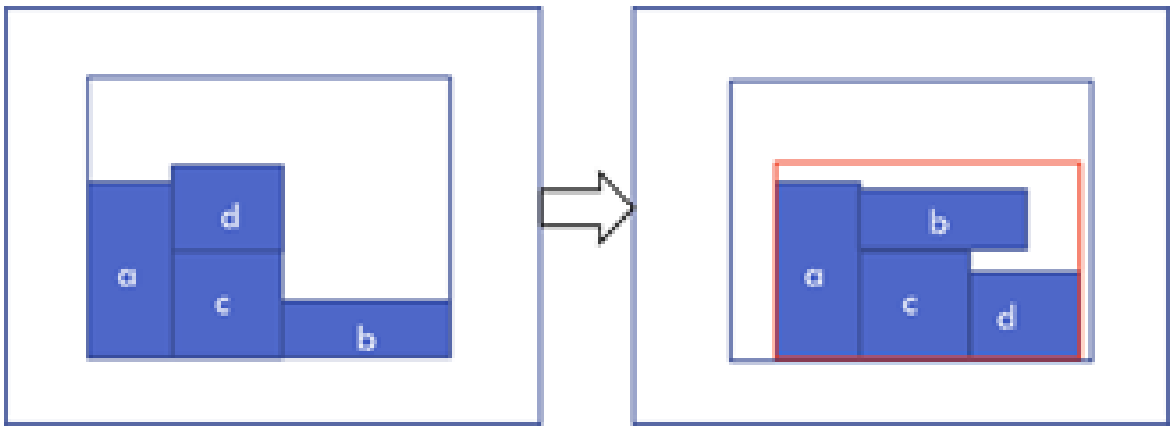


Figure 4.2: Swap blocks b and d in both sequences.

Rotating is the simplest of the moves and is equivalent to swapping the blocks width and height. Swapping a block pair in both sequences is equivalent to swapping their positions in the final packing. However, swapping a pair of blocks in one sequence only is a more peculiar permutation. This is due to the fact that performing such a swap can affect an arbitrary amount of blocks depending on:

Table 4.1: Relation Table for the Sequence Pair (ecadfb, dfbace).

Block	Right-Of	Left-Of	Above-Of	Below-Of
a	∅	∅	{c, e}	{b, d, f}
b	∅	{d, f}	{a, c, e}	∅
c	∅	∅	{e}	{a, b, d, f}
d	{b, f}	∅	{a, c, e}	∅
e	∅	∅	∅	{a, b, c, d, f}
f	{b}	{d}	{a, c, e}	∅

Table 4.2: Relation Table for the Sequence Pair (efadcb, dfbace). Swapped blocks c & f in the Positive Sequence.

Block	Right-Of	Left-Of	Above-Of	Below-Of
a	{c}	{f}	{e}	{b, d}
b	∅	{d, f}	{a, c, e}	∅
c	∅	{a, d, f}	{e}	{b}
d	{b, c}	∅	{a, e, f}	∅
e	∅	∅	∅	{a, b, c, d, f}
f	{a, b, c}	∅	{e}	{d}

1. The sequence in which the swap takes place (Here is the Positive Sequence),
2. Which blocks participate in the swap (Rows c and f in the example's Relation Table),
3. Which blocks lie between the 2 swappees in both sequences (if any) (Row a in the example's Relation Table) and,
4. Which blocks lie between the 2 swappees in the sequence in which the swap takes place (if any) (Row d in the example's Relation Table).

These facts make the swap in one sequence move an unpredictable one, since it can swiftly affect a lot of blocks, making the move seem almost nonsensical with a naked eye. On the other hand, this is a blessing in disguise. When the initial floorplan is random and the solution is in a state of high entropy, affecting a lot of blocks simultaneously can drastically alter and improve the solution as we will discuss further down the road.

4.1.2 Main Algorithm

For the purposes of this work we initially tested the original Longest path based evaluation proposed by Murata et al. [14]. As described in subsection 3.1.1 and subsection 3.2.1 Sequence Pair evaluation is the main bottleneck of the entire methodology. As we can also see, things get even worse when we take into consideration, that we have to maintain the Relation Table in order to have fast evaluation, which as seen in Table 4.2 can be incredibly tedious. This grows prohibitively expensive, because as is apparent the table is of size $O(n^2)$. For this reason we also implemented a slight variation of the evaluation methodology proposed by Tang et al. [16]. In our modified Tang algorithm we simply disregard the complicated binary search algorithm, in order to simplify the proposed data structure even further and all parent searching is performed by traversing the tree bringing the overall complexity to $O(n \log n)$. The modified Tang approach showed an immediate 10x improvement in execution which was due to a multitude of factors. Firstly, the method by Tang is based in LCS and not in the Longest Path algorithm, and thus eliminates the need to construct both

the Relation Table and the HCG and VCG, immediately gaining execution time and memory. Secondly, its algorithmic complexity is lower $O(n \log n)$ than the Longest Path approach and its simplified structure contributes to better memory access patterns, which in combination with low-level optimisations, such as representing binary trees as arrays, and only using characters, also drastically improves performance.

4.1.3 Description of novel moves

To our knowledge, all Sequence Pair works in the literature to this day use the same set of 3 moves described in subsection 4.1.1. Up to this point some small experiments were performed in order to better understand how Sequence Pair behaves under a simple optimisation loop. These early tests were performed under the following assumptions:

- All moves are equiprobable,
- The starting solution is a randomly generated Sequence Pair and
- We only accept improving moves.

A big issue that initial tests showed was the weakness of the optimiser to fit all blocks inside of the designated core, since it is agnostic of the core dimensions. One other observation was that, as stated in subsection 4.1.1, swap in one sequence move had great effects in the floorplan, especially in random floorplans. Armed with that knowledge, we devised a small and naive optimisation plan.

Algorithm 1: Naive Out of Core optimisation

Data: A floorplan, a list of the floorplan blocks, maxiterations
Result: A floorplan with improved white space

```
1: Find all out of core blocks and create a list called L
2: foreach outofcoreblockb ∈ L do
3:     for iinmaxiterations do
4:         Choose a random block inside the core
5:         Choose a random sequence
6:         Perform a swap in one sequence
7:         if whitespace NOT improved then
8:             Revert the move;
```

This methodology massively boosted the success rate of optimisation moves and improved the QoR. Later, we adapted algorithm 1 to also include the swap in both sequences move which further increased the QoR. An artifact of this methodology however, was that, it lead to the optimiser getting stuck to a local minimum, which is not ideal, which forced us

to abandon this methodology. In this section, we briefly touched on the subject of move success rate, a topic which seems and is very intriguing. For now though, we will conveniently disregard it and we will come to examine it exhaustively in section 4.5.

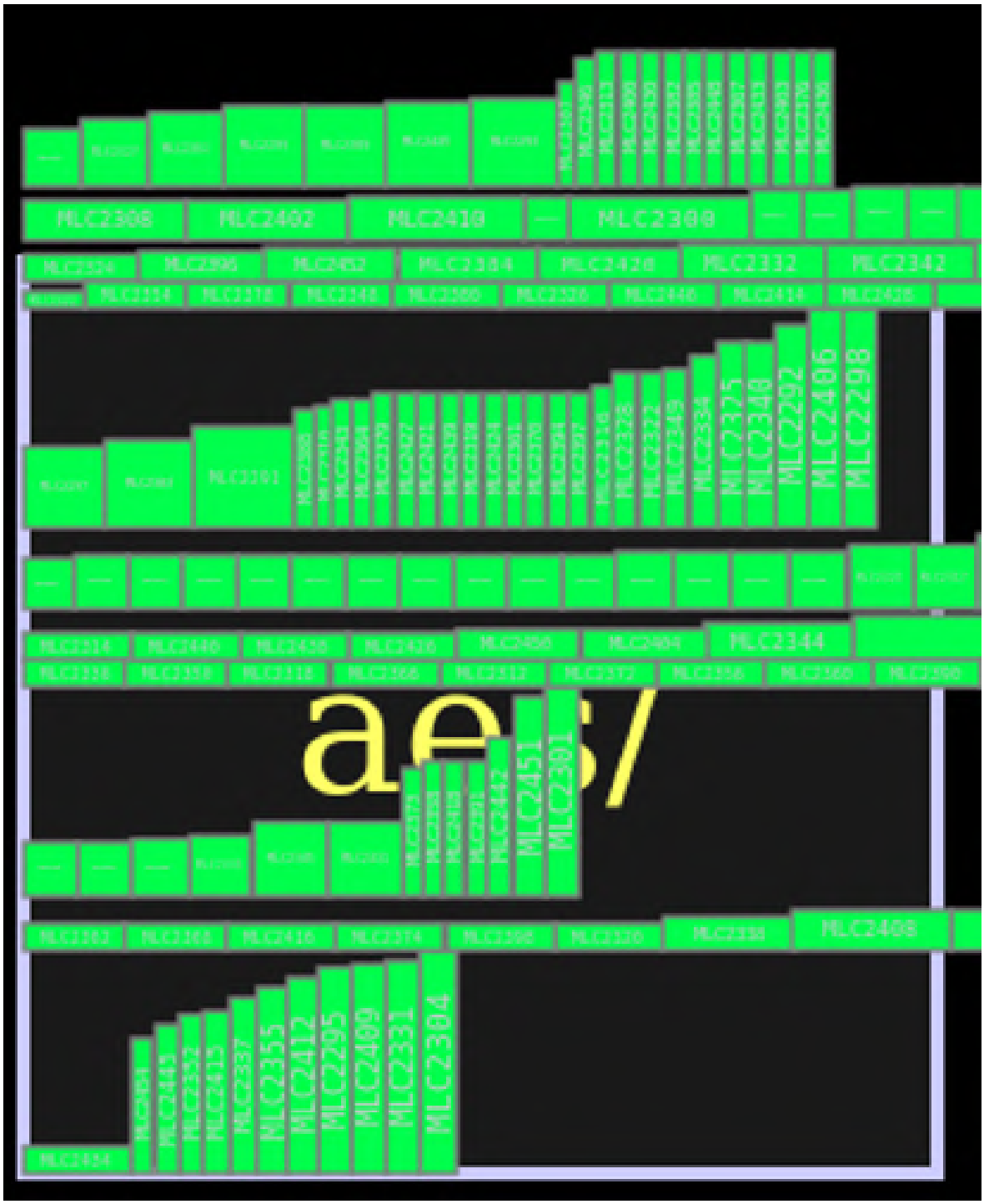


Figure 4.3: Random starting floorplan after a set of rows from out of core blocks is created.

In our search for better optimisation techniques, we paid close attention to the basic axioms of Sequence Pair, as described in section 3.1. We realised that we can exploit the basic block relations that determine the relative positions of blocks to create new optimisation

moves. These moves have Engineering Change Order (ECO) characteristics, since they effectively put 2 blocks directly next to each other, according to the desired relative position (Above/Below/Right/Left). These new 4 moves operate under the following simple steps:

- Select 2 blocks (target and swappee) and the desired relation to be enforced,
- Move the swappee directly next to the target in both sequences to create the desired relation.

An interesting side-effect of these procedures is the fact that, since the 2 blocks that took part in the move are now next to each other in both sequences of the pair, we can think of them as a new hyper-block with a bounding box that of the worst i.e. largest dimensions (width/height) of the 2.

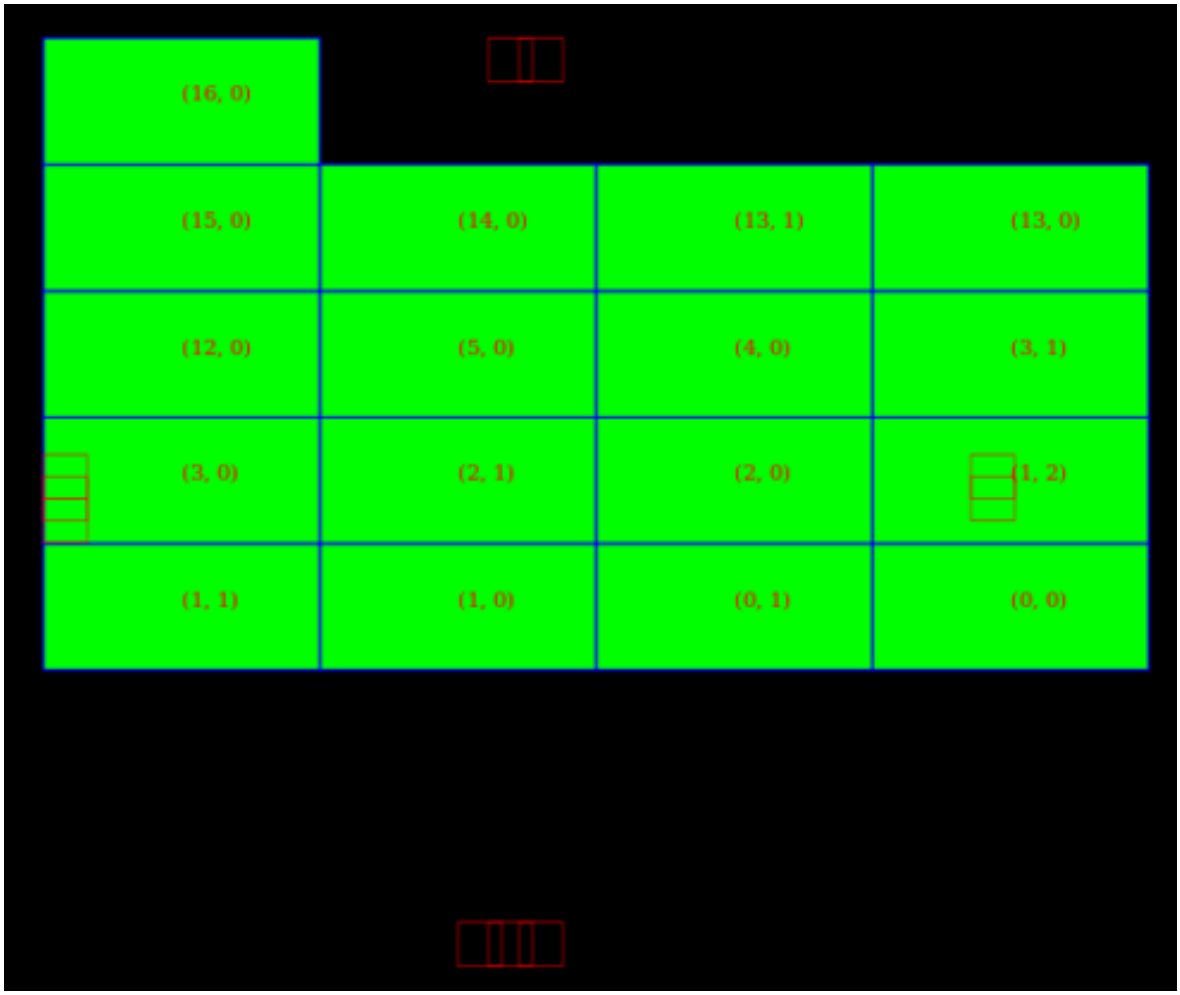


Figure 4.4: Initial floorplan for the description of Put Moves below.

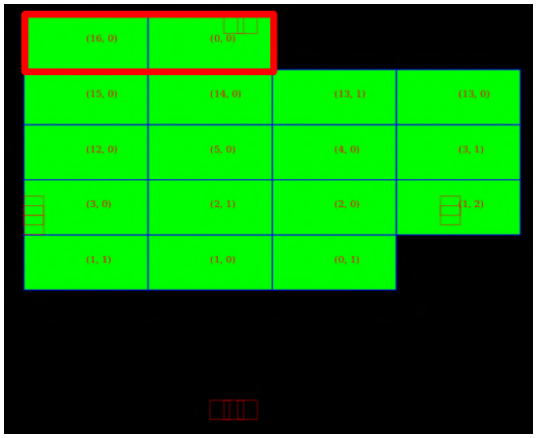


Figure 4.5: Put Right of.

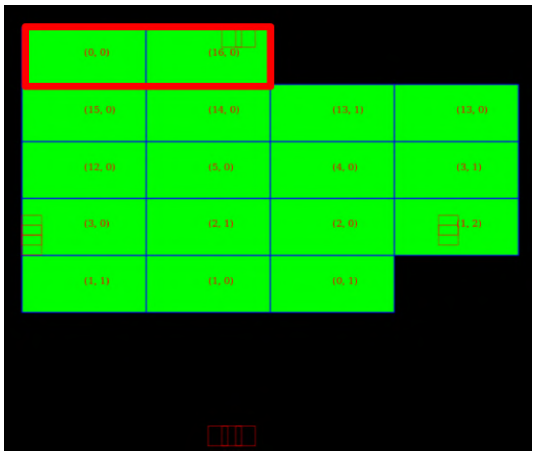


Figure 4.6: Put Left of.

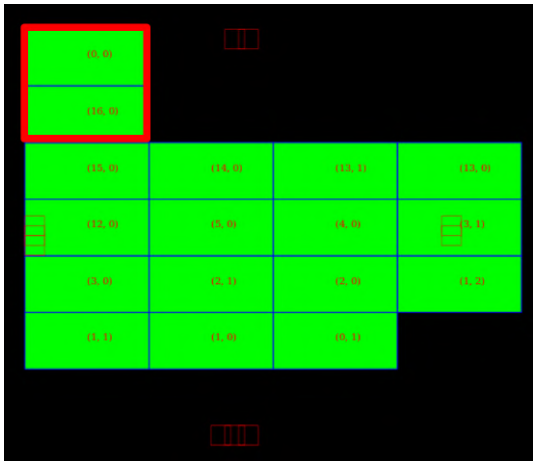


Figure 4.7: Put Above of.

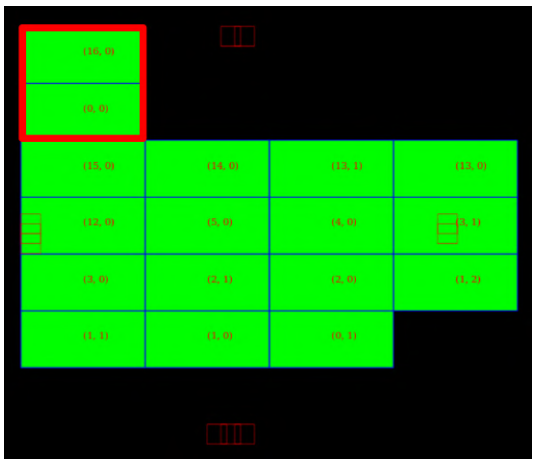


Figure 4.8: Put Below of.

Figure 4.9: Put move examples, where (16,0) is the target and (0,0) is the swappee.

This property can be further exploited. By creating hyper-blocks of arbitrary member number in this manner, we have essentially created a methodology that can create rows and or columns inside the Sequence Pair representation. This is an incredibly potent move, especially when our blocks have similar or identical dimensions, because this creates a floorplan with near zero or even zero white space when the blocks have one identical dimension (e.g standard cells). As we can see, this kind of move can serve as a superior alternative to the Naive Out of Core optimisation described earlier. We call this move Create Out of Core Rows. We will also discuss this in section 4.2.

4.2 Initial Sequence Pair

In the original Sequence Pair work very little was done to address the impact and even propose different starting solutions. In that work Murata et al. utilised a Sequence Pair, which had the form (abcde, abcde). it can be easily seen that such a Sequence Pair is of very little

interest because it simply arranges all blocks in a straight line, and presents a horrible starting solution. In later works, it became common to use a randomized Sequence Pair for the initial solution, but not further work was put in that aspect by the literature, since the random floorplan gave very promising results. Hence, the random starting solution remained unchallenged as the best starting solution.

Be that as it may, we felt that there was a lot of untapped potential for optimisation in this direction. If we combine what we learned in subsection 4.1.3 with the need for a better starting solution, we can see a real candidate. Creating a starting Sequence Pair out of block rows/columns is now extremely easy. If we are clever about how we create these, we can even tackle 2 birds with one stone, fixing both the out of core problem and provide a starting floorplan with minimal whitespace. Both of these can be achieved in the following steps:

- Since we have total control over which blocks participate in each row, creating a row that fits inside the core is as trivial as summing the widths of the blocks.

- By sorting the blocks based on height we can create as close to uniform height rows as possible, thus minimizing whitespace.

In this manner, we can achieve a starting solution that can even be acceptable in a time that is solely dependent on the complexity of the sorting algorithm used, a stunning accomplishment. Nonetheless, not everything is as bright as it may originally seem. How successful this method is, highly depends on the dimensions of the blocks themselves. Having a lot of variation in the block dimensions can cause big gaps of whitespace in the placement that can be hard to overcome by the optimiser later on. Sometimes, it may even be impossible to create a floorplan that exceeds the core dimensions in one direction. We can somewhat mitigate such results by rotating some of the problematic blocks but this is a heuristic optimisation that is neither foolproof nor provides any guarantees of optimality. If we are dealing with soft-blocks, there is another degree of freedom that can allow us to further optimise, but this creates other problems, mainly complexity ones, due to selecting the optimal aspect ratio for a block, being a problem with exponential complexity. For the purposes of this work, we determined that this solution will suffice for the moment. We will come back to the initial solution problem later in section 4.5.

Figure 4.10: Single column Sequence Pair.

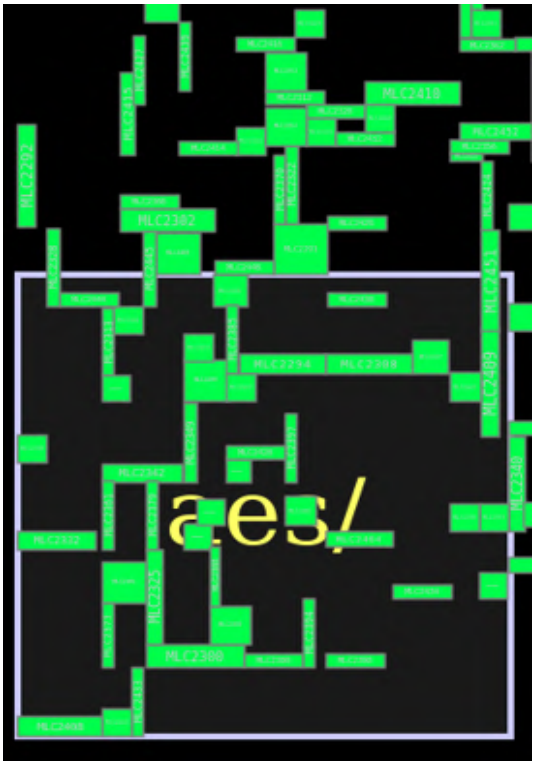


Figure 4.11: Random Sequence Pair.

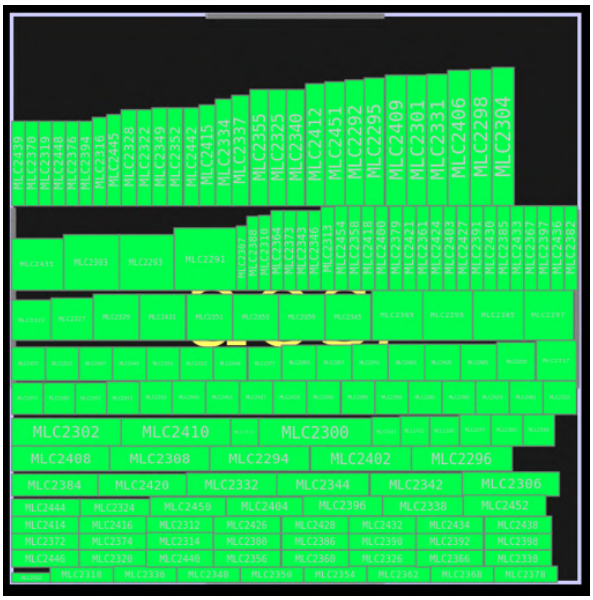


Figure 4.12: Similar height rows Sequence Pair.

Figure 4.13: Different starting Sequence Pair solutions.

4.3 Simulated Annealing Experimentation/Exploration for Sequence Pair application

SA is an optimisation technique first introduced by Kirkpatrick et al. [29] originally used to help speed up the traveling salesman problem. We must mention here that Lester Ingber has provided through his work incredible insight and expansions to SA. His work in [30], [31] and [32] has been detrimental to the point that, MATLAB uses his open source work. Over the years, the literature warmly embraced it as versatile tool, especially when it came to the optimisation of convex problems with multiple local minima. Much like other optimisation algorithms, it aims, through hill climbing, to ascend past locally optimal solutions to as close to a globally optimal one.

The origin of the technique is particularly interesting. The SA methodology originates from metallurgy, where metals are heated to a high temperature. At this state the materials exhibit a remarkable degree of malleability, and are very accepting to changes. As the temperature gradually drops, and the material becomes more rigid it no longer easily accepts changes. The SA methodology, for our purposes, attempts to emulate such behaviour, in order to shape the initial malleable solution of an initial floorplan to a compact block packing with minimal whitespace. To accomplish that, it uses temperature as a critical variable.

4.3.1 Understanding the complexity of Simulated Annealing

The SA process can be deconstructed into two fundamental components.

1. The way in which the temperature is gradually reduced.
2. How, based on the temperature value, we determine which moves should be accepted.

Although this might sound easy at first, it couldn't be further from the truth. As it turns out, SA is an extraordinarily number-sensitive method and the results we obtain from an optimisation loop depend on a multitude of factors ranging from:

- The initial temperature,
- The rate at which the temperature is decreased,
- The size of the design, with larger designs producing significantly larger metrics,
- The arithmetic magnitude of the cost function value. This is of immense importance, since different metrics used to measure the viability of a floorplan have considerable arithmetic deviation. For example, as a cost function we can use either the whitespace or the wire length of the floorplan or even its timing slack. Since these are completely different metrics their scale is also completely different. This becomes even worse, if we factor in the unit we use for each one.

4.3.2 Systematic Exploration

All these factors create a very chaotic environment. And this is something the literature has not poured a lot of light upon. We only found mentions of the parameters used in [20] and [33], but as we mentioned, these values tend to be way off the intended ones due to the sheer variability of the problem.

As has become apparent, there are a lot of problems and we have not yet talked about how the move acceptance probability is calculated, which is a testament to how complex this turned out to be. To tackle the problem, we developed in Python and in Desmos [34] a systematic a prototype to observe the properties of all these variables and determine the best possible approach for our purposes. To ensure that our work provides as general a solution as possible, and cover all possible use cases, we developed the prototype as a pure mathematical model. This will allow us to methodically study the effects all parameters have in the final result.

As proposed by Kirkpatrick et al., the heart of our model is based around the Metropolis Criterion as described in [35]. We have slightly modified the Metropolis Criterion to generalise it by adding a few scaling factors we will describe below, in order to combat problems due to the differences in numerical magnitude. The Metropolis Criterion is used to calculate the acceptance probability of a move and is presented in Figure 4.14.

$$P(x, U) = e^{-\frac{c \cdot \Delta E}{b \cdot T(x,U)}} \tag{4.1}$$

Figure 4.14: The Acceptance Probability Equation.

We will now explain the parameters of the equation in Figure 4.14.

- The inputs to the equation, x and U, refer to the current iteration of the optimisation loop and the current temperature value respectively.
- ΔE is the cost difference between the current and previous state (referred as the Energy Difference).
- $T(x, U)$ is a function that manipulates the temperature based on the current iteration, which we will discuss in detail later on.
- b and c are simple scaling factors used to handle the numerical magnitude differences.

What is integral to the understanding of how this equation behaves, is the fact that if the move is improving our cost the Energy Difference is numerically negative. Such moves are always accepted by the optimiser and thus the probability does not need to be calculated.

What is implied is, that since all other values of the Energy Difference are zero or positive. Combining that with the fact that the temperature and scaling factors are all positive numbers, the result of the equation is bounded to values between 0 and 1. Hence, this is a way to calculate the acceptance probability.

We will demonstrate a simplified example to showcase how acceptance probability can vary over a range of values in Table 4.3

P(10000) = 0.99004	P(9000) = 0.98895	P(8000) = 0.98757
P(7000) = 0.98581	P(6000) = 0.98347	P(5000) = 0.98019
P(4000) = 0.97530	P(3000) = 0.96721	P(2000) = 0.95122
P(1000) = 0.90483	P(100) = 0.36787	P(10) = 0.00004

Table 4.3: Acceptance Probability Values for $\Delta E = 100, b = 1, c = 1$ and varying the denominator of the fraction from Figure 4.14.

As we can see in the naive and artificial example of Table 4.3 the acceptance probability presents a very steep decline and accepts almost all moves for the better part of each temperature range.

We will now focus on temperature manipulation. We will manipulate the temperature descent by introducing 2 terms:

1. The annealing frequency, meaning the iteration interval in which temperature is decreased and
2. The annealing methodology, referring to the mathematical function used to reduce the function.

The temperature manipulation is described by Figure 4.15.

$$T(x,U) = \begin{cases} U, & \text{if } x \mod annealing\_frequency \neq 0, \\ U - w \cdot f(x,U), & \text{where w is weight and f is the mathematical function, otherwise.} \end{cases} \tag{4.2}$$

Figure 4.15: Temperature decrease function.

For the mathematical function we tested 3 difference approaches.

1. A linear based one, where we subtract the iteration number from the current temperature,
2. A square root based one, where the square root of the iteration number is subtracted from the current temperature and,
3. An exponential based one, where the temperature is multiplied by a constant factor.

We combined all of the above with different weights and different annealing frequencies and ended up with 12 methodologies each with a different sweet spot for optimal operation. For each experiment, a different annealing weight, frequency and factor was used, in order to produce viable results. For the most prevalent methodologies, a set of near-optimal parameters found was kept as a default set for that methodology. We will discuss them further down. The results of this series of experiments can be seen in Figure 4.16.

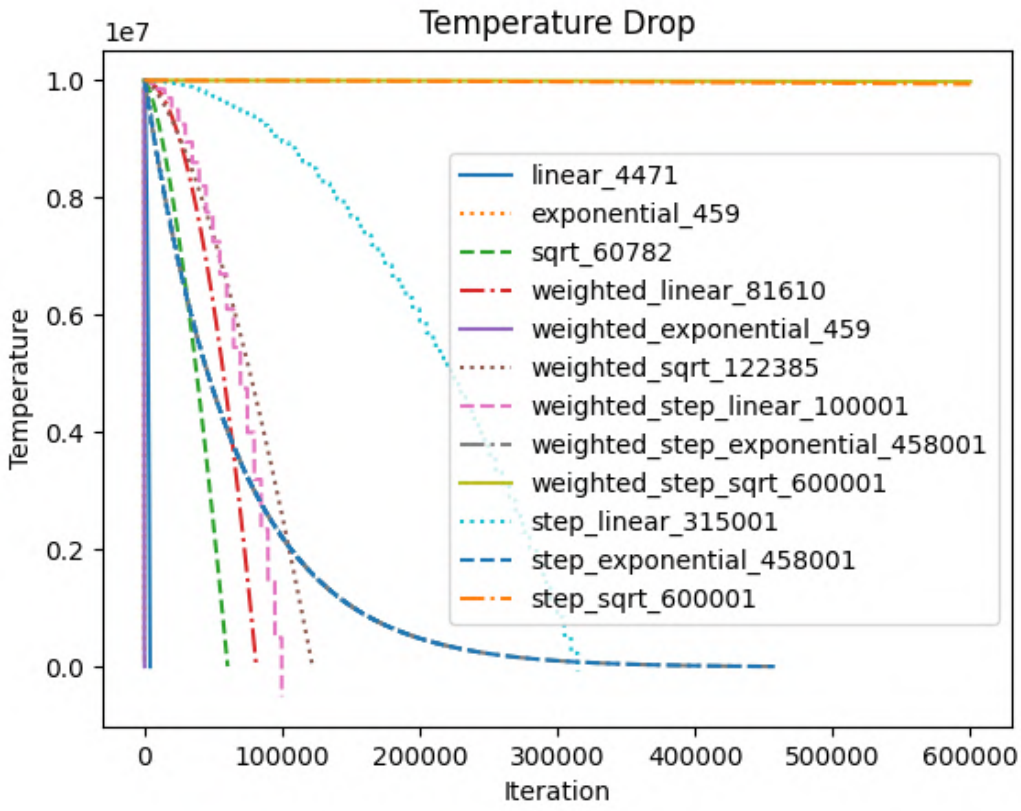


Figure 4.16: Different methodologies for decreasing annealing temperature. Legend shows in which iteration temperature reached 0.

To further understand how SA functions we also developed in Desmos' Graphing Calculator [34] a version of this prototype that can, in real time, vary all parameters over the desired ranges. This research is public [36]. Below we can see a frame of this analysis.

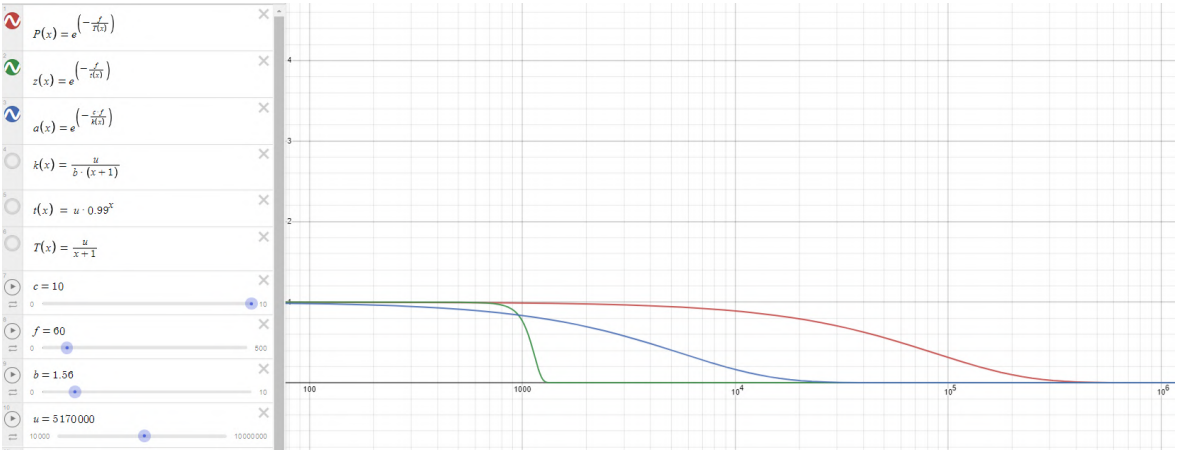


Figure 4.17: Acceptance Probability visualization in real time in Desmos.

It is evident from Figure 4.16 and [36] that our work in annealing exploration is generic enough that it can be used in a wide range of applications. However, we need to mention at this point that preliminary tests showed that for designs up to 500 blocks presented an ideal maximum iteration limit of around 400000 iterations. With that in mind, and aiming to reduce noise in our results and limit the excessive testing, we tuned the scaling factors, annealing frequencies and weights to produce distinct temperature curves around this range.

From the experiments in Figure 4.16 we determined that for our most suitable candidates temperature drop methodologies to continue testing were the following 7[1]:

- Step linear,
- Weighted step exponential,
- Weighted square root,
- Weighted step linear,
- Step square root,
- Square root and
- Weighted linear.

We will now showcase in the following figures, how the acceptance probability behaves for difference over an array of Energy Differences, over the iterations across the various methodologies.

[1]The term step refers to the annealing frequency being a number different than 1, i.e the temperature is only updated every few iterations. Weighted refers to the presence of tuning weights for the temperature manipulation function. When the term is not presence all weights are set to 1.

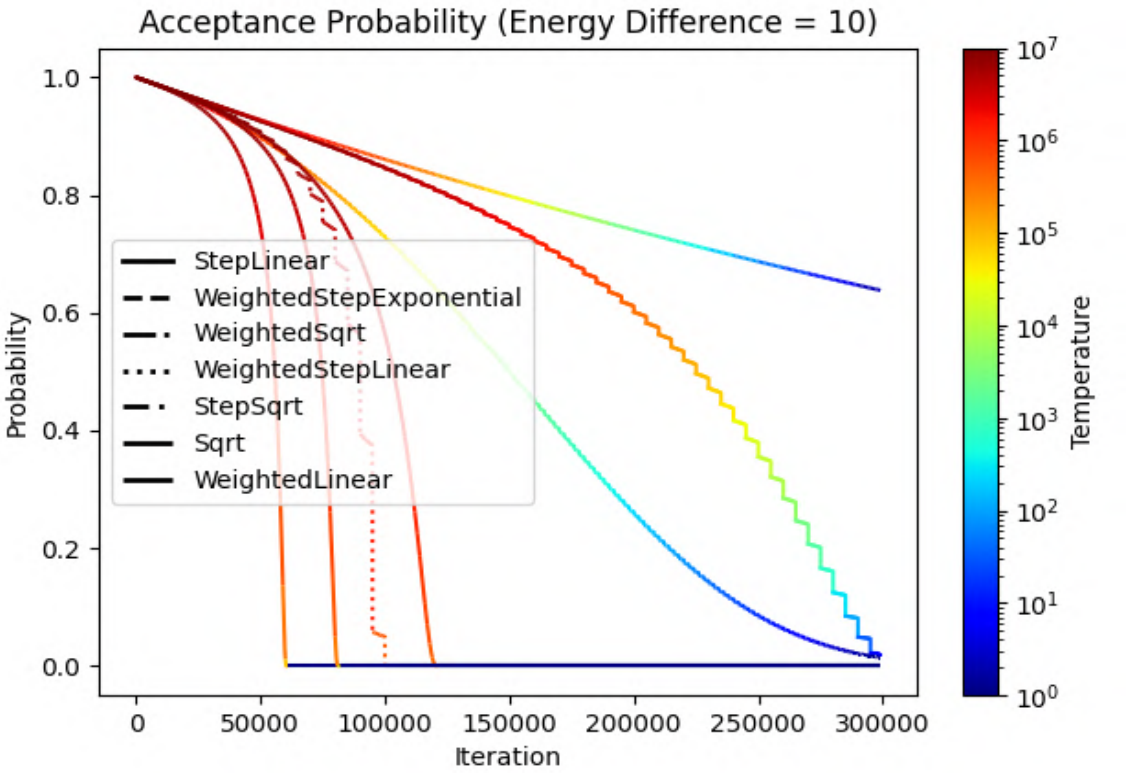


Figure 4.18: Acceptance probability with cost difference of 10.

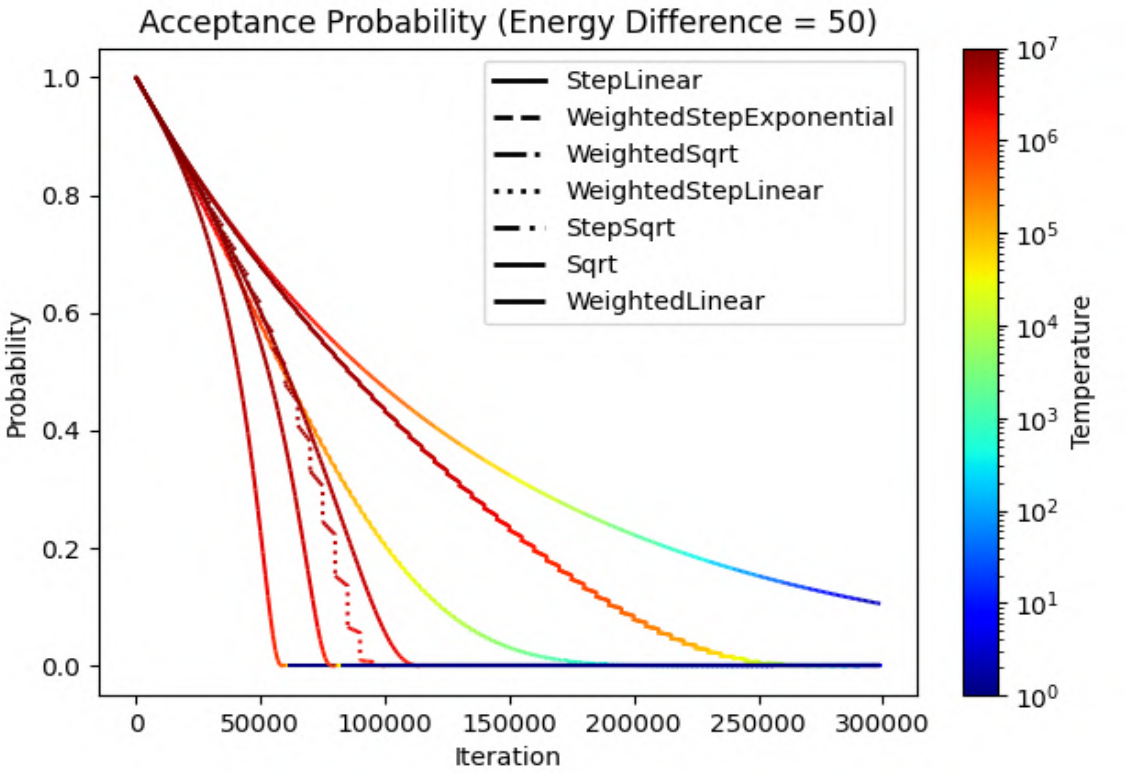


Figure 4.19: Acceptance probability with cost difference of 50.

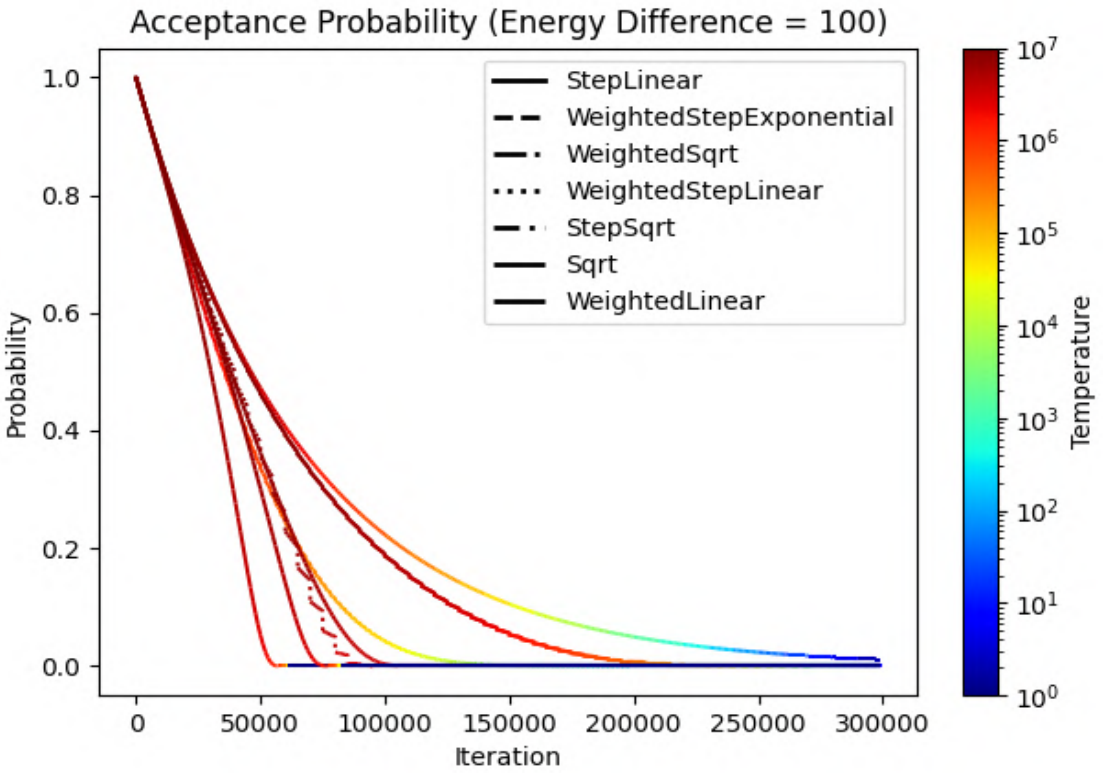


Figure 4.20: Acceptance probability with cost difference of 100.

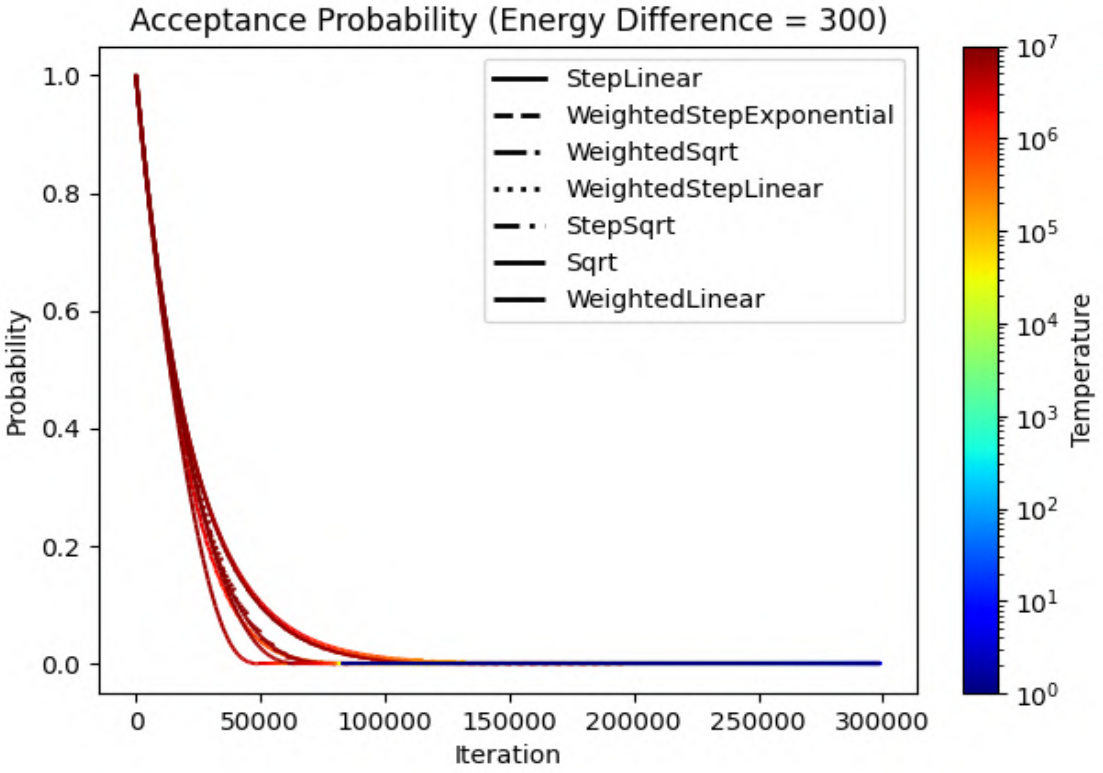


Figure 4.21: Acceptance probability with cost difference of 300.

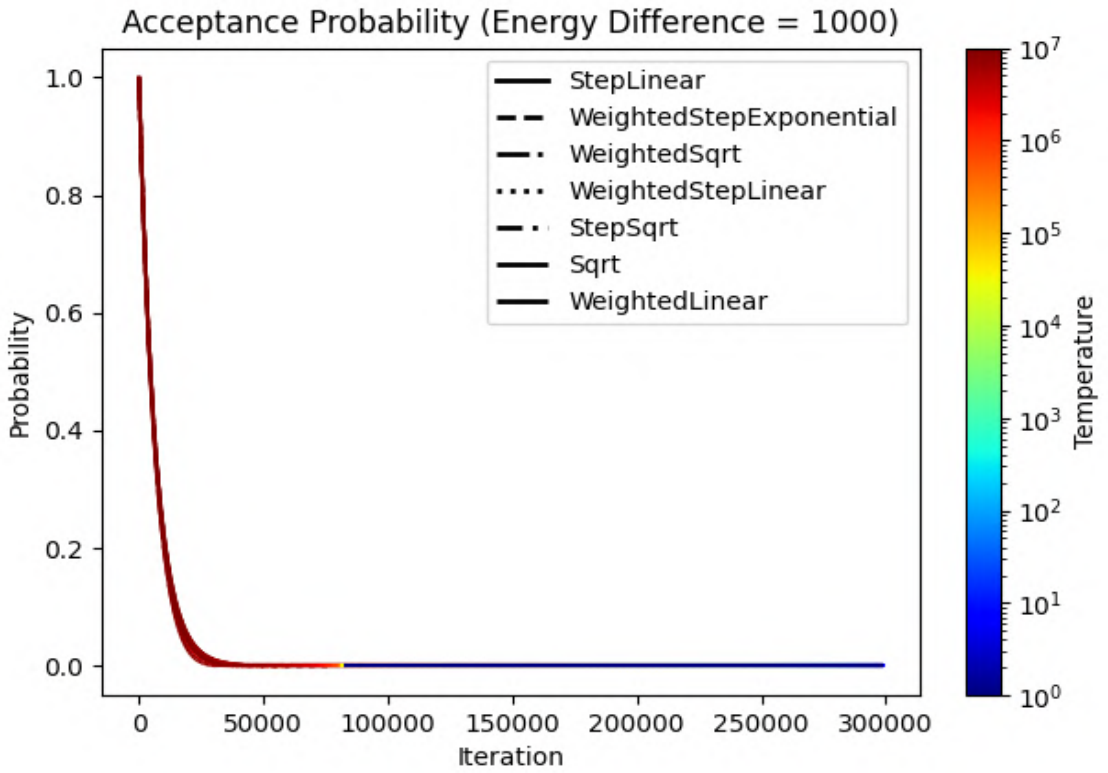


Figure 4.22: Acceptance probability with cost difference of 1000.

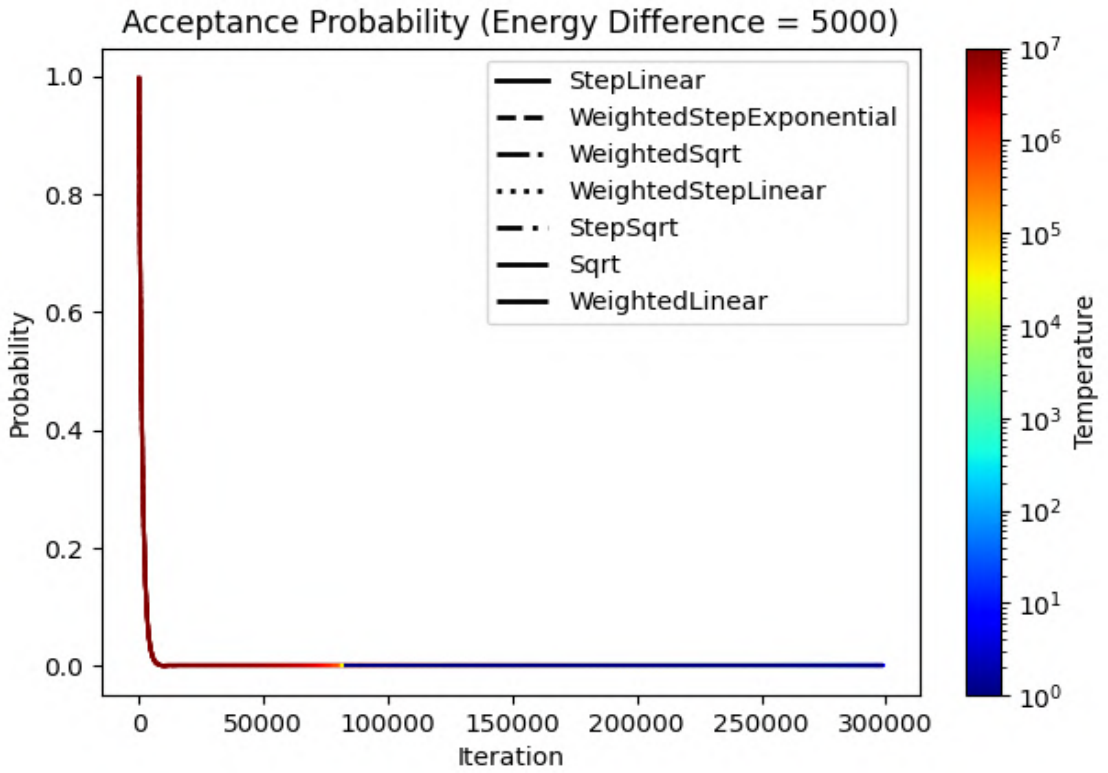


Figure 4.23: Acceptance probability with cost difference of 5000.

From the resulting graphs we can see that acceptance probability varies with different rates, something we can exploit to address a plethora of cost functions and floorplans.

4.4 On the topic of move success rate

Until now we have only developed narrow minded solutions and optimisation moves for general problems that will naturally arise during floorplan optimisation under the Sequence Pair representation. We only once and very briefly mentioned the topic of success rate of our optimisation techniques and this was not by accident. In all experiments up to this point, we have used the simplified optimisation loop first introduced in subsection 4.1.3. Here we will use a mixed SA-Greedy loop for the first time. At first glance the results from Figure 4.24 paint a very bleak picture.

In Figure 4.24, for a design with 165 blocks and for the specified parameters, we observe, for all moves, a success rate (i.e improving the global cost) of around 0.1%, which is very disheartening. It must be stated, however that this can be a little bit misleading due to the fact that this is an SA run and the acceptance rate for SA moves fluctuates from 2% to just under 10%. Not all hope is lost though. The same experiment also shows some very welcoming observations.

1. We achieve a sub 10% whitespace result in under 12 seconds.

2. This result is achieved through a very small number of total improving moves.

If we take a closer look at these results, things may not be that bleak after all, and even rejoice. What these results make clear is that our decision making is extremely poor. Since only a few moves were enough to achieve a result with sub 10% whitespace, choosing the correct sequence of moves to perform can significantly impact our runtime.

For this reason we performed an analysis of our basic moves, in search of a better move selection strategy.

Simulated Annealing		Greedy	
Average Energy Difference		Average Energy Difference	
6157.093		5204.21	
Swap in One Sequence	35747	Swap in Both Sequences	44381
Swap in Both Sequences	87031	Swap in Both Sequences	108809
Rotate	54948	Rotate	69084
Total	177726	Total	222274

Table 4.4: Average Energy Difference and move distribution for a 400k iteration optimisation loop of a 165 block floorplan.

```
INFO: Optimisation Results:
        INFO: Starting cost: 63594.340, Best Cost: 5751.886, Improvement: 90.96%
        INFO: Total Width = 214.879, Total Height = 270.518.
        INFO: Total Area = 58128.574.
        INFO: Total Whitespace = 5751.886/58128.574 (9.895%).
        INFO: Aspect ratio = 1.259.
        INFO: Best Result achieved in iteration 277547/277547.

INFO: Best Improving strategy stats
        INFO: Total Best Improving moves: 430/277547 (0.155%).
        INFO: Best Improving Rotations: 2/14183 (0.014%).
        INFO: Best Improving Swaps in Both Sequences: 318/159249 (0.200%).
        INFO: Best Improving Swaps in the Positive Sequence: 41/33087 (0.124%).
        INFO: Best Improving Swaps in the Negative Sequence: 6/14127 (0.042%).
        INFO: Best Improving Put Right Of Moves: 19/14102 (0.135%).
        INFO: Best Improving Put Left Of Moves: 19/14346 (0.132%).
        INFO: Best Improving Put Above Of Moves: 10/14177 (0.071%).
        INFO: Best Improving Put Below Of Moves: 15/14277 (0.105%).

INFO: Successful strategy stats
        INFO: Total successful moves: 22320/277547 (8.042%).
        INFO: Successful Rotations: 397/14183 (2.799%).
        INFO: Successful Swaps in Both Sequences: 15573/159249 (9.779%).
        INFO: Successful Swaps in the Positive Sequence: 3042/33087 (9.194%).
        INFO: Successful Swaps in the Negative Sequence: 506/14127 (3.582%).
        INFO: Successful Put Right Of Moves: 767/14102 (5.439%).
        INFO: Successful Put Left Of Moves: 810/14346 (5.646%).
        INFO: Successful Put Above Of Moves: 654/14177 (4.613%).
        INFO: Successful Put Below Of Moves: 572/14277 (4.006%).

Failed strategy stats
        INFO: Total failed moves: 255227/277547 (91.958%).
        INFO: Failed Rotations: 13786/14183 (97.201%).
        INFO: Failed Swaps in Both Sequences: 143676/159249 (90.221%).
        INFO: Failed Swaps in the Positive Sequence: 30045/33087 (90.806%).
        INFO: Failed Swaps in the Negative Sequence: 13621/14127 (96.418%).
        INFO: Failed Put Right Of Moves: 13335/14102 (94.561%).
        INFO: Failed Put Left Of Moves: 13536/14346 (94.354%).
        INFO: Failed Put Above Of Moves: 13523/14177 (95.387%).
        INFO: Failed Put Below Of Moves: 13705/14277 (95.994%).
DEBUG: clock_end = 14430000, clock_start = 2640000
REPORT: CPU Time = 11.790 sec
```

Figure 4.24: Example of typical move success rates in a simple annealing run.

What we realised from the collected data, aligns with the intuition we first presented in subsection 4.1.1. This means that swapping blocks in one sequence, with its high unpredictability, has the greatest effect early on in the process (high temperature) and falls of dramatically later on when more fine grained moves such as rotations and swapping blocks in both sequences are more favourable.

These results lead us to once for all abandon the equiprobable move model, that as can be seen in the tables we had already begin to abandon, in favour of a set of tweaked initial probabilities for each move based on statistical history. It further prompted us to develop a system that dynamically adjusts the probability of each move based on how each move performs throughout the optimisation process. This means greatly rewarding global improving moves, mildly rewarding annealing accepted moves and penalising worsening moves. This

Annealing Improvements		Greedy Improvements	
Average Energy Difference		Average Energy Difference	
-325.007		-12.148	
Swap in One Sequence	37	Swap in One Sequence	6
Swap in Both Sequences	230	Swap in Both Sequences	80
Rotate	183	Rotate	77
Total	450	Total	163
Swap in One Sequence Total Improvement	-36905.8	Swap in One Sequence Total Improvement	-56.661
Swap in Both Sequences Total Improvement	-90448	Swap in Both Sequences Total Improvement	-991.099
Rotate Total Improvement	-18899.5	Rotate Total Improvement	-932.419
Improvement Per Move		Improvement Per Move	
Swap in One Sequence	-997.453	Swap in One Sequence	-9.443
Swap in Both Sequences	-393.252	Swap in Both Sequences	-12.388
Rotate	-103.276	Rotate	-12.109
Average Gap Between Successful Moves	395.4922	Average Gap Between Successful Moves	1330.4876

Table 4.5: Average Improvement and Gap per successful move.

proved to be a grater challenge that predicted because tuning the probabilities over hundreds of thousands of iterations can quickly degenerate the probability of each move. This lead to further experimentation, to determine the exact rewards and penalty values, as well as the introduction of minimum and maximum cap values for each move. We shall from now on call this system Dynamic Probability Adjustment (DPA).

Although the performed optimisations lead to improvements we will later discuss, we must also address a few problems that may warp our perception of the statistical results.

1. As the floorplan converges to a final solution more and more moves will be worsening and thus, rejected. This is mentioned because, throwing more compute power at the problem, in the form of extra iterations, can only produce noise in our statistics and lead us astray.

2. The optimisation loop is inherently random. This may lead to extreme variation across runs both due to different starting positions and different decision making along the way. Thus, we need to be careful in how we extract our results, as we will see in chapter 5.

4.5 Novel Optimisation Loop

In this section we will provide details into, how we combined all methodologies and innovations described above in a versatile and customizable optimization flow. The flow is controlled by flags that can enable and disable all the features we have described so far, providing full control of the process to the user. This however can be a double-edged sword, since there is a plethora of parameters to control, and finding a set of optimal ones is really challenging. For this reason, through extensive experimentation, as described in section 4.3 and chapter 5, a default set of parameters was painstakingly created.

All Sequence Pair works use randomness to some degree, to determine which move should be performed in each iteration. In our initial experiments we followed such an approach. In that work we made all moves equiprobable and only accepted improving moves. Although the results were somewhat decent it quickly became apparent that such an approach would not suffice, as an enormous amount of iterations was required for the process to produce decent results. By turning to the literature we immediately realised that SA was the best way to tackle our problems.

The main idea of the optimisation flow we propose is its decomposition to 3 distinct phases.

1. The initialisation phase,
2. The Simulated Annealing phase and,
3. The Greedy phase.

The initialisation phase consists of the choice of the initial Sequence Pair, as well as a set of minor optimisation to improve the starting solution. These optimisations consist of the Naive Out of Core optimisation algorithm 1 (which is disabled by default) and the Create Out of Core Rows move (which is enabled).

The Simulated Annealing phase consists of a loop that performs the optimisation approach that bears the phase's name. As briefly mentioned in section 4.3, a lot of parameters are required for SA to operate. So far we have mentioned the different weights specific to each annealing sub-method, the initial temperature, the way we manipulate temperature, how frequently we manipulate it and the mathematical method we use to do it. Despite the multitude of parameters discussed, there is also an artifact of the methodology we are yet to discuss. Preliminary tests have showed that the SA optimisation methodology as whole can occasionally become stuck to a solution early on. This is due to the fact that the methodology inherently accepts worsening moves. If not controlled, the method can come to such a bad state cost-wise, that it will be unable to recover. For this reason, it was necessary to introduce the Reset-to-best flag. What this does is, if a large enough number of worsening moves is performed, the algorithm interprets it as being stuck in an unfavourable position, and instead

of reverting to the previous state, it reverts to the best known solution to resume its search. The number of the failed moves was set to 250 by consulting Table 4.4. Upon exiting the annealing phase another round of the minor optimisations can be performed described in the first phase can be performed.

Before we enter the final, Greedy, phase we can perform another optimisation. According to the findings from Table 4.4 and Table 4.5, it is clear that move effectiveness completely changes in the latter half of the optimisation loop and the swap in on sequence move loses its effectiveness. What we can do, is minimise the probability of this move in favour of the much more potent swap in both sequences (enabled by default). After that the Greedy phase continues by only accepting the best improving moves.

In both the SA and Greedy phases we have integrated the DPA described in section 4.4 (enabled by default). We also introduce a small execution time upgrade in these 2 phases. If at any point in these phases a certain predetermined whitespace percentage is reached, the optimisation flow exits as this is deemed as an acceptable solution. What is more, is the fact that all phases keep statistics to assess the solution's QoR. Finally, either phase can be disabled by the appropriate flag (all 3 enabled by default).

Algorithm 2: Core Optimisation Loop

Data: A Sequence Pair, the dimensions of the blocks in the Sequence Pair, probability of each move, maxiterations, the set of Simulated Annealing flags

Result: A near optimal and legal packing under the Sequence Pair

```
if Create Out of Core Rows is enabled then
    Perform the Create Out of Core Rows optimising move
if Naive Out of Core optimisation enabled then
    Perform Naive Out of Core optimisation
while i in range of Simulated Annealing iterations AND Temperature Limit NOT
 reached AND Acceptable Limit NOT reached do
    Get a random move with the given probabilities and perform the permutation
    if Evaluation shows move is NOT good then
        Calculate the annealing acceptance probability
        if Annealing Discards move then
            Revert to previous
    if A large number of moves fails in a row then
        Revert to best solution
    if Exit on whitespace percentage enabled then
        if Acceptable whitespace percentage reached then
            Exit
    Adjust move probability based on result
    Refresh Temperature according to the specified methodology
if Secondary Create Out of Core Rows enabled then
    Perform the Create Out of Core Rows optimising move
if Greedy phase probability adjustment enabled then
    Adjust move probability to favour swaps in both sequences while reducing the
     probability of swap in one sequence
while i in maxiterations range AND Acceptable Limit NOT reached do
    Get a random move with the given probabilities and perform the permutation
    if Evaluation shows move is NOT good then
        Revert to best
    if Exit on whitespace percentage enabled then
        if Acceptable whitespace percentage reached then
            Exit
    Adjust move probability based on result
    Refresh Temperature
```

4.6 Placement to Sequence Pair

A key problem in the literature was the Placement to Sequence Pair problem. Many in the literature even support that general placements can be represented by Sequence Pair. Through a few simple examples we will showcase, what we named as the Sequence Pair Unrepresentability Problem. Based on that we will demonstrate how popular approaches in the literature fail, to even extract, from a given Sequence Pair representable floorplan, one of the many possible Sequence Pairs that reproduce it.

The Sequence Pair Unrepresentability Problem is something we realised very late in our search for a solution to the Placement to Sequence Pair problem. But the Unrepresentability problem can be seen through the very basic example in Figure 4.25.

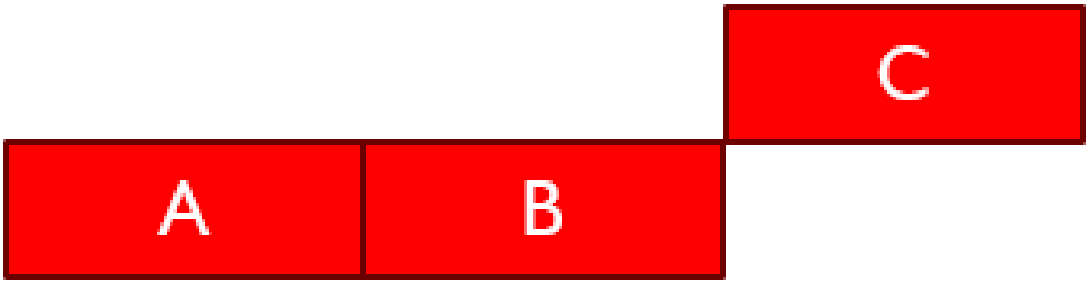


Figure 4.25: Example showcasing the Sequence Pair Unrepresentability Problem.

Although Sequence Pair may seem intuitive at first it has a few secrets. Its main weakness are diagonal relations. In Figure 4.25, block B can only have a horizontal relation with block C, since C is its only horizontal neighbour and thus it C can only get its horizontal coordinate from B. This forces the relation (BC, BC). Because B is designated as a horizontal neighbour, and 2 blocks can only have 1 relation as described in the Sequence Pair criteria in section 3.1, for the same exact reason this forces block A to have a vertical relation with C. This imposes the relation (CA, AC). If we add to that the fact that A is also Left-of B and therefore, (AB, AB) is forced as well, we have reached a situation where we have 3 statements that must all be true for the Sequence Pair to represent the given floorplan. However, if we add these statements together, we realise that we have reached a circular dependency. Therefore, there cannot exist a Sequence Pair to represent the given floorplan.

This revelation occurred to us late in the development of our own Placement to Sequence Pair solution. At this point in time, we had already implemented the algorithms described by Huo et al. [8], [11], and by Jens Egeblad [9]. In the following subsections we will debunk these algorithms along with the original Gridding procedure.

4.6.1 Gridding

The Gridding procedure, described in [14] and can be seen in Figure 3.3, is not even an algorithm as it is very loosely defined and has an estimated complexity of $O(n^3)$. Nonetheless, it has more serious problems. To begin with, since it is loosely defined and in the example the authors provide in Figure 3.3, the blocks are not packed, inconsistencies arise in the procedure. In a case that will be used to debunk all Placement to Sequence Pair methodologies and can be seen in Figure 4.26, when modules of identical dimensions are used we cannot decide the order in which they exit the top-right or top-left corners as specified by the procedure which leaves us unable to continue. Different orders were attempted but none managed to yield a Sequence Pair that can reproduce the given floorplan.

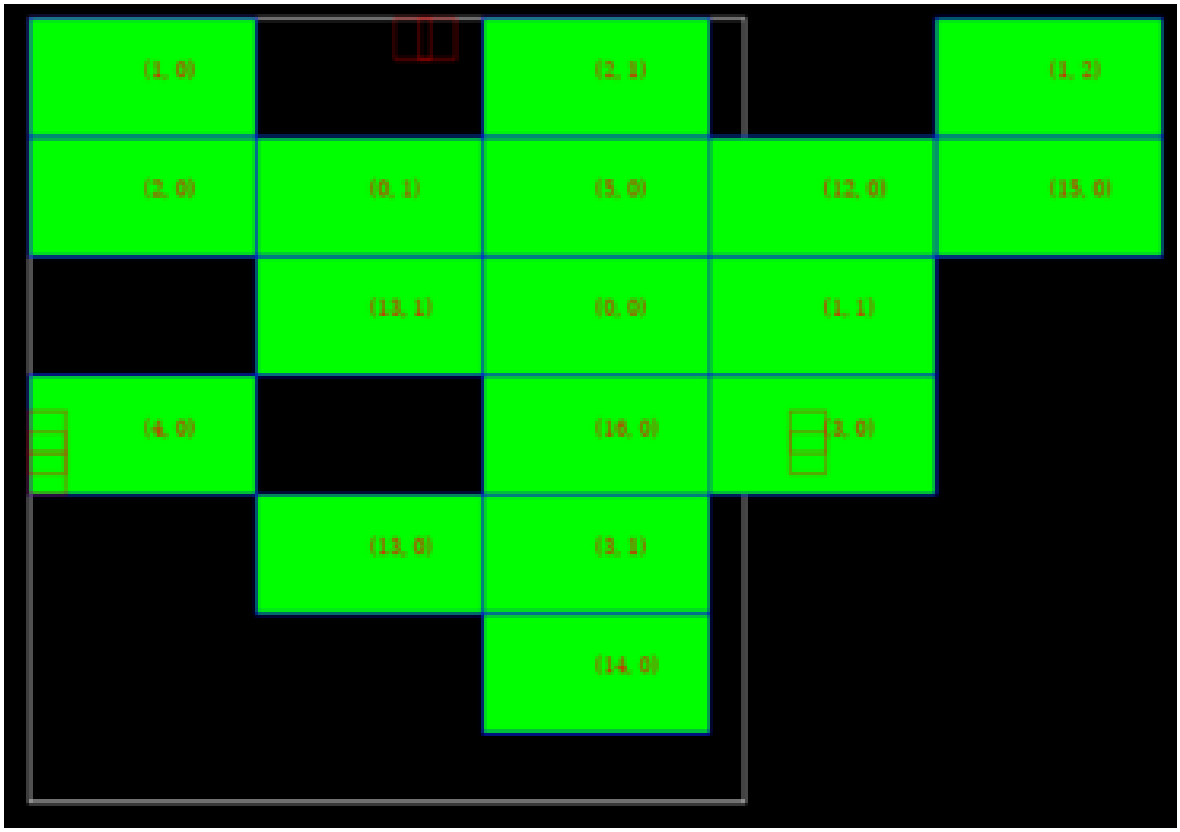


Figure 4.26: Placement to Sequence Pair debunking example.

4.6.2 Huo's Algorithm

The algorithm proposed by Huo can be seen in algorithm 3 [8]. For the sake of simplicity, we will only showcase the simple list version of the algorithm that produces the Positive Sequences, as the one that produces the Negative Sequence is trivially extracted by the Positive Sequence one and the updated version simply utilises a binary tree structure to reduce its overall complexity.

The major flaw of the algorithm by Huo et al. is that it verbosely degenerates all diagonal relations to Horizontal ones. This as discussed in the Sequence Pair Unrepresentability

problem presents a big issue. Similarly, At 3.4.1 [2] also claim that breaking constraints to horizontal ones leads to results as well.

Algorithm 3: Huo Placement to Sequence Pair [8]

Data: An empty list $L+$ representing the Positive Sequence, a list of block names (BL) along with their placed coordinates

Result: A Sequence Pair that corresponds to the given placement

```
1:  Sort blocks in BL ascending x coordinate
2:  Insert BL[0] in L+
3:  foreach other block i ∈ BL do
4:      name ← BL[i].name
5:      foreach block list_name ∈L+ do
6:          r ← relation(name, list_name)
7:          if r == LEFT || r == ABOVE // name precedes list_name
8:           then
9:              L+.insert(list_name, name) // insert name before list_name
10:             break
11:     if The end of L+ is reached then
12:         L+.append(name)
```

This produces a final result that looks like this.

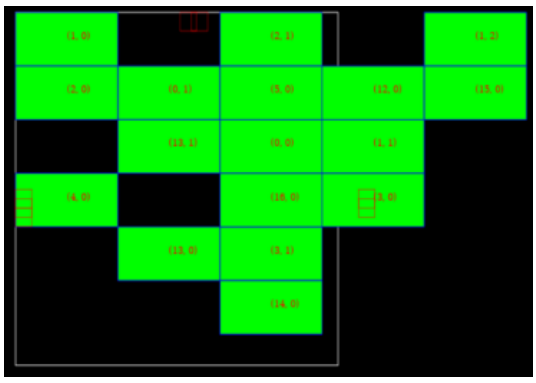


Figure 4.27: Initial Floorplan.

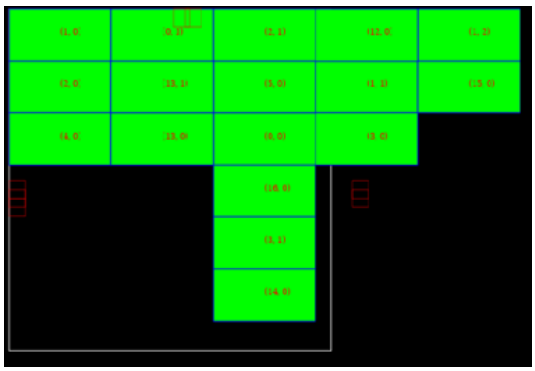


Figure 4.28: Result by Huo's algorithm.

In this example where the blocks are of identical dimensions and are packed in this checkerboard pattern, Huo's algorithm presents a nice compaction property as a side-effect. This property though, only holds for such floorplans and is not well defined.

4.6.3 Egeblad's Algorithm

The algorithm by Jens Egeblad [9], is a more complex one. It relies in performing a coordinate transformation to solve the ambiguities introduced by Gridding and succeeds to that end. However, that transformation is not well defined either, because it is based in a

secret parameter, named alpha, the value of which remains a mystery. The algorithm relies in a complex tree-like data structure in which the bottom-left and top-right points of each block are inserted in a specific manner. After the successful insertion of all points the structure is traversed in an in-order manner which produces the required Sequence Pair. In algorithm 4 we present how to extract the Positive Sequence, since the Negative one can be trivially extracted.

Algorithm 4: Egeblad Placement to Sequence Pair [9]

Data: A set of placed rectangles R, where $r \in R$, a transformation matrix as described above

Result: The Positive Sequence after an in-order traversal of the structure

```
foreach r ∈ R do
    Transform coordinates of the lower-left and upper-right corners of the using the
     input transformation matrix
    Generate breakpoints at the lower-left and upper-right transformed corners
Sort the breakpoints by increasing transformed x-coordinate
foreach Breakpoint do
    if Breakpoint is start of rectangle-segment then
        Insert the rectangle-segment into segment data structure
    if Breakpoint is end of rectangle-segment then
        Remove the rectangle-segment from segment data structure
    if Breakpoint is upper-right of rectangle r then
        Find segment s above
        Let t be the rectangle of s
    if s has negative slope in transformed coordinate system then
        Insert r as successor of t
    else
        Insert r as predecessor of t
Perform an in-order search of the tree-like structure
```

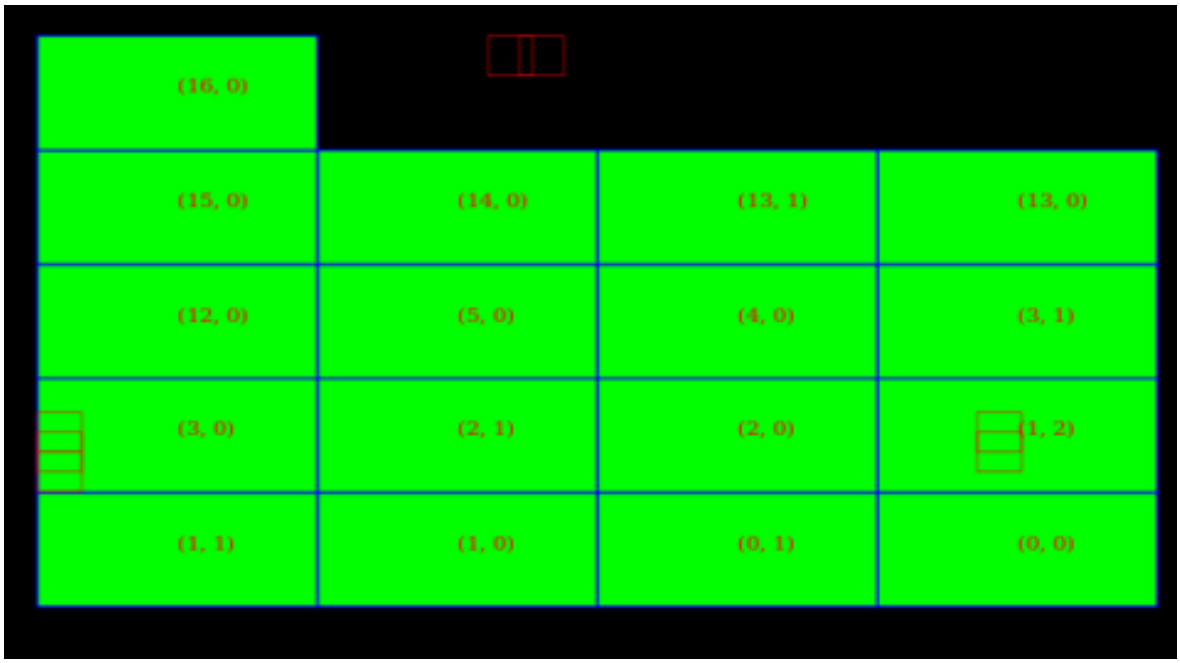


Figure 4.29: Floorplan for Egeblad's algorithm example.

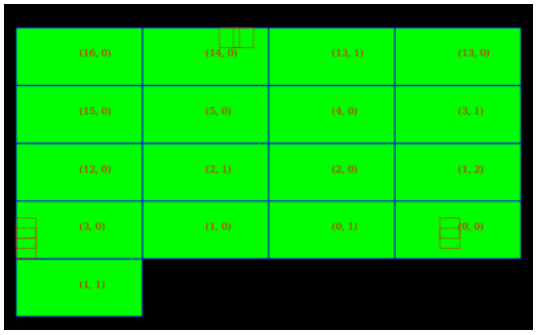


Figure 4.30: Result from Egeblad's algorithm for alpha = 1.

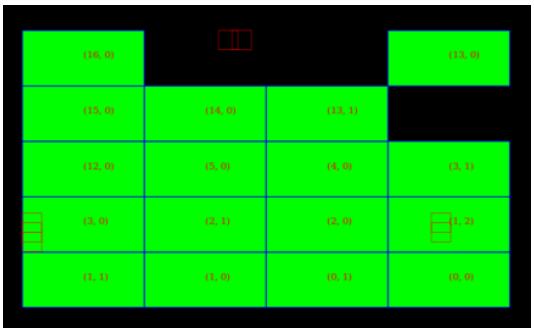


Figure 4.31: Result from Egeblad's algorithm for alpha = 10.

We experimented with multiple values of alpha and came close to a solution but never reached an exact solution as can be seen in the figures above. This small inconsistency in results scales really badly when sing the algorithm for larger more complex floorplans.

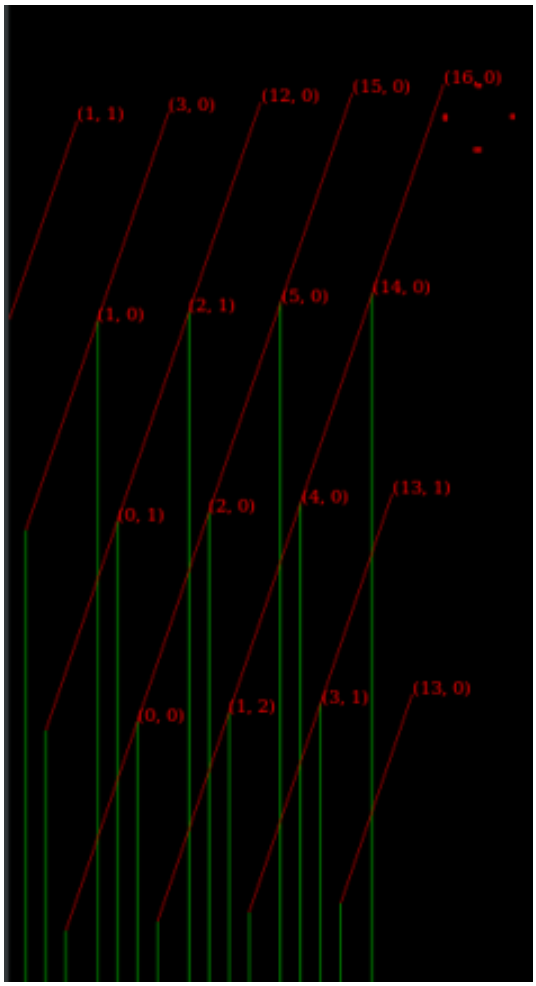


Figure 4.32: Positive steplines from Egeblad's algorithm for alpha = 10.

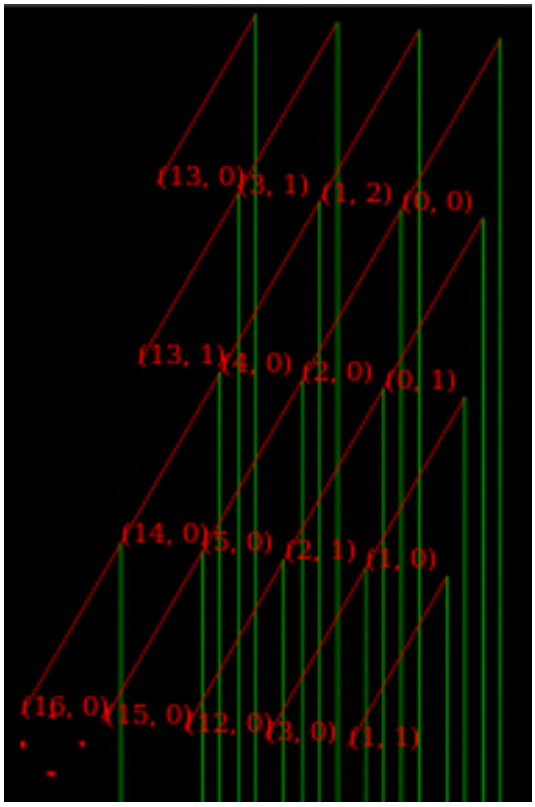


Figure 4.33: Negative steplines from Egeblad's algorithm for alpha = 10.

Chapter 5

Experimental Methodology

As mentioned in chapter 4, our work was integrated to an EDA environment developed by [28], called ASP. All development was performed in the C programming language. Due to its integration in such an environment, our work is capable of handling blocks with different properties. The blocks that can be handled range from standard library cells and modules to cell clusters, multilevel clusters and even design partitions.

All of the experiments and tests described in the following sections were performed in a server containing and Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz with 128GBs of RAM. The server's OS is CentOS 7 3.10.0-1160.99.1.el7.x86_64.

5.1 Experimental Setup

To test our work, we wanted a level playing field. Therefore we decided to perform all of our tests on the same benchmark. To showcase the integration with ASP, the benchmark is an AES core synthesized at a $0.25\,\mu\mathrm{m}$ commercial library, and consisted of 165 multi-level clusters.

Since we introduced a lot of optimisations, we decided to test our work in an additive manner. This means that we tested the optimisations one at a time, each time keeping the best one from the previous testing stage. Our aim with this was to determine by elimination, the best possible Sequence Pair optimisation setup. The way this was implemented was, by disabling all features from the optimisation loop as described in section 4.5 and re-enabling it in stages. Thus, comparing and aggregating the best methodologies across the various SA configurations discussed in section 4.3. In this manner we will be able to also deduce which SA methodology performs best in realistic conditions.

Because the nature of SA optimisation inherently introduces randomness to the process, we devised a setup to eliminate that as much as possible. Each test was performed a total of 12 times. This was done in order to remove the best and worst result as statistical outliers and averaging the other 10 results in the hopes of producing as accurate statistics as possible.

Every graph we will present from this point onwards is the average of the results we obtained with the outliers removed.

All of the performed tests performed 400k iterations with no target whitespace exit condition. The experiments also utilised the Revert-to-Best flag in the annealing loop. This meant that if the annealer performed a specified number of worsening moves in a row it would automatically revert back to the best solution to avoid being stuck in a local minimum. For our experiments this number was set to 250 worsening moves.

Below we will describe the order and attempt to interpret the results to the best of our abilities. All result graphs are presented together in section 5.3 below. In the results we will present below, we omit the execution time of the tests. Initially we were to include the execution time but when we measured it, we observed that all tests had finished in an average of 19.5 seconds. We expected to see significant differences in execution time across the different annealing methodologies since calculating the temperature involves a various degrees of computational complexity. However, the differences were not consistent across tests and could easily be attributed to cache misses and page faults out of our control. Thus, execution time results were not included.

5.2 Order of performed optimisations

Before we begin describing our order of optimisations, we need to mention that we did not include in this experimental setting the optimisation move introduced in algorithm 1. This was done on purpose since preliminary results showed that it lead to local minima that the optimiser usually is not able to escape.

We applied our optimisations in the following order.

Firstly, using the random Sequence Pair, as it was the most consistent form of starting Sequence Pair, in terms of preliminary results up to this point, we measured the effects of the newly introduced put moves. In this experiment all moves where equiprobable. In Figure 5.5 we can see a clear 2% benefit in whitespace by simply introducing the put moves to the optimisation loop. For this test, we only utilised the step linear annealing methodology as we felt it was enough to assess the effectiveness of the new moves.

Then we wanted to capitalize on the knowledge we gained about the move success rates in section 4.4. For that reason we compared 3 distinct approaches. In the first one we used equal probability for all moves, in the second one we used values we determined through the preliminary tests for the probability of each move and in the third one we also incorporated the DPA to the second approach. In Figure 5.7, the predetermined probabilities showed a clear and consistent advantage over the equiprobable setting ranging from 0.5% to 1.5% across all SA methodologies. The DPA approach also showed improvement over the second one although not as large and not as consistent. It is apparent that this method requires further parameter tuning.

Next we examined the effects of the different starting Sequence Pair solutions. For this test we also tested 3 methodologies while based in the DPA approach described above. The random Sequence Pair, the single column one, and the newly introduced one described in section 4.2, which creates rows of similar height. In Figure 5.6, as predicted by the preliminary tests, the random Sequence Pair outperforms consistently and across the board both other approaches by a clear margin of more than 1%. An interesting observation can be made about the newly introduced starting solution, which is by far the worst. This is of no surprise since initial testing showed that it provides the best initial solution, which can sometimes be acceptable as is. But this degree of order and optimality early on in the optimisation process does not allow the algorithm to hill climb and find better solutions, even against the single column starting Sequence Pair. It must be noted once again that the effectiveness of this method is highly dependent on the dimensions of the blocks participating in the floorplan.

Finally, we tested the the performance of the Create Out of Core Rows move we created in subsection 4.1.3. We performed 2 sets of experiments, based on the random starting Sequence Pair which was determined the best one above. One without the Create Out of Core Rows, and one where we utilised the new move in 2 places in the optimisation flow. Once in the beginning of the flow to gather inside of the core the blocks that were outside of it and once right after the annealing part of the loop exits and we enter greedy optimisation to achieve the same effect. This achieves the best of both worlds. We achieve a degree of high entropy while most of our cells remain mostly inside the core and have a starting solution before entering the optimisation loop that can still be largely affected from the SA optimisation. In preliminary tests, where a different experimental parameters and another benchmark were used, this showed incredible results of increased QoR, sometimes even upt to 3%. However as is evident in Figure 5.8, in this exact setting the method was severely outclassed by the default one. Our best theory as to why this was the case, once again is the methods reliance to the dimensions of the benchmark's floorplan blocks.

In our concluding remarks on the experiments, we have to mention the performance of the various SA methodologies. Over the course of the performed experiments, the Step Linear methodology stood as the sole winner, with an advantage over the competition ranging from 0.2% to slightly over 1.5% in certain cases.

Here we will take the chance to present some of our best results across all the tests we conducted throughout this thesis.

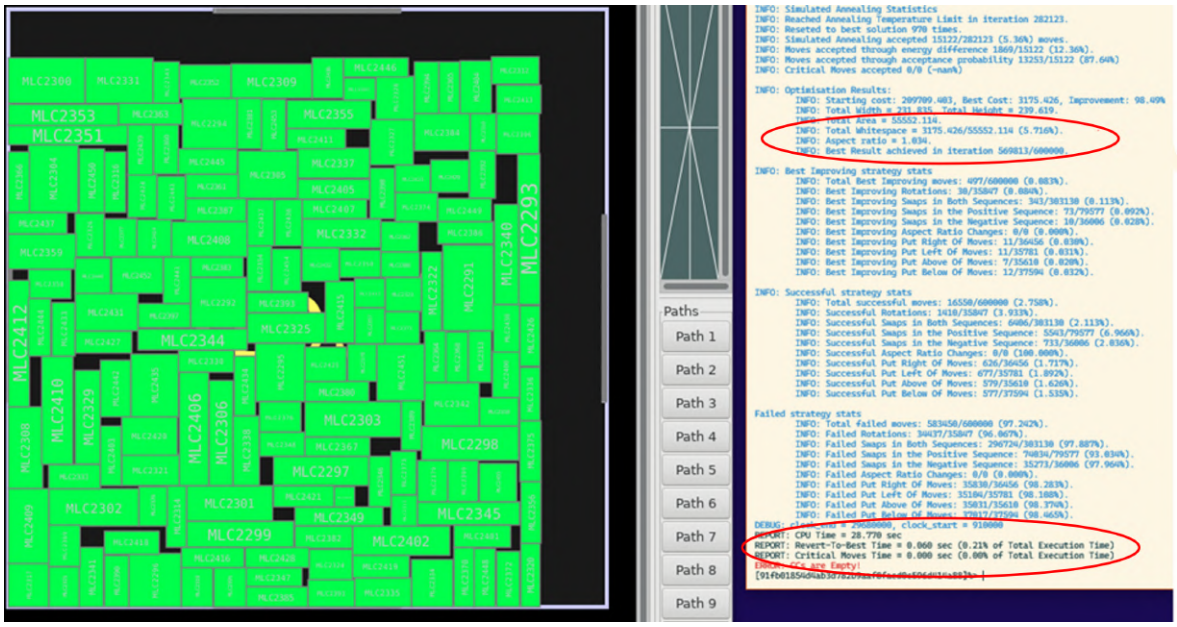


Figure 5.1: Some exquisite results part 1.

Figure 5.2: Some exquisite results part 2.

In this stage we have to mention that these parameters do not apply to a lot of other designs. When we applied our best methodology acquired below to a bigger benchmark we got the following floorplans. This benchmark consists of ∼700k standard cells and is partitioned to 2000 floorplan blocks through the ASP's partitioner. Here we can observe that there is a lot more empty space. This is mainly due to the fact that this 400k and 600k iterations respectively, where not enough to efficiently pack the larger design.

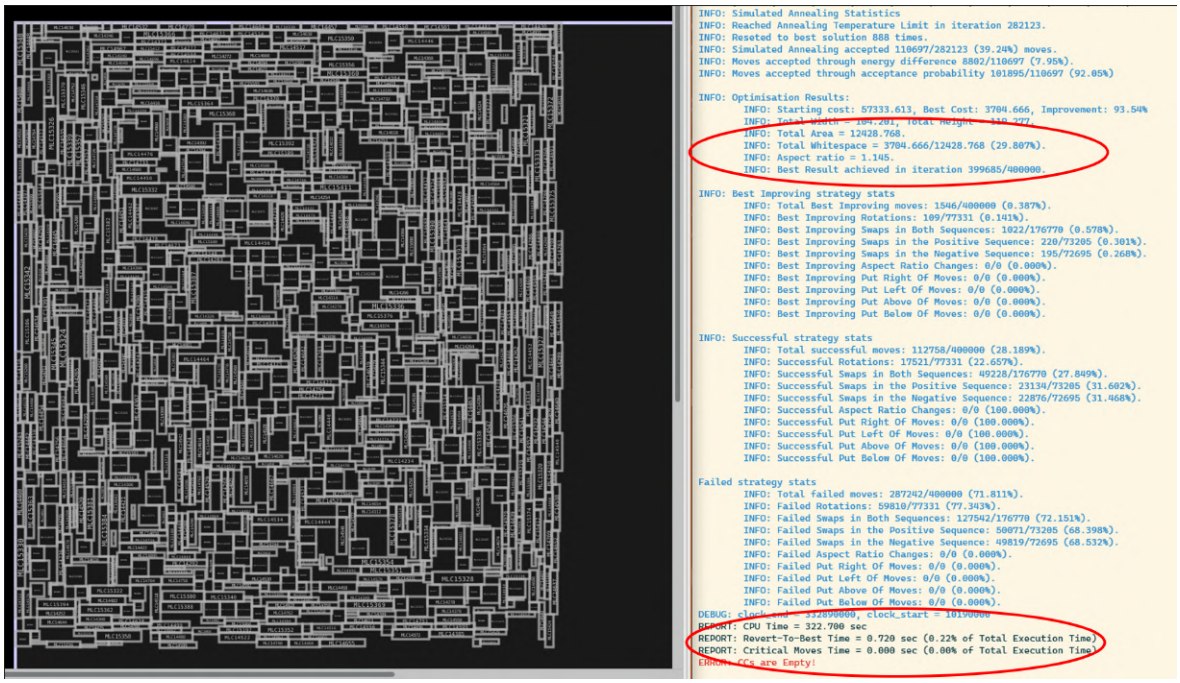


Figure 5.3: Results for a design consisting of 700k standard cells divided to 2000 partitions.

Figure 5.4: Results for a design consisting of 700k standard cells divided to 1000 partitions.

5.3 Results

In this section we will present the results for the experiments conducted in section 5.2.

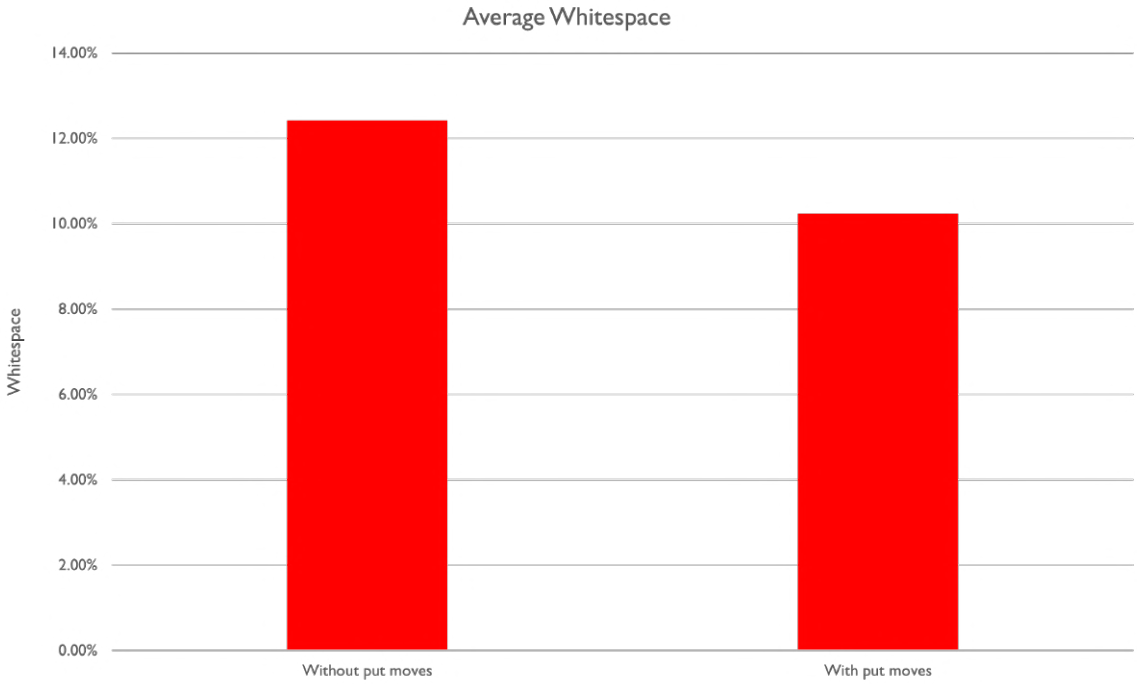


Figure 5.5: Put move effectiveness.

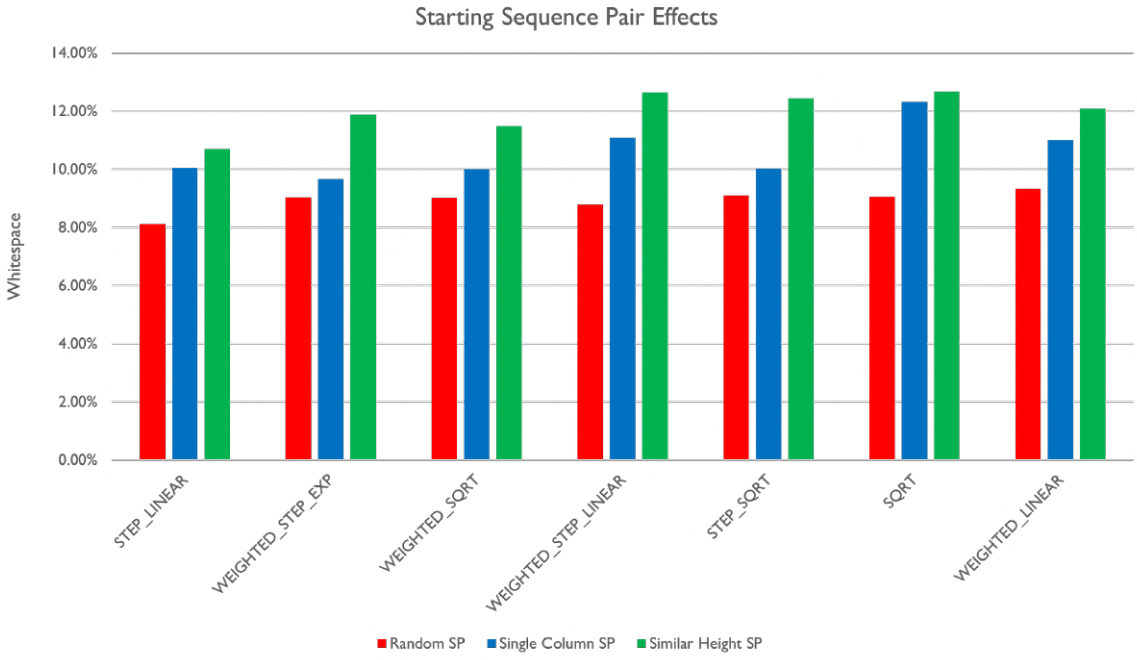


Figure 5.6: Effects of different starting Sequence Pairs.

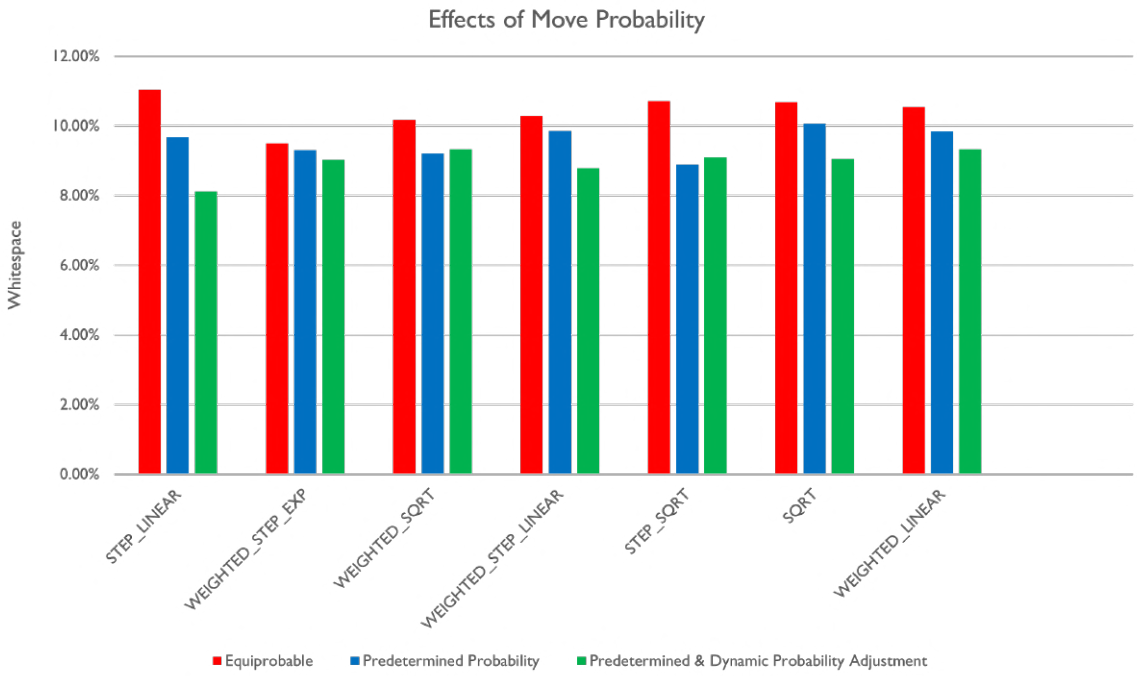


Figure 5.7: Effects of different move probability settings.

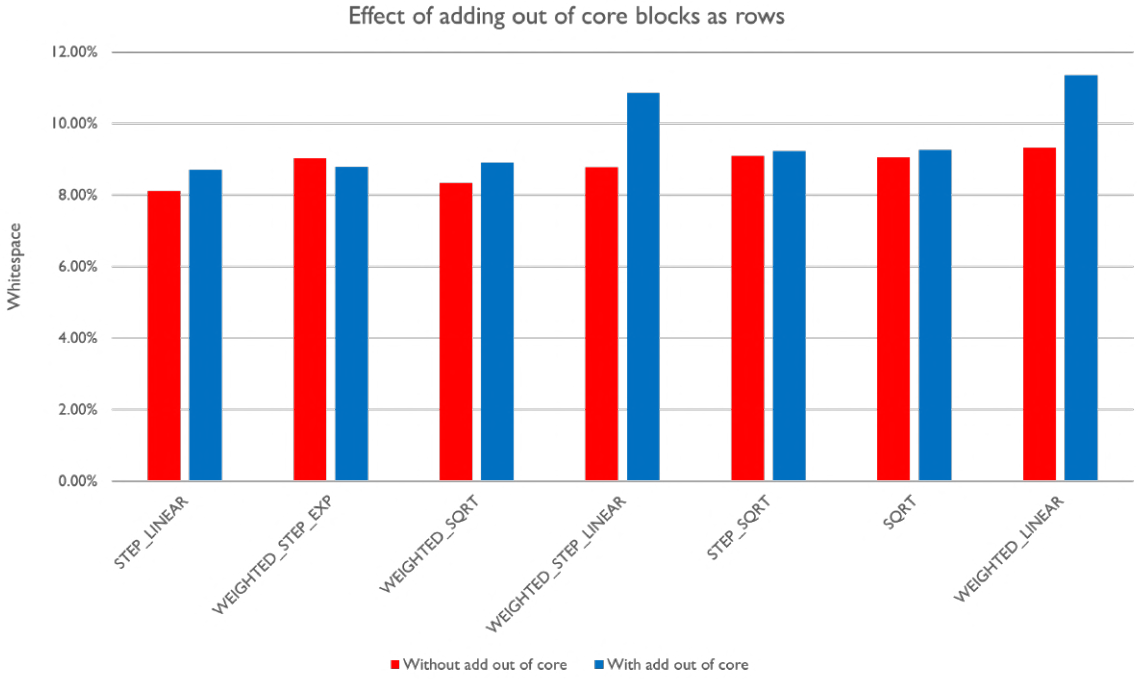


Figure 5.8: Effects of adding out of core blocks as rows.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

As technology continues to shrink and chip design grows ever more complex, there is no doubt in our minds that floorplanning's position in the ASIC design flow will become even more paramount. In this work we have provided a lot of interesting insight and intuition in the areas of Sequence Pair floorplanning and Simulated Annealing based optimisation. We presented novel ideas and expanded existing concepts to improve both the efficiency and QoR of algorithms that remain very popular to this day. We hope that this work will be the stepping stone for others to draw inspiration and expand.

6.2 Future Work

Albeit interesting, this work has left a lot to be desired. The biggest observation we can make is the fact that there is a lot of room for parameter tuning in our presented methodologies. Tuning for a large number of vastly different designs can be a daunting task.

Along with fine tuning SA parameters it may be worthwhile to pay closer attention to the work of Lester Ingber. His work on Adaptive Simulated Annealing and Reannealing [32], [30] and [31], has showed great promise.

In this work we solely focused our efforts in optimising the methodology for whitespace. However, this may not always be of interest. Already, our work supports various cost functions along with whitespace (total area and HPWL to name a few) but the optimiser has not been tuned to handle the different numerical magnitudes and is far from optimal. Preliminary results can be seen in Figure 6.1.

Nonetheless, these are not the only cost functions that interest the modern EDA industry. In literature we can already find works addressing more modern cost functions such as timing slack or thermal efficiency. Technology shrinking also presents another fascinating area, where Sequence Pair can be of great use, the world of 3D and 2.5D chip design. In this

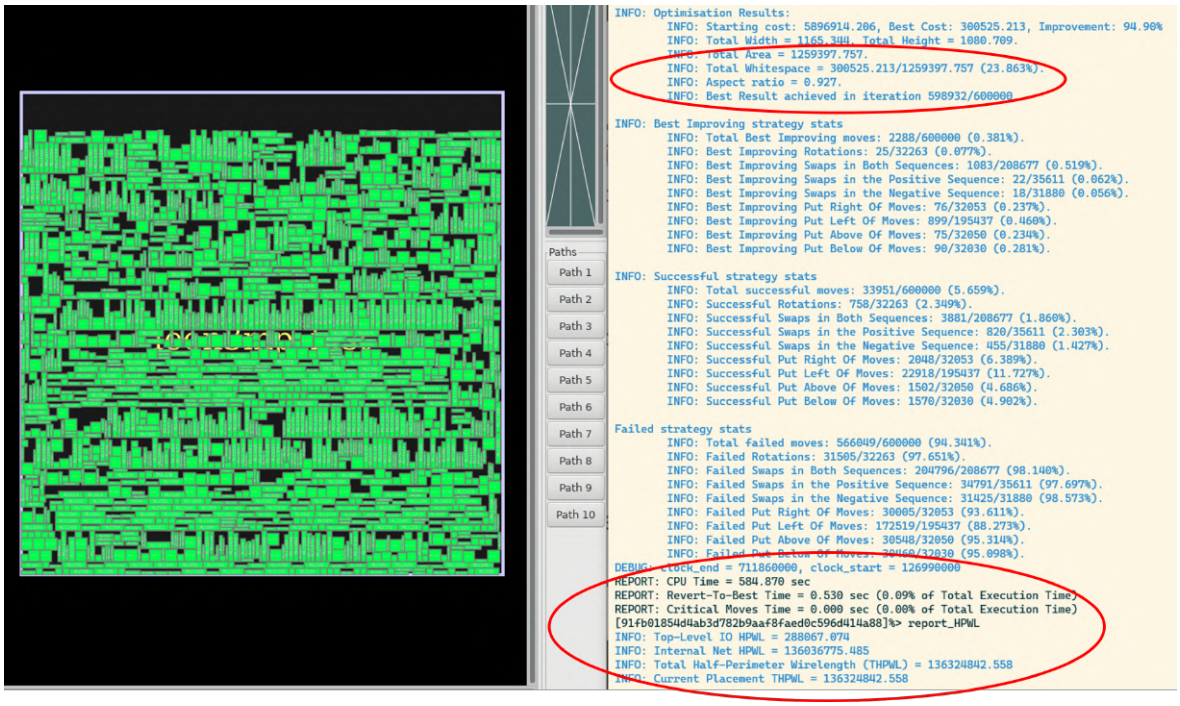


Figure 6.1: HPWL results from a 700k cell design consisting of 2000 partitions.

domain we can assign a different Sequence Pair per design tier and optimize each tier separately. Since the tiers are independent and the Sequence Pairs are completely independent such work can be trivially parallelized. This also clears the way for new optimisation moves, such as moving a block across different design tiers. This is as easy as removing the desired block from both sequences of the tier it currently resides, and moving it to the appropriate place in the both sequences of the target tier. Another move that easily comes to mind is swapping 2 blocks across tiers. This is equivalent to swapping the locations of the 2 blocks in both sequences in the desired tiers.

Some interesting work has been proposed in this field by works from Chiou et al. [37], who present an interesting parallel branch-and-bound approach and a unique Sequence Pair-tree formulation for Sequence Pair optimization, as well as in [24] where Prakash et al. propose the partitioning of the design and assigning a Sequence Pair per tier but use a weird set of moves that is not explained in great detail.

Work can also be done in the field of discovering new Sequence Pair moves. In [2] Kahng et al. at 3.4.1 mention floorplan wheels and provides a graph and Sequence Pair example of how a wheel is represented. This can be generalized and work with clusters that are heavily connected to massively improve the total wire length.

Since this is a manufactured example and blocks have one identical dimension the devil hides in the details. In fig 3.5, if blocks have different dimensions or blocks D and C have have different horizontal predecessors OR blocks C and F have different vertical predecessors there may be problems. We also need to compose such wheels by blocks of appropriate sizes, if we want them to have near-zero whitespace.

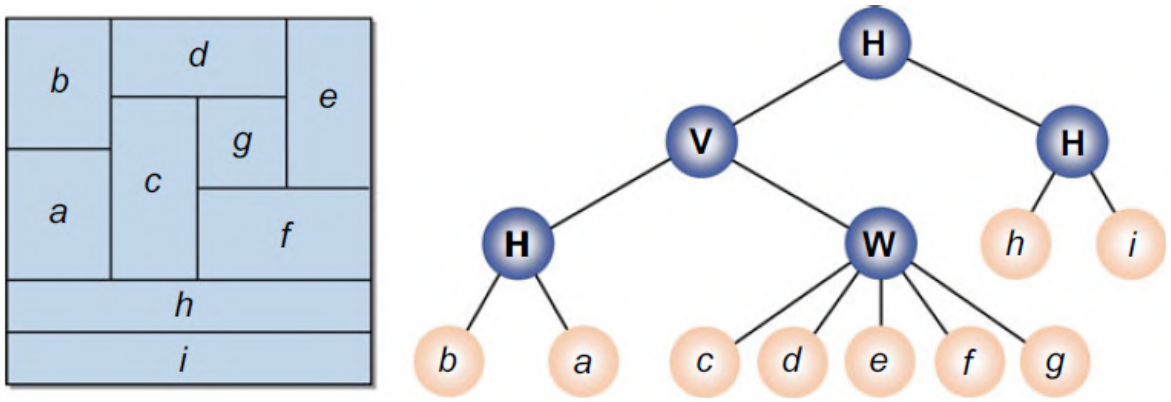


Figure 6.2: The concept of floorplanning wheels. Wheel sub-Sequence Pair (dcgef, cfgde)

Furthermore, it is of great importance that our work is compared against other floorplanning approaches to properly measure the effect of our contribution. ASP, CASLab's [28] EDA tool includes global placement and slicing floorplanning algorithms. It was in our plans to run experiments where we compare our work against the existing floorplanner but such work was cut from the scope of this thesis, due to timing constraints. This is an attainable goal for the future.

Finally, area that can greatly be improved is the decision making process. As we saw in section 4.4, only a small number of moves is required to achieve a near optimal floorplan. But as it stands our poor decision making is costing us in terms of the required number of iterations and thus execution time. The nature of the Sequence Pair algorithm makes it a prime candidate for the development of machine learning techniques to improve its decision making.

Bibliography

[1] Wikipedia contributors. Moore's law — Wikipedia, the free encyclopedia, 2023.

[2] Andrew B. Kahng, Jens Lienig, Igor L. Markov, and Jin Hu. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer International Publishing, 2022.

[3] Nikolaos Sketopoulos. *3D IC CAD Placement Flows and Algorithms Yielding Improved PPA*. PhD thesis, University of Thessaly, 2021.

[4] Bo Yao, Hongyu Chen, Chung-Kuan Cheng, and Ronald Graham. Revisiting floorplan representations. In *Proceedings of the 2001 international symposium on Physical design*. ACM, April 2001.

[5] Sung Kyu Lim. *Practical Problems in VLSI Physical Design Automation*. Springer Netherlands, 2008.

[6] Hiroyuki Yamazaki, Keishi Sakanushi, Shigetoshi Nakatake, and Yoji Kajitani. The 3d-packing by meta data structure and packing heuristics. *ACM Journal of Experimental Algorithms - JEA*, 83, 01 2000.

[7] Rajendra Bahadur Singh, Anurag Singh Baghel, and Ayush Agarwal. A review on vlsi floorplanning optimization using metaheuristic algorithms. *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 4198–4202, 2016.

[8] Mingxu Huo and Koubao Ding. A quick generation method of sequence pair for block placement. In *Lecture Notes in Computer Science*, pages 954–957. Springer Berlin Heidelberg, 2005.

[9] Jens Egeblad. Placement techniques for vlsi layout using sequence-pair legalization, 2003.

[10] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1518–1524, 1996.

[11] Mingxu Huo and Koubao Ding. An improved algorithm for sequence pair generation. In *Computational Science – ICCS 2006*, pages 482–489. Springer Berlin Heidelberg, 2006.

[12] Evangeline Young. Floorplan representations. In *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications, nov 2008.

[13] Adam Teman. Lecture 6: Moving to the physical domain, 2018.

[14] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. Rectangle-packing-based module placement. In *The Best of ICCAD*, pages 535–548. Springer US, 2003.

[15] Xiaoping Tang, Ruiqi Tian, and D. F. Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Proceedings of the conference on Design, automation and test in Europe*. ACM, January 2000.

[16] Xiaoping Tang and D. F. Wong. FAST-SP. In *Proceedings of the 2001 conference on Asia South Pacific design automation - ASP-DAC '01*. ACM Press, 2001.

[17] Donald B. Johnson. A priority queue in which initialization and queue operations takeO(loglogD) time. *Mathematical Systems Theory*, 15(1):295–309, December 1981.

[18] A. Kozik. Fully dynamic evaluation of sequence pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):894–904, June 2013.

[19] Hiroshi Murata and Ernest S. Kuh. Sequence-pair based placement method for hard/soft/pre-placed modules. In *Proceedings of the 1998 international symposium on Physical design*. ACM, April 1998.

[20] Gang Huang, Xianlong Hong, Changge Qiao, and Yici Cai. A timing-driven block placer based on sequence pair model. In *Proceedings of the ASP-DAC '99 Asia and South Pacific Design Automation Conference 1999 (Cat. No.99EX198)*. IEEE, 1999.

[21] Norihide Okada, Chikaaki Kodama, Takashi Sato, and Kunihiro Fujiyoshi. Thermal driven module placement using sequence-pair. In *APCCAS 2006 - 2006 IEEE Asia Pacific Conference on Circuits and Systems*. IEEE, December 2006.

[22] J. Cong, Jie Wei, and Yan Zhang. A thermal-driven floorplanning algorithm for 3d ICs. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.* IEEE, 2004.

[23] Jai-Ming Lin and Yao-Wen Chang. TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans. In *Proceedings of the 38th conference on Design automation - DAC '01*. ACM Press, 2001.

[24] Atul Prakash and Rajesh Kumar Lal. Simultaneous optimization of the area, wirelength and TSVs in a 3d IC design. *Sādhanā*, 47(4), December 2022.

[25] The OpenROAD project. `https://theopenroadproject.org/`.

[26] Dipanjan Sengupta, Andreas Veneris, Steve Wilton, Andre Ivanov, and Res Saleh. Sequence pair based voltage island floorplanning. In *2011 International Green Computing Conference and Workshops*. IEEE, July 2011.

[27] Lalin L. Laudis, Shilpa Shyam, Caroline Jemila, and V Suresh. MOBA: Multi objective bat algorithm for combinatorial optimization in VLSI. *Procedia Computer Science*, 125:840–846, 2018.

[28] Circuits and Systems Lab. `https://caslab.e-ce.uth.gr/`.

[29] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[30] L. Ingber. Very fast simulated re-annealing. *Mathematical Computer Modelling*, 12(8):967–973, 1989.

[31] L. Ingber. Adaptive simulated annealing (ASA): lessons learned. *Control and Cybernetics*, 25(1):33–54, 1996.

[32] Adaptive Simulated Annealing. `https://www.ingber.com/#ASA`.

[33] Jason Brownlee. Simulated annealing from scratch in python, 2021.

[34] `https://www.desmos.com/calculator`.

[35] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[36] Simulated annealing acceptance probability analysis. `https://www.desmos.com/calculator/twqnccnfbp`.

[37] Hong-Wen Chiou, Jia-Hao Jiang, Yu-Teng Chang, Yu-Min Lee, and Chi-Wen Pan. Chiplet placement for 2.5d IC with sequence pair based tree and thermal consideration. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*. ACM, January 2023.