Electronic Design Automation Algorithms for Standard Cell Legalization in Microelectronic Circuits

Master Thesis by

Nikolaos K. Sketopoulos

University of Thessaly

Department of Electrical and Computer Engineering

Supervisor:
Dr. Christos Sotiriou, Associate Professor, University of Thessaly
Committee:
Dr. George Stamoulis, Professor, University of Thessaly
Dr. Nestor Eumorfopoulos, Assistant Professor, University of Thessaly

Volos, Greece
October 2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αλγόριθμοι Ηλεκτρονικού Αυτοματισμού για Νομιμοποίηση Στοιχείων σε Μικροηλεκτρικά Κυκλώματα

Διπλωματική Εργασία
για την Απόκτηση Μεταπτυχιακού Διπλώματος Σπουδών
του:

Νικόλαος Κ. Σκετόπουλος

Επιβλέποντες:
Δρ. Χρήστος Σωτηρίου, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Θεσσαλίας
Δρ. Γεώργιος Σταμούλης, Καθηγητής, Πανεπιστήμιο Θεσσαλίας
Δρ. Νέστωρ Ευμορφόπουλος, Επίκουρος Καθηγητής, Πανεπιστήμιο Θεσσαλίας

Εγκρίθηκε από τη τριμελή εξεταστική επιτροπή την 26 Οκτωβρίου 2016

.....	.....	.....
Χρήστος Σωτηρίου	Γεώργιος Σταμούλης	Νέστωρ Ευμορφόπουλος
Αναπληρωτής Καθηγητής	Καθηγητής	Επίκουρος Καθηγητής

Acknowledgments

To my family, my friends & Dr. Sotiriou

Nikolaos Sketopoulos
Volos, Greece
26/10/2016

CONTENTS

LIST OF FIGURES

Abstract

Nowadays EDA tools use both combinatorial and analytical methods to place circuits' components. However, analytical methods render placement illegal. This phenomenon occurs because analytical methods use components as physical points. Thus, after global placement, components may overlap each other and receive non-aligned positions in the circuit's grid. For this reason, legalization is used by eliminating components' overlapping and aligning them in the circuit's grid. The aim of legalizers is to minimize the components' movement from their positions at global placement.

In this work, implementation, optimization and evaluation of a legalizer are presented. We present a novel evolution of fundamental Abacus legalizer [10]. Abacus has been chosen due to its great performance in terms of minimizing the perturbation of the optimal solution. However, the fundamental algorithm supports legalization only at flat circuits and without blockages. In this way, the fundamental legalization algorithm has been extended to support not only flat circuits, but also hierarchical and circuits with blockages. Moreover, fundamental Abacus legalizer, can not handle components which height are greater than the placement row. As a sequence, we modify and tune Abacus2 also to support cells with different heights than the placement row height.

Additionally, different approaches for the individual stages of legalization have been implemented, such as the calculation of the components' movement cost. Execution time of legalization has been optimized by using heuristic algorithms and legalization with blockages have also been studied. We performed experiments and comparisons between the features of Abacus and Abacus2. Finally, the legalizer has been integrated in developing an industrial EDA tool, taking its restrictions into account.

Περίληψη

Τα σημερινά εργαλεία EDA πραγματοποιούν την τοποθέτηση των στοιχείων τόσο με συνδυαστικούς, όσο και με αναλυτικούς μεθόδους. Ωστόσο, οι αναλυτικοί μέθοδοι καθιστούν την τοποθέτηση μη έγκυρη. Το φαινόμενο αυτό παρουσιάζεται διότι οι αναλυτικοί μέθοδοι μεταχειρίζονται τα στοιχεία ως σημεία στο χώρο. Έτσι, μετά την τοποθέτηση παρατηρούνται επικαλύψεις μεταξύ των στοιχείων και επιπλέον τα στοιχεία λαμβάνουν μη ευθυγραμμισμένες θέσεις στο πλέγμα του κυκλώματος. Για το λόγο αυτό πραγματοποιείται η νομιμοποίηση των στοιχείων εξαλείφοντας τις επικαλύψεις και ευθυγραμμίζοντας τα στοιχεία στο πλέγμα του κυκλώματος. Η διαδικασία αυτή πραγματοποιείται από τους νομιμοποιητές. Στόχος των νομιμοποιητών είναι η ελαχιστοποίηση της μετακίνησης των στοιχείων από τις βέλτιστες θέσεις που έλαβαν από την γενική τοποθέτηση.

Στη παρούσα διπλωματική διατριβή παρουσιάζεται η υλοποίηση, η βελτιστοποίηση και η αξιολόγηση ενός αλγορίθμου για τη νομιμοποίηση των στοιχείων ενός κυκλώματος, ο οποίος ονομάζεται Abacus2. Ο νομιμοποιητής αυτός αποτελεί εξέλιξη και επέκταση ενός ευρέως διαδεδομένου νομιμοποιητή, του Abacus [10]. Η επιλογή του, πραγματοποιήθηκε λόγω των καλών επιδόσεων σε ό,τι αφορά την ελάχιστη τροποποίηση της βέλτιστης λύσης. Ωστόσο, ο κλασικός αλγόριθμος υποστηρίζει τη νομιμοποίηση μόνο επίπεδων κυκλωμάτων και χωρίς εμπόδια. Επομένως, μελετήθηκαν και υλοποιήθηκαν προσεγγίσεις για την υποστήριξη ιεραρχικών κυκλωμάτων και κυκλωμάτων με εμπόδια. Επιπλέον, Επιπλέον, ο θεμελιώδης αλγόριθμος δε μπορεί αν χειριστεί κυκλώματα των οποίων τα στοιχεία έχουν ύψος, το οποίο είναι πολλαπλάσιο του ύψους των γραμμών του κυκλώματος. Η αδυναμία αυτή, μας ώθησε στην περεταίρω ανάπτυξή του αλγορίθμου για να μπορεί να αντιμετωπίζει τέτοια στοιχεία.

Επιπλέον, υλοποιήθηκαν διαφορετικές προσεγγίσεις για τα επιμέρους στάδια της νομιμοποίησης, όπως για παράδειγμα του υπολογισμού του κόστους μετακίνησης των στοιχείων. Βελτιστοποιήθηκε ο χρόνο εκτέλεσης του αλγορίθμου χρησιμοποιώντας ευριστικούς αλγόριθμους και μελετήθηκε η υποστήριξη κυκλωμάτων με εμπόδια. Πραγματόποήθηκαν πειράματα για να ελεχθεί η ποιότητα της τελικής έγκυρης τοποθέσης. Επιπρόσθετα πραγματοποιήθηκαν συγκρυσεις μεταξύ του Abacus και του Abacus2. Τέλος, ο νομιμοποιητής ενσωματώθηκε σε ένα υπό ανάπτυξη βιομηχανικό εργαλείο EDA, λαμβάνοντας υπόψιν τους βιομηχανικούς περιορισμούς.

CHAPTER 1

Introduction to EDA

Integrated circuits (ICs) have had an astonishing effect on our everyday life as there are vital parts of conveniences such as cell phones, personal computers, navigation systems and music players, just to name a few. In fact, almost everything and every daily task has been influenced by ICs. The modern integrated circuits are among the most complex products ever built by humans. Moreover, the number of transistors per integrated circuit has been doubled almost every two years, following the Moore's Law. So, the design of very large-scale integrated (VLSI) circuits, has become very challenging, inspiring designers to develop electronic design automation (EDA) tools. The aim of EDA tools includes area and power minimization, circuit performance optimization, and manufacturability `e.t.c.`.

EDA is a software which helps engineers to create new ICs. EDA tools have always been focusing on automating the entire circuit design process and combining the design tasks into a complete design flow. Due to the high complexity of modern designs, EDA handles many aspects of the IC design flow. However, such integration is challenging, since some designing tasks need additional degrees of freedom, and scalability requires tackling some tasks independently. On the other hand, the constant decrease of transistors and wire dimensions have obscured not only the boundaries, but also the abstractions that separate successive designing tasks. That is, EDA tools are mostly used in automated design tasks such as logic design, physical design, simulation and verification.

1.1 Placement

Circuit placement is one of the most important tasks of EDA tools. After partitioning the circuit into smaller modules and floorplanning the layout to determine block outlines and pin locations, placement aims to determine the locations of logic components within each block. The placement's main objective is to optimize wirelength, timing, and congestion, thermal hotspot and power consumption [7]. If logic cells are not all exactly the same in size, then the physical size of each cell must be known so that placement does not overlap with the cells in

the layout. Some standard cell systems support large array macros (soft and/or hard) such as RAMs. The placement of these components is troublesome for the automated procedure, so these macros might have to be manually placed. Because the locations of circuit components and corresponding interconnect delays are determined during the placement procedure, it has notable impact on the final performance of the circuit [1].

In the placement process, there is no single cost function or trivial algorithm that guarantees success. Hence, it is crucial to choose the right algorithm to optimize the right cost function at the right time. This needs a deep understanding of different aspects for the placement problem. Today's placement tools use this strategy, but in an ad-hoc way. Fundamental research is required to devise the methodology which systematically suggests the solution to the placement problem [9].

In order to handle large-scale circuits, placement is usually done in three tasks, Figure 1.2: (i) global placement, (ii) legal placement (or legalization) and (iii) detailed placement. Global placement is mainly concerned with the location of the cells, e.g., which region of the chip a cell is located. Some cells may be overlapping with each other in a global placement. These overlaps are then removed during legal placement and local optimizations are done during detailed placement.

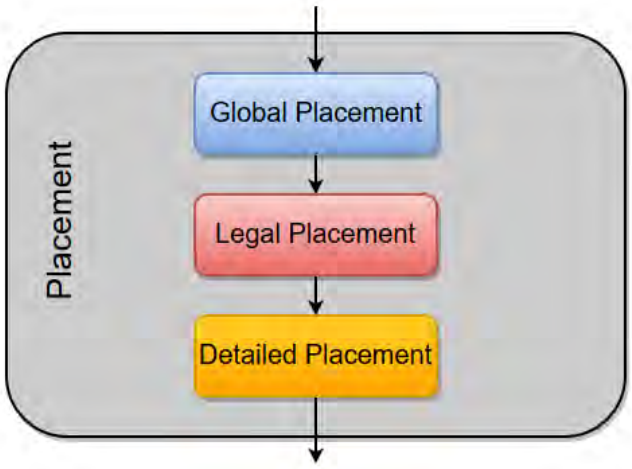


Figure 1.1: Placement Flow

1.1.1 Global Placement

There are many different methodologies which can be used to find where a cell must be placed in the global placement stage, like methods which are based on a simulated annealing, top-down cut-based partitioning, or analytical techniques [1].

Simulated annealing is an iterative optimization method that has been inspired by the physical metal cooling process. With the given objective function, the process tries to achieve

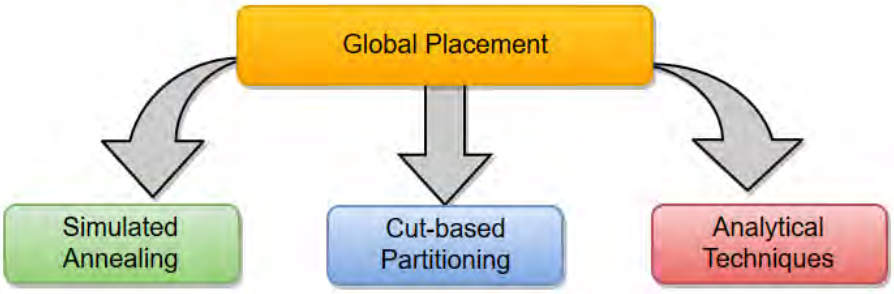


Figure 1.2: Global placement Algorithm Methodologies

a better solution via a set of predefined moves. The move which achieves a better solution, is always accepted. If a move produces a worse solution, it is accepted based on some probability functions. At early stages (with high temperature), a bad move has a higher chance to get accepted while at later stages of placement (with lower temperature), the probability is reduced. These worse-yet-accepted moves are essential for a simulated annealing placement algorithm to overcome a local optimum solution where a placement might be stuck. When a greedy move-based placement method steps into local optimum, it cannot escape from this sink.

Top-down cut-based partitioning placers, partition circuit area into either two or four regions, then recursively partitions each region until a good coarse placement solution is achieved. When each region is partitioned, every circuit component outside the region is assumed to be fixed at the current location and pseudopins are created around the region under consideration. This is called a terminal propagation. Because the main algorithm is based on partitioning, the typical objective function is the number of netcuts between sub-regions, `i.e.` the cut-size. Finding a good partitioning indicates that good logical clustering of circuit elements are found with less cut-size among them that can lead to a better total wirelength (TWL). In general, cut-based multilevel partitioning placement can be performed quite well when designs are dense. Moreover, partitioning-based placement is a relatively fast placement algorithm, as the placement problem is subdivided into smaller placement problems with less parameters.

The main idea of analytical placement is based on first placing the cells optimally, in terms of wirelenght estimation and then working toward disjointness. The second task, aims to modify the objective function in small steps so as to force cells to move away from each other. Such force-directed approaches reduce overlaps by recursive partitioning of the chip area resulting in the set of cells to be placed in the core area. This partitioning is done in such a way that no sub-region of the chip area contains more cells than it can fit. Consequently, when the regions are small enough, the cells will be spread over the chip area. There are plenty of techniques for cell spreading in force-based analytic placement techniques. During placement, some form of density analysis is performed to calculate spreading forces. Once the spreading forces are determined, these forces can be applied to each circuit component

via forces.

1.1.2 Legalization

Global placement assigns locations to standard cells and larger circuit modules, e.g. macro blocks. However, these locations do not align with the circuit placement grid, and may have continuous coordinates rather than discrete coordinates. Therefore, a legalization step must be performed. The allowed legal locations are equally spaced within placement rows, and the positions from global placement should correspond to the closest possible legal position [6]. Legalization is necessary not only after global placement, but also after incremental changes as cell resizing and buffer insertion during physical synthesis. Legalization tries to find legal, non-overlapping placements for all cells so as to minimize its impact on wirelength, timing and other design objectives as little as possible. Unlike algorithms for cell spreading during global placement, legalization typically assumes that the cells are sufficiently distributed throughout the core area and have relatively small mutual overlaps.

Some algorithms for legalization and placement are co-developed with global placement algorithms. For instance, in the context of min-cut placement, detailed placement can be performed by optimal partitioners and placers invoke in very small bins that are produced after the netlist is repeatedly partitioned. Given that these bins contain a small number of cells, optimal locations can be found by exhaustive position search. For larger bins, partitioning can be performed optimally. Some analytic algorithms perform legalization in iterations [3]. At each iteration, cells closest to legal sites are identified and snapped to legal sites, then they are considered fixed thereafter. After a round of analytic placement, another group of cells is snapped to legal sites, and the process continues until all cells have been given legal locations. A common problem with simple and fast legalization algorithms is that some cells may travel a long distance, thus significantly increasing the wirelength and, hence, delaying the incident nets. This phenomenon can be eased by detailed placement.

1.1.3 Detailed Placement

Once a legal placement is available, it can be improved with respect to a given objective by means of detailed placement techniques, such as swapping neighboring cells or sliding cells to one side of the row when unused space is available, to reduce total wirelength . Some detailed placers target routability, given that route topologies can be determined once a legal placement is available.

CHAPTER 2

Background

Many legalization strategies have been proposed throughout the years trying to minimize the impact on the legalizer as little as possible. These strategies can be classified as (i) global or (ii) local approaches [8]. The main difference between these approaches is that the former legalizes groups of standard cells simultaneously, while the latter legalizes one standard cell at a time. Table 2.1 presents additional techniques which are widely used directly or combined in modern legalizers.

Legalization Techniques	Greedy moves to free locations
	Ripple Cell Movement
	Diffusion Based
	Dynamic Programming
	Computational Geometry
	Network Flow
	Linear Programming
	Top-Down Opt. & Clustering

Table 2.1: Legalization Techniques [8]

2.1 Global Legalization Approaches

Global legalization approaches are applied, mainly, in network flow techniques or similarly in maximum bipartite matching to get a direction guideline in which cells have to be moved. The main idea of these approaches is to exploit the global view of the cells' positions and guide them to positions avoiding local optima [2]. In this way, the placement area is subdivided into regions or bins and cells are moved from dense to sparse regions by solving a transportation problem, where the nodes are the regions and the cells, the edges are the matching between cells and regions, and the edges' weights are the movement cost of the cells.

Brenner [2] subdivides the chip area into bins and assigns cells to bins. Then, in order to manage bins which exceed their cell capacity, a network flow problem is solved to distribute cells between bins, while achieving minimum total cell displacement. Doll et al. [4] propose an iterative approach, whereby the chip area is divided into overlapping regions. For a given order of the regions, a minimum-cost, maximum-flow problem is then solved, per region, to identify the cells with minimum local displacement, which may be legalized per row. Another iteration is then performed with different region orders, until no improvement is achieved.

The drawback of global legalization algorithms is its high complexity, and the fact that flow models estimate the cost of moving cells between regions. In the case that the estimation is inaccurate, the end result may be suboptimal. Moreover, the majority of global legalization approaches do not lead directly to a legal placement. Although the cells have been spread in the placement core area, many cells may continue overlapping. So, a final legalization step must be taken to assign the cells to the placement rows without overlaps.

2.2 Local Legalization Approaches

On the other hand, local approaches, like Tetris [5] and Abacus [10], legalize one cell after another by using mostly greedy decisions. Each cell is selected to be legalized based on an order. Cell order may depend on the cell GP position, the cell area and the cell influence on the critical path `e.t.c.`. The cells' order significantly influences the legal result, as each alternative order may lead to different legal placements. A legal position is selected in order to minimize the GP perturbation as little as possible. The next sections present the two most common local approaches, Tetris and Abacus legalizers.

2.2.1 Tetris Legalizer

Tetris [5] is a greedy and sequential legalizer which handles mixed cells, `i.e.` standard cells and macroblocks. This method is remarkably simple and trivial to be implemented. Algorithm 1 describes how Tetris works. For the simplicity of the above pseudocode, let's assume that a component is either a standard cell or a macroblock.

Tetris, first assumes a virtual grid which corresponds to the available positions, x and y, where each component can be placed. Actually, the coordinates x and y are determined by "Library Exchange Format (LEF)" files which depend on current technology. x coordinates are based on the vertical core area sites and y coordinates on the horizontal, which is actually the placement rows. Then, the components are legalized one at a time, lines: 3-14. For each component C_i, all the available positions (x, y) are checked for each position where there is no overlapping with pre-placed components and cost D is determined, lines: 3-8. Tetris aims to place each component to the nearest GP position without overlaps. As a consequence, it's cost function is the displacement between the GP position and the trial legal position. If the current component's displacement is less than the minimum, then the best cost is updated, lines: 9-11. When all available positions are checked, then the current component C_i is assigned to the legal position (x_{best}, y_{best}) with the lowest displacement cost, $best\_cost$, line:

Algorithm 1: Tetris

```
1  create (x, y) grid;
2  cell_ordering();
3  for each component C_i do
4  |  best_cost = ∞;
5  |  for each x do
6  |  |  for each y do
7  |  |  |  if component_fits_at_position(C_i, x, y) then
8  |  |  |  |  Determine cost D;
9  |  |  |  |  if D < best_cost then
10 |  |  |  |  |  x_best = x, y_best = y;        /* Trial */
11 |  |  |  |  |  best_cost = D;
12 |  |  end
13 |  end
14 |  Assign C_i to (x_best, y_best) ;            /* Final */
15 end
```

14.

In its purest form, Tetris has several known drawbacks [6], one being its obliviousness to the netlist, because the cells' relative GP order is not maintained. As a consequence, if component a is on the left of (or above) b in GP, then component a may be legalized on the right of (or below) b, leading to greater total displacement cost and total wirelenght. Another drawback is that in the presence of large amounts of whitespace, once a module is legalized, it will not be moved anymore. These main drawbacks are solved by several evolutions of Tetris, like Abacus.

2.2.2 Abacus Legalizer

Abacus [10] is also a sequential and greedy algorithm which is more effective, as for total wirelength, than Tetris. In contrast to Tetris, Abacus legalizes only standard cells with the same height, but different width, trying to minimise their displacement (movement) from the GP positions. In order to achieve the minimal total displacement, standard cells are allowed to be moved through placement rows by keeping their initial global placement order. Quadratic and dynamic programming is used to find the cells' position with the minimum displacement from their GP positions. The following sections describe the most important features of Abacus legalizer.

2.2.2.1 Quadratic Program

Abacus tries to move the cells as little as possible, in order to minimise it's influence on the optimal global placement solution. In this way, pre-placed cells are allowed to be shifted

through its placement rows to optimize the total quadratic movement of all cells within one row. The new positions are found by the following quadratic program:

$$min \sum_{i=1}^{C_r} [x(i) - x'(i)]^2 \tag{2.1}$$

$$s.t. \quad x(i) \geq \ x(i-1) - w(i-1) \ \ i = 2, ..., C_r \tag{2.2}$$

For Equations 2.1 and 2.2 we assume that the row has C_r cells and for each cell i we have the following properties, the initial global placement x-coordinate $x'(i)$, the legal placement x-coordinate $x(i)$ and the width $w(i)$. Besides this, the cell selection order is known. So, cells i and $i-1$, equation $x(i) \geq \ x(i-1)$ must be satisfied, so as to keep the cells initial order. Objective function 2.1 presents the total squared displacement of all row cells between the global and legal positions. The objection can be weighted, like Equation 2.3, where $e(i)$ is a weight parameter for cell i. This parameter can take the cell's area, cell's connections, `e.t.c.` into account.

$$min \sum_{i=1}^{C_r} e(i) * [x(i) - x'(i)]^2 \tag{2.3}$$

Constraint 2.2 guarantees that there is no overlapping between the two cells i and $i-1$. Additionally, this constraint ensures the cells initial order maintenance. However, the solution of Objective function 2.3 with Constraints 2.2 is time consuming. Abacus faces this problem by solving the system with "=" constrains. In this way, the system is solved very fast, but cells must abutt to satisfy "=" constrain. So, 2.2 is transformed to:

$$x(i) = x(1) + \sum_{k=1}^{i-1} w(k) \quad i = 2, ..., C_r \tag{2.4}$$

By, using 2.2 in 2.3 the quadratic function depends only on $x(1)$ and 2.3 can be re-written as:

$$\sum_{i=1}^{C_r} e(i) * x(1) - \left[e(1) * x'(1) + \sum_{i=2}^{C_r} e(i) * \left[x'(i) - \sum_{k=1}^{i-1} w(k) \right] \right] = 0 \tag{2.5}$$

Then the optimal position $x(1)$ of cell $i = 1$ is given by 2.6 and optimal positions $x(i)$ of the remaining cells in the row are given by Equation 2.4.

$$x(1) = \frac{e(1) * x'(1) + \sum_{i=2}^{C_r} e(i) * \left[x'(i) - \sum_{k=1}^{i-1} w(k) \right]}{\sum_{i=1}^{C_r} e(i)} \tag{2.6}$$

2.2.2.2 Displacement Cost Function

Abacus, tentatively places each cell to all or to a number of the placement rows, until the best row is found. The best row is the row with the minimal displacement cost. The displacement cost is determined by the movement of cell i between its global and legal placement position. Equation 2.7 shows the Euclidean displacement cost function of Abacus, for cell i legalization,

$$\sqrt{(x(i) - x'(i))^2 + (y(i) - y'(i))^2}, \tag{2.7}$$

where $\{x(i),\ y(i)\}$ and $\{x'(i),\ y'(i)\}$, the legal and global placement coordinates for cell i.

2.2.2.3 Row Search Bounding

In order to find the row where the cell displacement cost is minimal, Abacus theoretically will try to legalize each cell in each placement row. However, this is time consuming even for small designs. In this way, this algorithm eliminates the number of the row with bounds. The bounds depend on the current best displacement cost. It is pointless to search for rows where their distance is greater than the best displacement cost. So, each cell is trial legalized in it's nearest row depending on it's global placement y-coordinate and then the displacement cost is determined. This algorithm will not try to legalize the cell in any row, which vertical distance from the nearest row is greater than the calculated best displacement cost. The aim of row search bound is only to reduce the execution time of legalization.

2.2.2.4 Cells Selection Order

Similar to Tetris, Abacus is a sequential algorithm, `i.e.` cells must be sorted in a specified order and then they are placed in the core area, one by one. In Abacus, Spindler suggests using the increasing and decreasing order. Abacus, tries to keep this initial cell order by supposing that the maintenance of the order will lead to the least GP perturbation, `i.e.` to minimal total cells displacement.

2.2.2.5 Abacus Algorithm

On the top level Algorithm 2, cells are sorted either in increasing or in decreasing order based on their GP x-coordinates, line: 2. In this way, each cell in the specified order, is trial legalized in a number of rows and finally placed in the row with the minimum displacement cost, lines: 2-15. In the case that blockages are presented, the algorithm slices the placement rows to sub-rows, so that all new sub-rows are blockage free. The current cell is inserted in each available (sub-)row and `PlaceRow` function is used to find the new cells' position, lines: 6-7. When the cell position is found, the displacement cost is determined, line:8. In the case that new displacement cost is better than the best, Abacus updates the best cost and row. Next, the number of the available rows is updated.

Algorithm 2: Abacus Top Level Algorithm

```
1  cell_ordering();
2  for each cell i do
3      B = all placement rows;
4      c_best = ∞;
5      for each row r in B do
6          Insert cell i into row r;
7          PlaceRow r (trial);
8          Determine cost c;
9          if (c < c_best) then
10             cbest = c;
11             rbest = r;
12         B = min(cost_to_rows(D), B);
13         Remove cell i from row r;
14     end
15     Insert Cell i to row r_best;
16     PlaceRow r_best (final);
17 end
```

Algorithm 3, describes the dynamic programming implementation, where Abacus finds the optimal legal position for each cell. Overlaps with other cells are detected, for each cell in the current row. If there is no overlapping, `i.e.` $x_c(c) + w_c(c) \leq x'(i)$, a group or cluster of cells is created. The cells of a group will be abutted and all these cells will be considered as one. The width of the group $w_c(c)$ is the sum of the widths of the cells that belong to this group, the $x_c(c)$ will be the optimal legal position of the group and $e_c(c)$ the total weight of the group. Moreover, $q_c(c)$ will be the dividend of Equation 2.5.

On the other hand, if the cell overlaps with a group (pre-placed cells), the cell is inserted in the group with Function 4, $AddCell(c, i)$, which adds the cell's parameters to the group. Moving on, Function 6, $Collapse()$, finds the final position for the cluster, line: 2. The final position is determined by Equation 2.6. However, the new position of the cluster may protrude from the core area, so the alignments are necessary, lines: 4-7. Next, the algorithm must check if the current cell group overlaps with other groups, lines: 10-14. If an overlap exists, the two groups are merged to create one group with no cell overlapping. Function $AddCluster()$ is similar to Function $AddCell()$, but corresponds to group of cells.

Finally, Abacus places the cells of each group to the legal positions depending on the position of the group, that has been previously found, lines: 15-22. Abacus achieves an impressive improvement on TWL in contrast to Tetris. This is due to the fact that Abacus ensures the maintenance of the initial cell order. On the other hand, Abacus is slower than Tetris since many cells must be re-legalized in each placement row, so as to find the legal positions with the best total displacement.

Algorithm 3: PlaceRow

```
for i = 1, ..., C_r do
    c = Last cluster;
    /* First cell or cell i does not overlap with last cluster: */
    if i == 1 or x_c(c) + w_c(c) ≤ x'(i) then
        Create new cluster c;
        Init e_c(c), w_c(c), q_c(c) to zero;
        x_c(c) = x'(i);
        n_first(c) = i;
        AddCell(c, i);
    else
        AddCell(c, i);
        Collapse(c);
end
/* Transform cluster positions x_c(c) to cell positions x(i) */
i = 1;
for all clusters c do
    x = x_c(c);
    for i ≤ n_last(c) do
        x(i) = x;
        x = x + w(i);
    end
end
```

Algorithm 4: AddCell(c, i):

```
n_last(c) = i;
e_c(c) = e_c(c) + e(i);
q_c(c) = q_c(c) + e(i) * (x'(i) − w_c(c));
w_c(c) = w_c(c) + w(i);
```

Algorithm 5: AddCluster(c, c'):

```
n_last(c) = n_last(c');
e_c(c) = e_c(c) + e_c(c');
q_c(c) = q_c(c) + q_c(c) − e_c(c) * w_c(c);
w_c(c) = w_c(c) + w_c(c');
```

2.2.3 Our Motivation

Standard cell placement is one of the most important steps in the physical design flow. Placement, just as it's sub steps, like legalization, are very complicated and not trivial prob-

Algorithm 6: Collapse(c)

```
/* Place cluster c: */
x_c(c) = q_c(c)/e_c(c);
/* Limit position between x_min and x_max - w_c(c) */
if x_c(c) < x_min then
    x_c(c) = x_min;
if x_c(c) > x_max − w_c(c) then
    x_c(c) = x_max − w_c(c);
/* Overlap between c and its predecessor c'?: */
c' = Predecessor of c;
if c' exists and x_c(c') + w_c(c') > x_c(c) then
    /* Merge cluster c to c': */
    AddCluster(c', c);
    Remove cluster c;
    Collapse(c');
```

lems. Standard cells legalization significantly influences the final placement, motivating many people to solve this problem efficiently. So, we have also been motivated to implement a novel legalizer, that legalizes standard cells not only fast but also with the less influence on the optimal global placement solution.

In this chapter we analyze the two categories of legal placement, the global and the local. The former legalizes many cells at a time, by having a global overview of all cells' positions. However, these legalizers are very slow. On the other hand, local approaches are quite fast, but make greedy decisions.

Abacus is a local and greedy approach achieving minimal standard cell displacement, which is widely used in industry. However, in designs with blockages, Abacus legal placements are sub-optimal, as it violates the initial standard cells order which is the key to minimizing total wirelength. Moreover, the original Abacus, legalizes only standard cells with the same height. Last but not least, Abacus and generally, local legalization approaches do not produce efficient solutions in overlapping dense regions.

We developed our legalizer, Abacus2, based on the well-known greedy legalizer Abacus [10], as it achieves minimal movement to the standard cells. Abacus2, adopts many ideas from the fundamental Abacus legalizer, like dynamic programming and row search bounding. Additionally, Abacus2, is capable of handling designs with blockages efficiently, by keeping the initial standard cell order. Abacus2 can also handle standard cells with different heights, with a strategy which is based on both Tetris and Abacus approaches. Moreover, Abacus2 has a super set of features which focus on better QoR (Quality of Results). In the following chapter we present Abacus2 and its features analytically and we compare them to the corresponding original Abacus features.

CHAPTER 3

Our Work

Chapter 2 presented the state of the art techniques in placement legalization. Abacus [10] is one of the dominant legalization algorithms, which achieves minimum standard cell displacement. This chapter describes Abacus2, an evolution of the original Abacus algorithm.

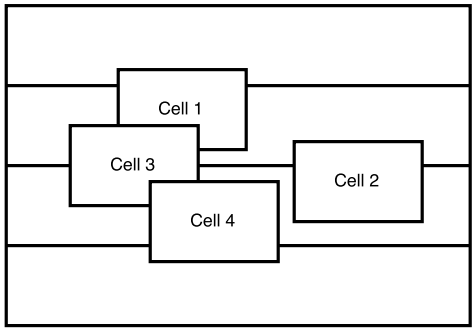
Abacus2 is based on the original Abacus framework, but it contains several new features. Abacus2 supports (i) two new displacement cost functions, multi-cell mean and multi-cell total, (ii) an additional cell order, centre-outwards, (iii) and two approaches for handling blockages, SRA, the proposed approach in the original Abacus paper, and SRR, a more advanced blockage handling approach, which allows cells to move between sub-rows. The new displacement cost functions also establish different row search bounds. Moreover, Abacus2 also includes a row overflow check, to ensure that a row has enough space for the current cell. Lastly, Abacus2 is capable of legalizing multi-height standard cells, `i.e` cells with different heights. Table 3.1 illustrates a feature set comparison between the two algorithms. These features are analyzed analytically in the following sections.

3.1 Cell Selection Order

Abacus2, like Abacus, is a sequential legalizer. Cells are placed one at a time in legal positions in a specified order. Cell ordering in different approaches, depend on different parameters, like cell global position, cell area or even the influence of each cell on the timing of the circuit. Abacus2 focuses on minimizing its influence on global placement, by minimizing the displacement of the cells from its global to legal positions. In this way, cells are legalized based on their global x-coordinate. In Abacus2, three orders are supported, the (i) increasing, (ii) decreasing, (iii) and center-outwards. Figures 3.1a and 3.1b, show a GP example and the cell legalization order, depending on the selection order for the given GP, respectively. It is worth mentioning, that cell ordering significantly influences the final legal placement, so the key is to use many orders and choose the best one.

Features			Abacus	Abacus2
Cell Ordering Support	Increasing		Yes	Yes
	Decreasing		Yes	Yes
	Centre-outwards		No	Yes
Displacement Cost Functions Support	Single-cell		Yes	Yes
	Multi-cell Mean		No	Yes
	Multi-cell Total		No	Yes
Row Search Bounding Methods	Exhaustive		Yes	Yes
	Bounded	Single-cell	Yes	Yes
		Multi-cell Mean	No	Yes
		Multi-cell Total	No	Yes
Blockages/ Hard-Macros Support	Sub-Row Assign Approach		No	Yes
	Sub-Row Re-Assign Approach		No	Yes
Row Overflow Checks			No	Yes
Multi-Row Height Cell Support			No	Yes

Table 3.1: Abacus, Abacus2 Feature Set Comparison



(a) Global Placement

Order Type	Legalization Order
Increasing	{Cell 3, Cell 1, Cell 4, Cell 5, Cell 2}
Decreasing	{Cell 2, Cell 5, Cell 4, Cell 1, Cell 3}
Center-Outwards	{Cell 4, Cell 1, Cell 5, Cell 3, Cell 2}

(b) Cell Selection Orders

Figure 3.1: Cell Selection Order Example

3.2 Displacement Cost Functions

Abacus and Abacus2 try to minimize cells displacement from the global to legal positions. Both algorithms, choose the legal positions of the cells depending on the displacement cost of each cell. Abacus algorithm uses the Euclidean distance of the last legalized cell. Equation 3.1 corresponds to the displacement cost function of Abacus, assuming that $x_c(p),\ y_c(p)$ and $x_c(n),\ y_c(n)$ are the legal placement and global placement positions, respectively, for the current cell c. We call this displacement cost function `single-cell`.

$$d_s = \sqrt{(x_c(p) - x_c(n))^2 + (y_c(p) - y_c(n))^2} \tag{3.1}$$

However, single-cell displacement cost function has a local overview of the global placement perturbation, as it only considers the displacement of the last legalized cell. This cost function does not take the perturbation of the pre-placed cells into account. This phenomenon appears mainly in dense overlapping regions. So, we propose two additional displacement cost functions, the `multi-cell mean` and `multi-cell total`, which take all the perturbed cells into account. Equations 3.2 and 3.3 determine the cost of the above cost functions.

$$\frac{d_s + \sum_{i=0}^{N} \sqrt{(x_i(p) - x_i(n))^2 + (y_i(p) - y_i(n))^2}}{N+1} \tag{3.2}$$

$$d_s + \sum_{i=0}^{N} \sqrt{(x_i(p) - x_i(n))^2 + (y_i(p) - y_i(n))^2} \tag{3.3}$$

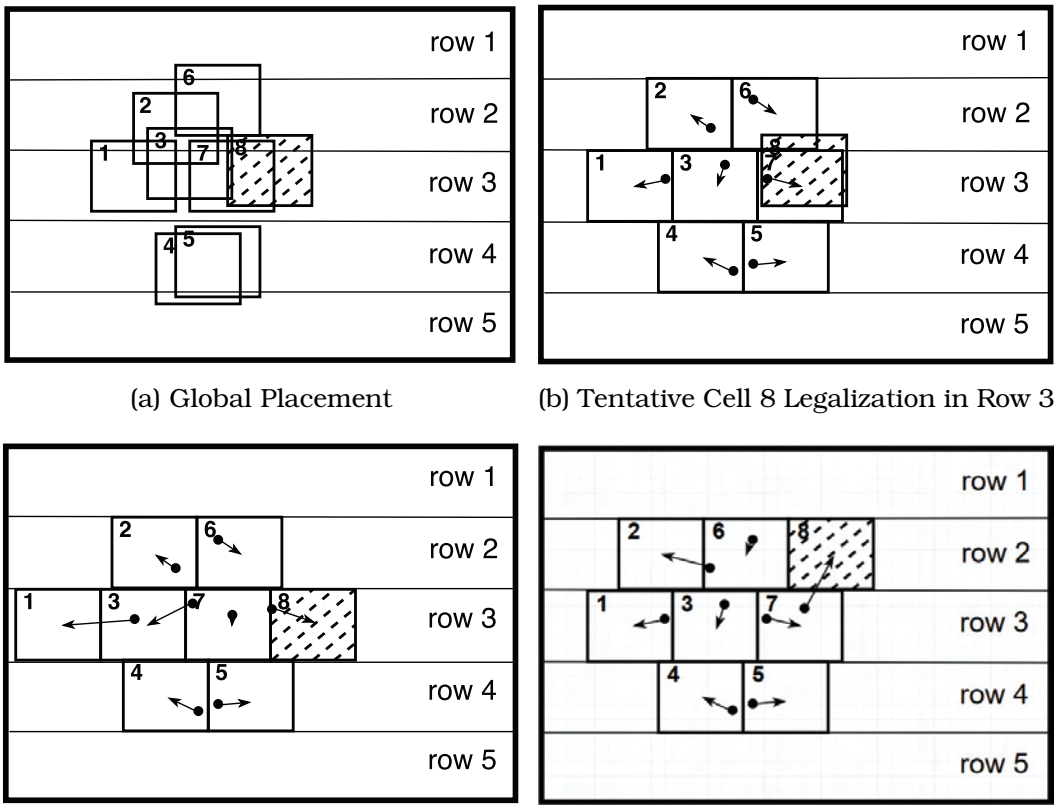
The multi-cell mean and multi-cell total cost functions are the mean and the sum displacements of the N perturbed cells and the current cell c, respectively. These cost functions have a global overview of the global placement influence.

3.3 Row Search Bounding

So as to reduce the number of the candidate rows, we first place a cell in the nearest row, to its global placement y-coordinate, and then determine its displacement. The displacement cost function is interpreted to a number of rows, depending on the placement row height, `i.e.` the number of the searched rows, around the nearest placement row, will be the result of the division in Equation 3.4. The number of candidate rows is updated when a new displacement cost is greater than the best.

$$\left\lceil \frac{best\ displacement\ cost}{core\ site\ height} \right\rceil \tag{3.4}$$

It is important to point out, that row search bounding must not influence the quality of the solution, but only the execution time. The results from an exhaustive and a bounding row search must be the same. However, the single-cell displacement cost function may lead to sub-optimal solutions, as its row bounds are extremely tiny. Figure 3.2 illustrates a simple example to understand why single-cell displacement function can lead to suboptimal solutions. The cells are legalized in increasing order and their names correspond to their selected order.



(a) Global Placement

(b) Tentative Cell 8 Legalization in Row 3

(c) Suboptimal Legal Placement - Single-cell

(d) Optimal Legal Placement - Multi-cell Mean, Total or Exhaustive

Figure 3.2: Row Search Bound Example

Figures 3.2a and 3.2b depict the global placement and the legalization of the first seven cells, respectively. Figure 3.2b also illustrates the global position of cell 8 and its nearest row, row 3. Firstly, single-cell displacement cost function will try legalizing cell 8 in the row 3 and will determine its displacement cost, which is depicted as an arrow from the global to the legal position. As we can see, this displacement cost function is very small, so, the row search bound will be extremely tiny, bounding the row search only to row 3. On the other hand, if the exhaustive or the multi-cell displacement cost functions are used, a legalization with a better total displacement cost may be found. In the same example, Figure 3.2d, the algorithm exhaustively explores all the available rows and finds a better legal placement,

in terms of total displacement cost. In the same way, multi-cell total and multi-cell mean displacement cost functions will examine the necessary rows, as they have a global overview of the legalization in the row. The results of the latter two displacement cost functions are the same even if we use the row search bounds or not. However, multi-cell total creates very loose bounds in overlapping dense regions, as the sum of all the perturbed cells are very big. So, the execution time for multi-cell total displacement cost function is comparable to the relative exhaustive row search execution time.

3.4 Legalization Artifacts

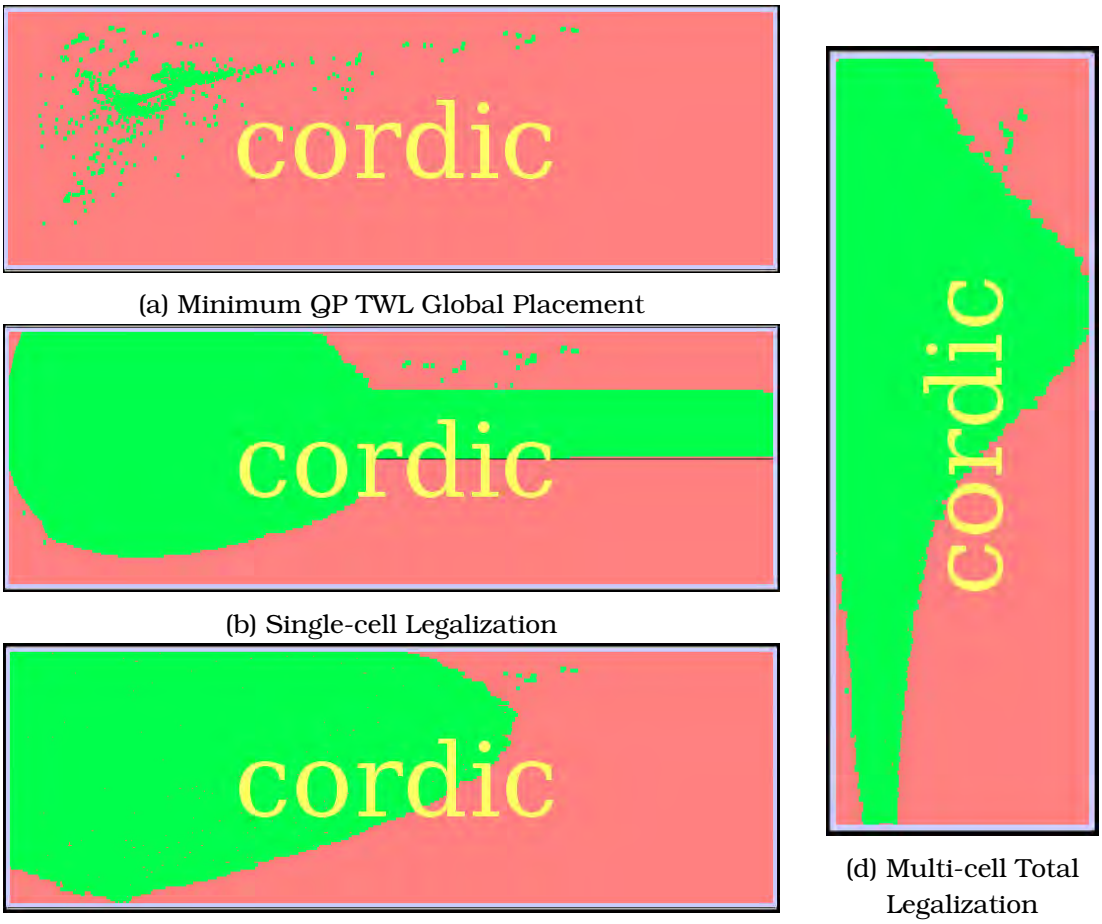
Another interesting finding of our legalization experiments was the observation of certain placement artifacts, stemming from the single-cell and multi-cell total displacement functions. We tested Abacus2 using minimum quadratic TWL placements, generating from the solution of the formulated QP placement problem. The difference in the results, deriving from a GP, is that the QP solution does not spread cells therefore, reducing overlaps.

We observe *horizontal stripe artifacts* in very dense QP placements and when the single-cell displacement function is used, *i.e.* uneven row occupancy, with certain rows forming horizontal stripes. On the other hand, when using the multi-cell total displacement function for the same dense design, we observe *vertical stripe artifacts*, *i.e.* vertical stripes formed across rows.

Figure 3.3 illustrates the `cordic_I4` benchmark, which reveals this trend. In this example, chip utilisation is 50%, the aspect ratio is 3:1, and cells are selected in decreasing x-coordinate order. The aspect ratio is selected so as to accentuate the horizontal stripe artifact. The reason why the horizontal stripe artifact occurs is the following. As the design is very dense, after the first set of GP cells is placed, any new legalized cell will push its already legalized counterparts to the right for two reasons. Firstly, since both Abacus and Abacus2 aim to maintain the original cell order, thus scanning new cells to the left, pushes the already legalized ones to the right. Secondly, since the single-cell bound does not take the displacement of the cells that moved to the right into account, the algorithm will greedily stick to this solution. Thus, other rows further than the nearest one will seldom be searched. Figure 3.3b, illustrates the horizontal stripe artifacts produced by the single-cell displacement function and bound.

In contrast to single-cell, multi-cell total displacement considers all the perturbed cells. In this way, the more overlapping cells there are, the greater increment in displacement cost is observed. So multi-cell total actually tries to move as less cells as possible from their GP positions. However, in overlapping dense regions, in order to move few cells, current cell is legalized far away from its nearest row, creating vertical artifacts like the example in Figure 3.3d. Moreover, as displacement cost is increasing, row search bound is also increasing, leading multi-cell total to great execution time.

When multi-cell mean is used as displacement function and bound, the legalization result will not produce any artifacts, as illustrated in Figure 3.3c. Multi-cell mean is horizontal



(a) Minimum QP TWL Global Placement

(b) Single-cell Legalization

(c) Multi-cell Mean Legalization

(d) Multi-cell Total Legalization

Figure 3.3: Legalization Artifacts for Single-cell and Multi-cell total displacement functions

artifact free because it takes all the perturbed cells into consideration.

3.5 Blockage Handling Strategies

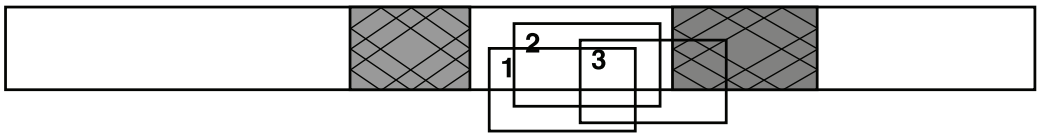
Abacus2 supports two approaches for handling blockages. The first approach, `Sub-Row Assign (SRA)`, was proposed in the original Abacus work but not experimentally evaluated. In the SRA approach, blockage locations divide rows in row segments, called sub-rows. The legalization algorithm can then operate on sub-rows, instead of placement rows, with the restriction that cells were once assigned, they may not move between sub-rows. Thus, legalization will identify the nearest sub-row, instead of the nearest chip row.

The key disadvantage of the SRA approach is that it will only maintain the relative cell order of GP within sub-rows, but will very likely violate it across sub-rows. We illustrate this with a contrived example. Figure 3.4a shows a GP with two blockages, depicted in gray, and three cells, 1, 2 and 3. If we assume increasing x-coordinate order, cells will be considered in the order 1, 2, 3. Figures 3.4b, 3.4c illustrate the legalized positions for cells 1 and 2. *cell* 1 is legalized in the middle sub-row, where its displacement cost is minimal. As *cell* 2 cannot fit there too, it is placed within the right sub-row. The last cell, *cell* 3 will have little choice, but to be legalized in the left sub-row. Figure 3.4d, illustrates the SRA legalization, with arrows representing cell displacement from their GP location. The original cell order is thus not maintained, due to the presence of blockages, and the fact that cells may not move across sub-rows.

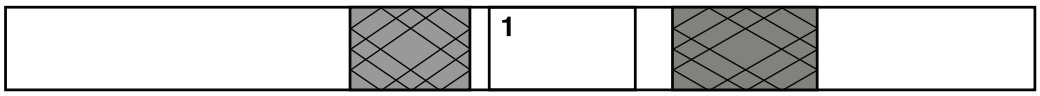
The alternative blockage handling approach, `Sub-Row Re-assign (SRR)`, tackles this issue by allowing cells to be reassigned to other sub-rows, so as to maintain the original relative cell order as much as possible. The operation of SRR is recursive in the case where a sub-row becomes full when a cell is moved there. An inter sub-row cell move thus creates a wave of recursive, or iterative, such moves, to ensure that: (i) no sub-row overflows, and (ii) the original cell order is maintained.

SRR identifies the nearest sub-row for the cell, but it maintains cell order. The selected cell order determines how to achieve this. First, SRR identifies the cell's closest sub-row. Then, it identifies the rightmost cell of the current placement row. The nearest sub-row then becomes the sub-row with the largest x-coordinate between the closest cell sub-row, and the sub-row of the rightmost legalized cell. For the other two cell orders, SRR uses the same idea in their respective orders.

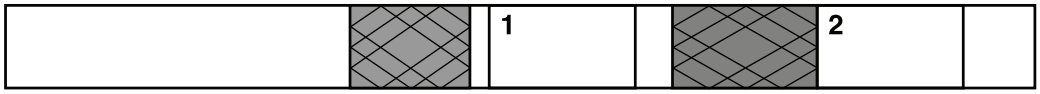
In the same example of Figure 3.4, and after *cell* 1 is legalized, the SRR approach will attempt two tentative legalizations for *cell* 2. The first is to move *cell* 1 to the left sub-row, so as to make room for *cell* 2 in the middle sub-row. The second is to place *cell* 2 in the right sub-row, as in the SRA approach. Note that in the case where a decreasing x-coordinate order is used, the first tentative move of SRR would correspond to moving the existing cells of the middle sub-row to the right instead. Out of the two tentative moves, the one with the least displacement is the second one, *i.e.* the same result as that of SRA, Figure 3.4c. When *cell* 3 is to be legalized by SRR, it will not fit in either the middle or right sub-rows. Thus, the former is assigned to its nearest sub-row, the right sub-row, reassigning *cell* 2 to the middle sub-row, which then recursively reassigns *cell* 1 to the left sub-row. The result of SRR, which



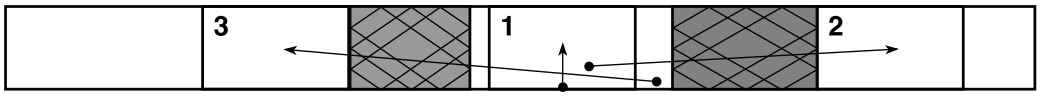
(a) Global Placement



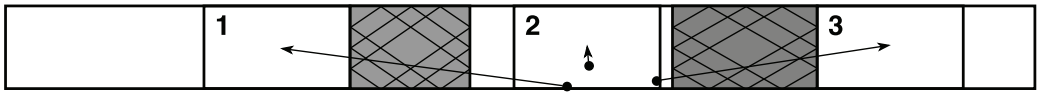
(b) Cell 1 Legalization in Middle Sub-row



(c) Cell 2 Legalization in Right Sub-row



(d) SRA - Cell 3 Legalization in Left Sub-row



(e) SRR - Cell 3 Legalization in Right Sub-row

Figure 3.4: Blockage Handling Approaches Example

maintains the original, relative cell order, is illustrated in Figure 3.4e. Its TD is less then the SRA solution.

It is interesting to note that if blockages are not present, the two legalization approaches produce the same result as sub-rows do not exist and both SRA and SRR will maintain the cell order.

3.6 Multi-Row Height Cells Handling Approach

Our legalizer, Abacus2, can also handle cells with different heights, `i.e. Multi-Row Height Cell (MRHC)`. MRHCs can have arbitrary heights, but it must be integral multiple of the row height. In contrast to the classic soft macros, the number of MRHCs in a modern circuit is much larger. In this way, the legal positions of MRHCs must be selected efficiently, so as to influence TWL as little as possible. MHRCs are mainly flip-flops, which play an important role in the design timing, so their legal positions must differ as little as possible from the optimal, in terms of TWL, global placement positions. It is inefficient to use the original Abacus approach to legalize MRHCs, as MRHCs will move cells from many rows and

increase both legalization execution time and final TWL. As a consequence, we decided to use the legalization strategy of Tetris [5] algorithm, to place MRHCs, by fixing them to their nearest GP position, in order for pre-placed cells not to be moved. Abacus2 can be subdivided into two stages, the Multi-Row Height Cell and Single-Row Height Cell legalization. The latter, legalizes cells according to the features that have been described in the previous sections. Single-Row Height Cells are the standard cells, which height is equal to the placement row height. The former, legalizes and fixes each MRHC to the legal position with the minimum cell displacement. The MRHC will not be moved again from this legal position, throughout the legalization procedure. As a result, the displacement cost function for the MRHCs is the single-cell, as only one cell moves at a time.

The MRHC handling approach, treats the already legalized MRHC as blockages, by subdividing placement rows into sub-rows. Figure 3.5 shows a simple example, so as to understand how a (sub-)row can be subdivided into sub-rows.

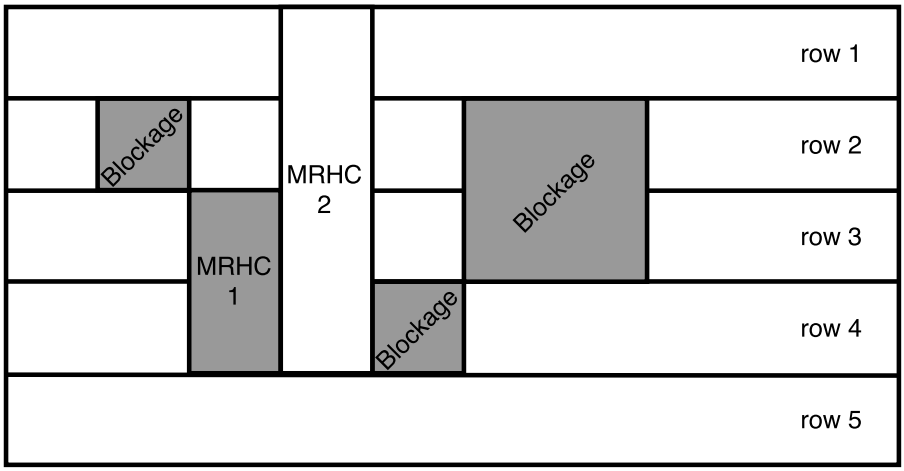


Figure 3.5: (Sub-)Row Diviation from a MRHC

Blockages and already legalized MRHCs may exist in core area, gray colored boxes. We assume that the optimal legal position for $MRHC\ 2$ is depicted in the figure. In this example, we can see how a (sub-)row can be subdivided, `i.e.` (i) in row 1, $MRHC\ 2$ subdivides the row into two sub-rows, (ii) in the second row, MRHC subdivides the middle sub-row, (iii) in row 3, the current sub-row is shrinked and (iv) in row 4, the $MRHC\ 2$ occupies the hole sub-row, so this sub-row ceases to exist.

Next, the MRHC handling approach will be presented with a contrived example. The algorithm starts from the global placement solution with blockages already placed in the core area, Figure 3.6.

The white MRHC must be legalized in the nearest legal position. The algorithm will find all the legal positions for the cell and will legalize it to the position with the minimum displacement from the global position. In order to find the corresponding sub-rows, Abacus2 scans the core area into two opposite directions, `i.e.` bottom-up and top-down. Both scan

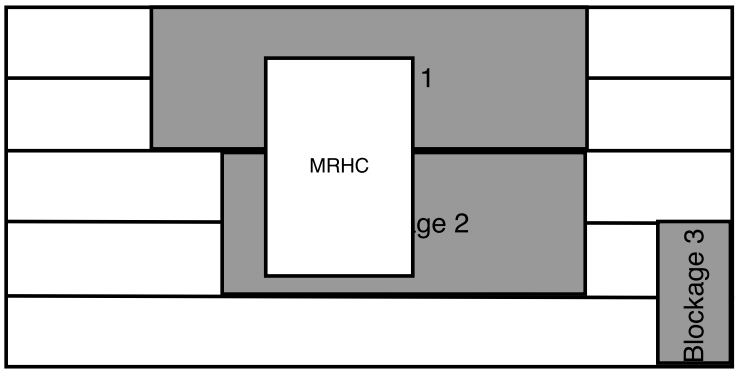


Figure 3.6: MRHC Legalization Example: Global Placement

directions are necessary, so as to find all the available sub-rows. Figures 3.7 and 3.8 depict the core area bottom-up and top-down scan of the GP example in Figure 3.6.

Firstly, by taking the lower y-coordinate of the MRHC as a reference, the algorithm, starts from the nearest row and tests all the available sub-rows that the cell fits. Then, algorithm determines the new cell x-coordinate. The algorithm searches, from the current sub-row to the height of the MRHC, if the cell can be legalized in the determined by x-coordinate position. This mean that in a legal position the MRHC will not overlap with any MRHC or blockage. In the example of Figure 3.6, the nearest row to the lower side of the cell, is row 4 and more specifically the left most sub-row. The upper left figure of Figure 3.7 illustrates that in the new x-coordinate, cell can't be legalized, since Blockage 1 blocks the second row. The next nearest sub-row is the sub-row in row 5. Again, the nearest x-coordinate to the GP position will lead to an illegal placement because of the existence of Blockage 2. Moving on, the left most sub-row of row 3 will also lead to illegal placement. Finally, the right most sub-row of row 3 leads to a legal placement (lower right figure). When a legal placement is found, the displacement cost is determined. Each MRHC will be placed in the position with the minimum displacement cost of the cell.

However, the bottom-up scan is not enough to find all the available sub-rows. In Figure 3.7, it is clear that there are many other corresponding sub-rows. As a consequence, a top-down scan is necessary to find the remaining available sub-rows. Only the combined results of bottom-up and top-down scans cover all the possible cases. In Figure 3.8, the top-down scans are presented, starting each time from the nearest sub-row of the upper y-coordinate of the cell. The algorithm scans the sub-row in the same way as in bottom-up, but in the opposite direction.

Finally in this example, we can see that there are four available positions to legalize the MRHCs. The algorithm will legalize the cell in the position with the least displacement cost, `i.e.` the position which is illustrated in the lower left figure in Figure 3.8. After the legalization of each MRHC, it is fixed in this position and never moved again. The solution

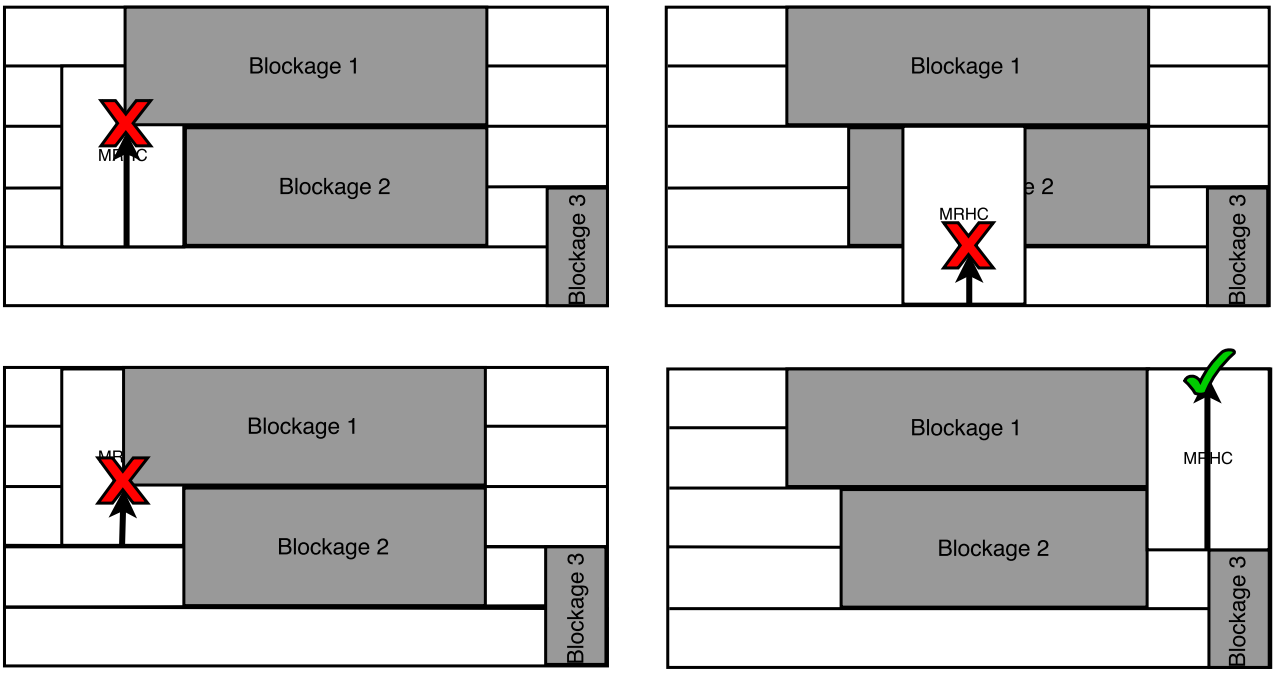


Figure 3.7: Bottom-Up Sub-row Scan

space can be reduced by using the row search bounding approach (see Section 3.3). This method can be used not only to reduce the number of the explored rows, but also to reduce the number of the explored sub-rows in each row.

The drawback of this approach, like the Tetris approach, is that the fixed cells will create many whitespaces between the MRHC. This problem can be handled efficiently, in terms of TWL, by the SRR algorithm as cells may be shifted to the gaps with the recursive re-legalizations. However, the execution time, of the legalizer, will be very slow.

3.7 Abacus2 Algorithm

In this section we present how features from the previous sections are combined, in Abacus2. Algorithm 7, describes the top level pseudocode of Abacus2 legalizer. The algorithm starts from the solution of a global placement (GP), where cells C have been placed in the optimal, in terms of TWL, positions. C contains information about the cells, like their global coordinates, their width and height, `e.t.c.` Moreover, the coordinates of blockages and placement rows are already known. The user can specify the legalization cell order O (see Section 3.1), the blockage handling approach (SRA of SRR) (see Section 3.5) and the displacement function (DF) . The output of the algorithm is the legal placement (LG) of cells C, in the core area, where blockages may exist.

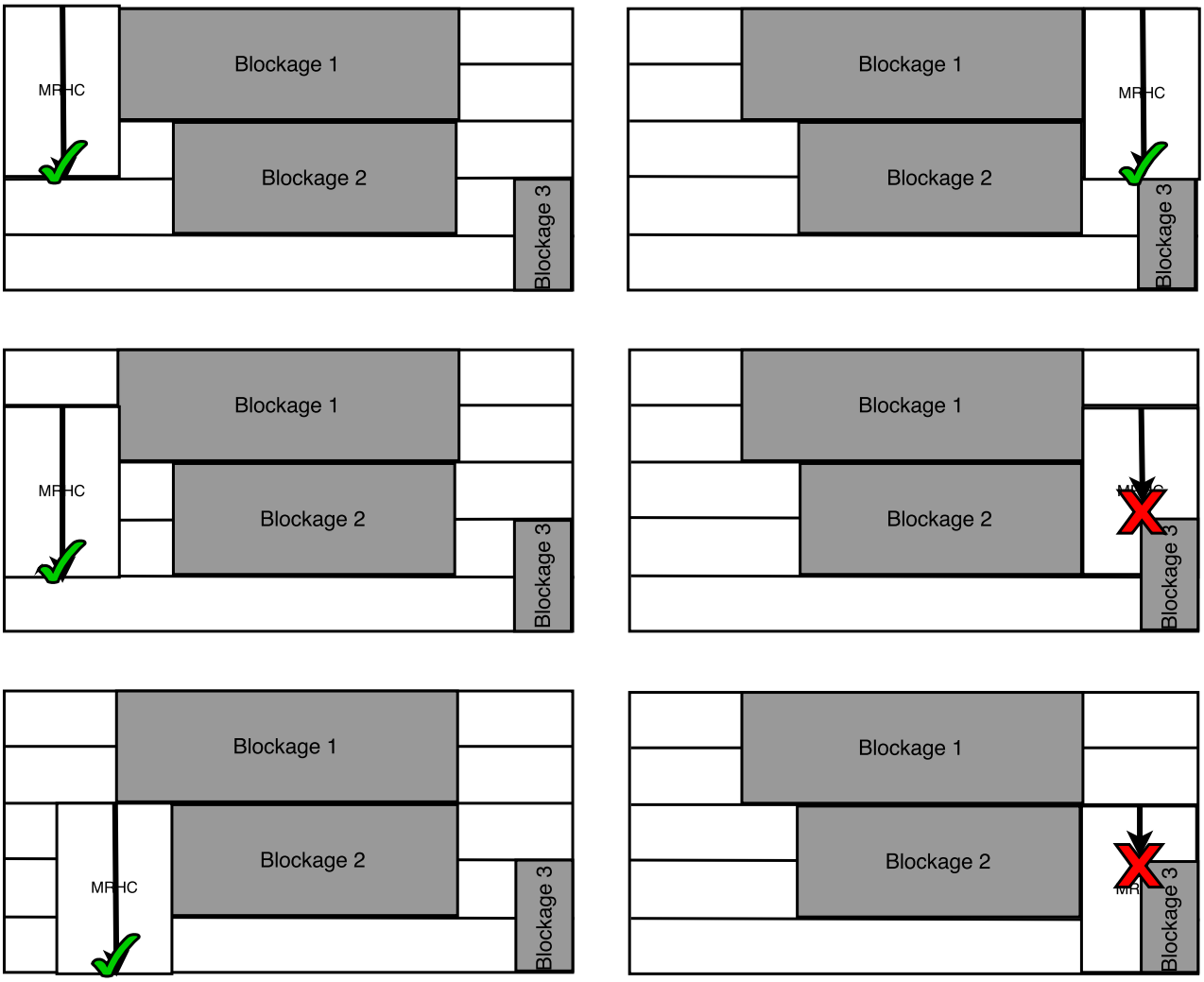


Figure 3.8: Top-Down Sub-row Scan

Firstly, Abacus2 defines the placement sub-rows S_R of each row r, lines: 7-10. The sub-rows are created by the intersection of the original placement rows R and the positions of the blockages H. If blockages are not present, rows and sub-rows are the same. Next, cells are subdivided into the Multi-Row Height Cells (MRHC) and the classic `Single-Row Height Cells (SRHC)`, lines: 13-18. Function $isMHRC$, checks the cell's height and determines if the cell is MRHC or not. In the first case, the cell is inserted in C_M, a union of all the MRHC. However, in the second case, C_S contains all the SRHC. Then, the algorithm separates the cells, and it legalizes the MRHC first and then legalizes the SRHS. It is worth mentioning, that MRHC can be sorted not only by their x-coordinate, or by their height, but also by combining them. For example, the tallest cells may be sorted and legalized first and then the shortest.

Algorithm 7: Abacus2 Algorithm

```
1  Input: GP (C), Blockages (H), Placement Rows (R), Order (O),
2         Blockage Approach (SRA), Displacement Function (DF)
3  Output: Legalized Placement
4  Multi-Row Height Cells C_M;
5  Single-Row Height Cells C_S;
6  /* Define placement sub-rows */
7  for each r ∈ R do
8      for each r ∩ H do
9          create sub-rows S_R;
10     end
11 end
12 /* Find Multi-Row Height and Single-Row Height Cells */
13 for each c ∈ C do
14     if isMRHC(c) then
15         C_M = C_M ∪ c;
16     else
17         C_S = C_S ∪ c;
18 end
19 /* Legalize Multi-Row Height Cells */
20 C_M = cell_sorting(C_M, O);
21 if C_M ≠ ∅ then
22     legalize_MRHC(C_M);
23 /* Legalize Single-Row Height Cells */
24 C_S = cell_sorting(C_S, O);
25 if C_S ≠ ∅ then
26     if SRA) then
27         SRA(C_S, DF);
28     else
29         SRR(C_S, DF, O);
```

As for cells with the same height, the user specified order O determines the legalization order.

Function 8, legalizes the MRHC. The algorithm gets a list of MRHC (C_M) and the placement sub-rows S_R and produces a legal placement for the MRHC. For each MRHC and each placement row in the bounded list of rows B, Abacus2 makes tentative legalizations to find the optimal legal position, in terms of TD. As we described in Section 3.6, MRHC are legalized by taking two points of view into account , the bottom-up and top-down. Starting from the lower row r, bottom-up approach finds the available sub-rows, line: 9. Function $get\_subrows$, returns a sorted list of sub-rows in row r, where cell c fits. This list, is sorted based on the the distance of the sub-row from cell c x-coordinate.

Algorithm 8: legalize_MRHC

```
1  Input: Sorted Multi-Row Height Cells List (C_M), Placement Sub-Rows (S_R)
2  Output: MRHC Legalization
3  for each c ∈ C_M do
4      B = placement rows;
5      best_cost = ∞;
6      flag = 0;
7      for each r ∈ B do
8          /* Bottom-Up Trial Legalization */
9          A = get_subrows(c, r, S_r);
10         backtracking_trial_legalizations(c, best_cost, r + 1, r + get_height(c), A);
11         /* Top-Down Trial Legalization */
12         A = get_subrows(r + get_height(c), S_r);
13         backtracking_trial_legalizations(c, best_cost, r + get_height(c) − 1, r, A);
14     end
15     Select best legalization;                /* final */
16     update_subrows();
17 end
```

Then, Algorithm 9, backtracking_trial_legalizations, scans the sub-rows to find if there is free space in the sub-rows above the current sub-row. The number of rows above, that the algorithm checks, depends on the height of the current cells, which is given from function get_height. The backtracking_trial_legalizations algorithm finds the new position that the cell c can be placed, for each sub-row in the sorted list A. This position is the nearest x-coordinate to the global position that the cell can be legalized in sub-row s. For the new x-coordinate, the algorithm checks the above (bottom-up) or the bellow (top-down) rows vertically, and discovers if MRHC fits in the respective sub-rows of these rows. Function $can\_legalized$ scans if the cell fits in the specified sub-row and if it does, a flag rises. On the other hand, if the cell can not be legalized in a sub-row, the flag falls so the backtracking scan for the current sub-row s fails. If backtracking scans, find a number of sequential vertical sub-rows that the cell fits, the displacement cost D is determined. In the case that cost D is less than the best, the best legal placement is updated.

Moving on, when all the available sub-rows in A are checked, the algorithm uses the top-down approach. This method is similar to the bottom-up scan approach, but scans the rows from the upper to the lower row of the current cell. Finally, when all the available rows are checked, the algorithm selects the legalization result with the less displacement cost and updates the sub-rows (see Section 3.6). The legalization of each MRHC changes the placement sub-rows as the cell is fixed to the legal position.

The result of the function $legalize\_MRHC$, it is that all the MRHC have been legalized in the core and new sub-rows have been created. From now on, MRHC are treated as blockages, so as to disable them from moving. Then, Abacus2, legalizes the Single-Row Height Cells

Algorithm 9: backtracking_trial_legalizations

```
1  Input: Current Cell (c), Best Cost (best_cost), Lower Cell Row (l_r), Lower Cell Row (u_r),
     Sub-Rows of Current Row (A)
2  Output: Best Cost (best_cost), Tentative Best Legalization
3  for each s ∈ A do
4      find_new_position(c, s);
5      for each u ∈ [l_r : u_r] do
6          if !can_legalized(c, u) then
7              flag = 0;
8              break;
9          else
10             flag = 1;
11     end
12     if flag then
13         D = determine_cost();
14         if D < best_cost then
15             Update best legalization;  /* tentative */
16             best_cost = D;
17 end
```

(SRHC), with the user specified blockage handling approach (SRA or SRR).

The SRA approach pseudocode is illustrated in Algorithm 10. Each cell, in the specified order is tentatively legalized within a bounded range of sub-rows, lines: 3-7, where the bound is initially the entire set of rows. For each sub-row, where the cell can fit, a trial legalization is performed, and the displacement cost D is determined, lines: 8-10. If the new cost D is greater than the current best cost, the current best cost and the cells best legal location are updated, lines: 11-13. At this point, the row bound is relaxed, based on the displacement cost of the current best solution, line: 14. After all sub-rows within the bounded range are explored, the cell will be legalized at the sub-row location of minimum displacement cost, line: 17.

The SRR approach pseudocode, which supports cell sub-row reassignment, is illustrated in Algorithm 11. Similarly to SRA, each cell, in the specified cell order, is tentatively legalized within a bounded range of sub-rows, lines: 3-6. Here, the nearest valid, sub-row is identified, line: 7. The nearest valid, sub-row is the nearest sub-row, where (i) the current cell may fit, and (ii) the original cell order is maintained, within the chip row. If cells need to be moved for the current cell to fit within the identified sub-row, SRR will recursively move the necessary number of already legalized cells, from sub-row to sub-row, by calling function $recursive\_moves$, line: 9. The pseudocode for $recursive\_moves$ is illustrated in Algorithm 12.

In the first recursive iteration, we check if the current cell C_i fits in the nearest sub-row, considering C_i as the only cell in the moving group M, line: 3. If the cell fits in the sub-row, it is legalized there, line: 12. In the case where the nearest sub-row is already occupied,

Algorithm 10: SRA Approach

```
1  Input: Sorted Single-Row Height Cells List (C_S), Displacement Function (DF)
2  Output: Legalized Placement
3  for each cell C_i in C_S do
4      B = # placement rows;
5      best_cost = ∞;
6      for each row R in B do
7          for each sub-row S_R do
8              if cell_fits_in_subrow(C_i, S_R) then
9                  Legalize C_i in S_R;
10                 D = determine_displacement_cost(DF);
11                 if D < best_cost then
12                     Update best legalization;  /* tentative */
13                     best_cost = D;
14                 B = min(cost_to_row_range(best_cost), B);
15         end
16     end
17     Select best legalization;                 /* final */
18 end
```

already legalized cells will be re-legalized in previous sub-rows. Function $previous$ returns the previous sub-row for a specified sub-row, depending on cell order O, *i.e.* in the increasing order, the previous sub-row of the i^{th} is the $i-1^{th}$, in the decreasing order, the previous sub-row is the $i+1^{th}$ and in the centre-outwards order, the previous sub-row depends on the direction of the selected cell compared to the central cell. Similarly, function $next$ returns the next sub-row, to a specified sub-row. On the other hand, function $next$ returns the next sub-row. In this way, if there is a previous sub-row, a new moving group M' is created, line: 5. M' consists of the cells that must be removed from the current sub-row so as to legalize M group there. However, in very thin sub-rows, the new group M' may contain cells from the previous moving group M, so after each legalization each moving group must be updated, line: 11.

After the creation of the new moving group M', the algorithm recursively tries to move cells to previous sub-rows, until either all cells are legalized in the sub-rows, or there is a moving group but there are no more previous sub-rows, line: 7. In the second terminating condition, the whole legalization procedure fails to re-legalize cells. If the recursive re-legalizations are achieved, the displacement cost D is determined and the best legalization is updated, if the new cost is greater than the best, lines: 10-13.

Next, SRR will try to legalize C_i in the opposite direction of the recursive cells move, so as to find a legal position without any cell movement, again maintaining their initial order and finding the new cost, lines: 15-20. When the above approaches are both finished, the algorithm updates the number of the available rows which may be influenced by the best

cost.

When all the available rows are checked, cell C_i will be assigned to the best sub-row by reassigning cells in other sub-rows, if needed, and then will continue legalizing the remaining cells.

Algorithm 11: SRR Approach

```
1  Input: Sorted Single-Row Height Cells List (C_S), Displacement Function (DF), Order (O)
2  Output: Legalized Placement
3  for each cell C_i in C_S do
4      B = # placement rows;
5      best_cost = ∞;
6      for each row R in B do
7          Find nearest sub-row S_n;
8          /* phase1 */
9          if recursive_moves(C_i, S_n, O) then
10             D = determine_displacement_cost(DF);
11             if D < best_cost then
12                 Update best legalization;  /* tentative */
13                 best_cost = D;
14         /* phase2 */
15         if next(S_n, O) then
16             Legalize C_i in next(S_n);
17             D = determine_displacement_cost(DF);
18             if D < best_cost then
19                 Update best legalization;  /* tentative */
20                 best_cost = D;
21         B = min(cost_to_row_range(best_cost), B);
22     end
23     Select best legalization;                /* final */
24 end
```

3.8 Extra Features

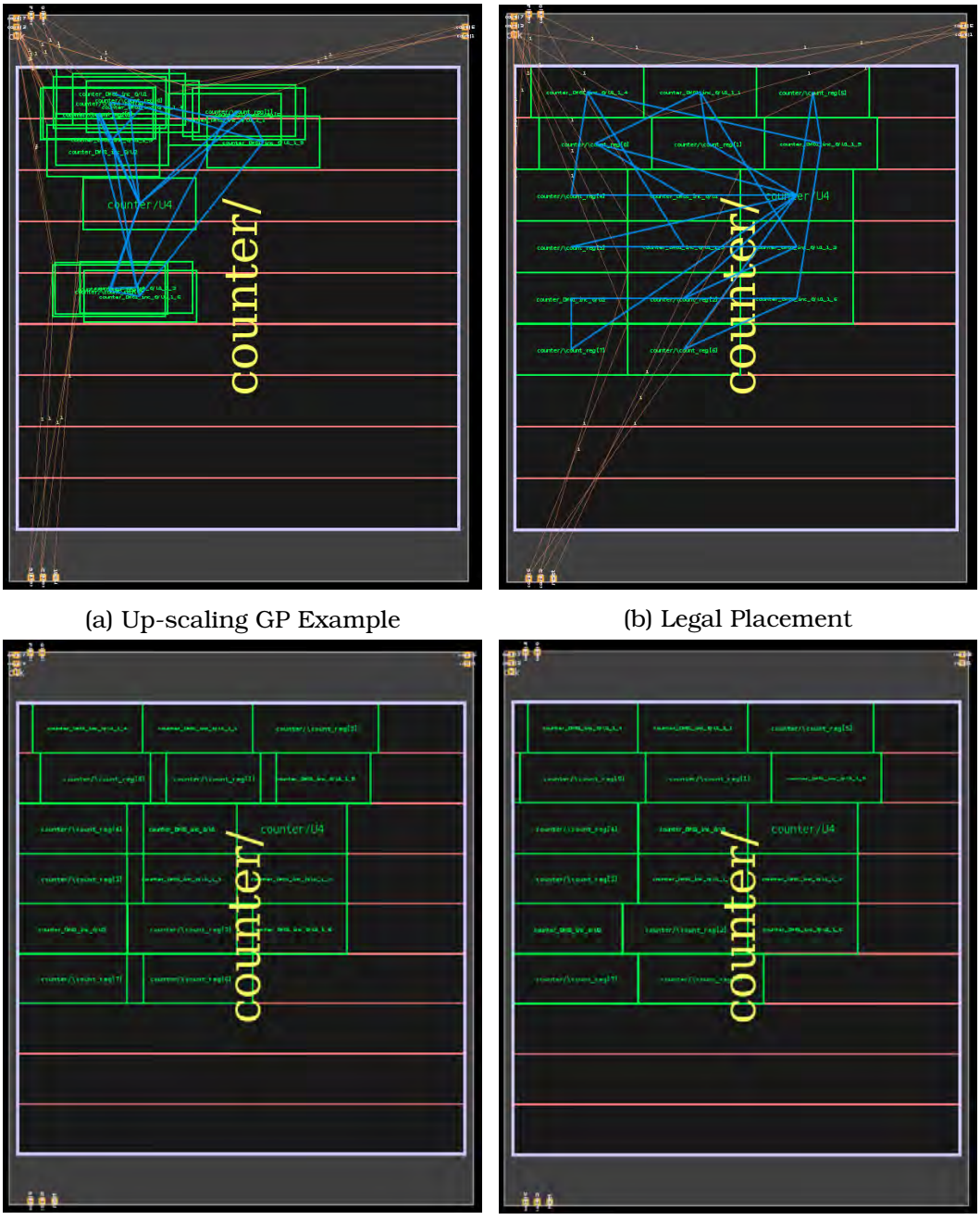
This section contain extra applications that Abacus2 supports. Abacus2 is capable of handling highly dense overlapping regions with and/or without blockages. In this way, Abacus2 can be used as a look-ahead legalizer in the placement flow. In [3], a look-ahead legalizer is used so as to legalize the cells after a number of spreading iterations. In this hill climbing approach, the final legal placement is the best legal placement found through all iterations. However, the whole placement execution time is very slow, mainly in the first iterations, due to the great number of cell overlaps.

Algorithm 12: *recursive_moves*

```
1  Input: Moving Group M, Current Sub-row S, Order (O)
2  Output: Legalized Cell in Sub-row
3  if !cell_fits_in_subrow(M, S) then
4      if previous(S, O) then
5          Create M';
6          S' = previous(S, O);
7          if !recursive_moves(M', S', O) then
8              return(False);
9      else
10         return(False);
11 Update M;
12 Legalize M in S;
```

Abacus2 can be used when cells are up-scaled, `i.e.` to increase its width. In this way, we can use the legalizer in a previous legal placement, taking the new cells' sizes into account. The same procedure can be used if the height of the cells is changed. This application of the legalizer is important, as it is time consuming to start placement procedure from scratch. Figure 3.9 presents a simple example so as to understand how Abacus2 legalizes an up-scaled circuit. Figures 3.9a and 3.9b show the example's GP and LG, respectively. Blue lines depict the cell connections and the orange lines the connections between placement I/O pins and cells. Then, if we increase, for example, the width of the same cells by 14%, cell overlaps will occur, Figure 3.9c. Finally, Figure 3.9d shows the legalization of the up-scaled placement of Figure 3.9c. As we can see in this simple example, the legalizer's work will be trivial and the final legal placement differs infinitesimally. More specifically, the initial TWL in Figure 3.9b is 121.250μm, the TWL after the legal placement of Figure 3.9c is 178.250μm and the final TWL, after the cells' up-scale legalization, is 186.030μm. As for the execution time, the legalizer is very fast as there are a few overlapping cells.

Moreover, Abacus2 can be used to legalize hierarchical circuits. Abacus2 handles hierarchical circuits with a divide and conquer technique. A floorplanning approach can subdivide the core area into smaller areas, called bounded boxes. Each bounded box contains other bounded boxes and/or cells. Abacus2 can be used in any stage of the divide and conquer technique. The legalization of hierarchical designs is faster than if the same designs is flattened because the legalizer manages a fewer number of cells and placement rows for each separate problem. However, the TWL of the legal placement is worse in hierarchical than in flat legalizations, as the legalizer loses the global overview of the cells.



(a) Up-scaling GP Example

(b) Legal Placement

(c) Up-scaling about 14% Flops Widths

(d) Up-scaled Legal Placement

Figure 3.9: Cell Up-scaling Example

CHAPTER 4

Results

This chapter presents the results of this thesis. Abacus2 is implemented in C and has been embed in an under development industrial placer. The placer reads verilog and Library Exchange Format (LEF) files and produces the necessary structures for placement. Verilog files have the description of the circuit and LEF files have the physical information about the cells and its connections. In this way, the experiments presented in this chapter, are produced from the ISPD2014, ICCAD2014 and small industrial benchmarks. Moreover, the core area utilization is set to 50% and the core aspect ratio to 1:1, `i.e.` the core area width is similar to its height. The 1:1 aspect inhibits artifacts occurring in the original Abacus, as discussed in Section 3.4. Table 4.1, presents the names, the number of the standard cells, the number of the placement rows and the GP TWL of the benchmarks that we examined. Our legalization results were obtained by using minimum quadratic TWL placements, obtained directly from the solution of the formulated QP placement problem, *i.e.* without any cell spreading to reduce overlaps. We focused on legalizing circuits with high cell overlaps, to handle the worst-case scenario of a legalizer. By ensuring that Abacus2 works efficiently, even for designs without cell spreading, we demonstrate that it can be used as a look-ahead legalizer, *i.e.* it can be use during the cell spreading steps of Global Placement [3].

In order to test Abacus2 with blockages, we used a standard, scalable blockages pattern, which tests many legalization corner cases. Our scalable blockages pattern is illustrated in Figure 4.1. We scaled this pattern so as to have blockages which occupy about 20% of the entire chip area.

4.1 Cell Selection Order Comparison

We tested three different cell selection orders, *i.e.* increasing, decreasing and centre-outwards. Table 4.2 illustrates the results of our cell ordering experiments, for the three displacement cost functions, in terms of total displacement, and the best ordering, per benchmark. The minimum displacement cost for each experiment is highlighted in bold.

Benchmark	**Num. of Cells**	**Num. of Rows**	**GP TWL (μm)**
ind1	2346	71	1.04E+5
ind2	18796	160	4.54E+5
bridge32_1	30675	135	5.29E+5
fft	32281	144	5.51E+6
cordic_I4	41601	152	2.02E+5
des_perf_1	112644	299	8.33E+5
edit_dist_1	130661	273	5.12E+6
matrix_mult	155325	294	2.13E+7
b19	219268	521	1.51E+6

Table 4.1: Benchmarks Characteristics

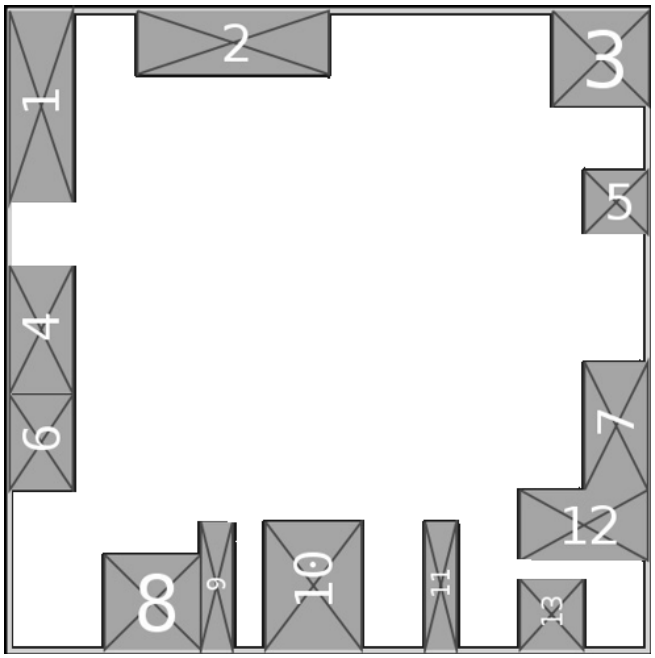


Figure 4.1: Blockages Pattern

The average results are presented respectively to the increasing cell order and can even differ up to 40%.

Moreover, the table shows that, the decreasing order produces the best, on average, results for the three displacement cost functions. This is a random observation, which is produced because the GP places the cells on the left side of the core for all testcases. The reason why the cells are placed on the left side is that most of the placement I/O pins are places on that side. As a consequence, the cells which are connected to the I/O pins are forced to be placed near them. Figure 4.2, depicts the GP of edit_dist_1 benchmark. The

Benchmark	Single-cell			Multi-cell Total			Multi-cell Mean		
	Inc.	Dec.	C-O.	Inc.	Dec.	C-O.	Inc.	Dec.	C-O.
ind1	**2.01E+4**	2.16E+4	2.40E+4	8.62E+3	8.68E+3	**8.45E+3**	9.10E+3	**7.85E+3**	8.21E+3
ind2	4.30E+5	**4.24E+5**	4.40E+5	2.35E+5	**2.22E+5**	2.35E+5	2.82E+5	**2.09E+5**	2.34E+5
bridge32_1	**2.18E+6**	2.22E+6	2.27E+6	**1.23E+6**	1.26E+6	1.25E+6	1.58E+6	**1.08E+6**	1.12E+6
fft	2.22E+6	**2.03E+6**	2.21E+6	**5.92E+5**	4.70E+5	5.85E+5	5.15E+5	**3.06E+5**	4.10E+5
cordic_I4	3.31E+6	3.29E+6	**3.20E+6**	2.07E+6	2.15E+6	**2.07E+6**	2.65E+6	**1.82E+6**	1.89E+6
des_perf_1	**1.53E+7**	1.54E+7	1.56E+7	8.59E+6	**9.23E+6**	8.94E+6	1.14E+7	**7.65E+6**	8.23E+6
edit_dist_1	1.69E+7	**1.69E+7**	1.80E+7	8.73E+6	**9.57E+6**	9.73E+6	1.08E+7	**8.44E+6**	9.28E+6
matrix_mult	**2.02E+7**	2.11E+7	2.23E+7	1.02E+7	**1.02E+7**	1.08E+7	1.17E+7	**8.40E+6**	9.41E+6
b19	6.10E+7	**5.47E+7**	5.70E+7	**4.33E+7**	3.24E+7	3.86E+7	5.64E+7	**3.19E+7**	3.43E+7
Average	1.00	**0.96**	1.00	1.00	**0.87**	0.96	1.00	**0.63**	0.68

Table 4.2: Cell Selection Orders Displacement Costs

cells are in green and the connections between cells and I/O pins are depicted with orange lines. The mass of the cells are on the top left side of the core, where there are many I/O placement pins. On the other hand, in the case where the mass of the cells are placed on the right core side, increasing order is observed to produce the best, on average, results.

Figure 4.2: Placement I/O Pins Position

4.2 Displacement Functions Comparison

In this section we compare the results of the three displacement functions, `i.e.` single-cell, multi-cell total and multi-cell mean. Tables 4.3 and 4.4 present comparative, relative Total Wirelength (TWL) and Total Displacement (TD) results, with and without the scalable blockages pattern respectively. The best ordering result of Table 4.2, for the multi-cell mean and total functions is compared against the original Abacus single-cell.

As for the results in the Table 4.3, no blockages are present, thus the SRA, SRR approaches produce the same result, as sub-rows correspond directly to chip area rows. It can be observed that multi-cell total presents worst TWL and TD, on average. This occurs, because the latter considers the total displacement of all cells, per legalization. Thus, it cannot distinguish between a cell which moved far, and another which moved near their original GP, as it considers the sum of their distances. It thus tends to move certain cells further away from their GP location. Multi-cell mean produces, on average, comparable results to the original single-cell, for designs without blockages, with multi-cell mean producing better results on certain benchmarks.

Benchmark	Multi-cell Total		Multi-cell Mean	
	TWL	TD	TWL	TD
ind1	1.27	2.49	0.99	1.04
ind2	1.32	1.97	1.16	1.03
bridge32_1	1.73	1.96	1.25	1.1
fft	1.64	6.43	0.86	1.49
cordic_I4	1.05	1.71	0.97	1.10
des_perf_1	1.71	1.93	1.33	1.09
edit_dist_1	1.48	1.95	1.16	1
matrix_mult	1.61	2.33	1	1.17
b19	0.98	1.66	0.7	0.99
Average	1.42	2.49	1.05	1.11

Table 4.3: Displacement Cost Functions without Blockages Comparison to Single-cell

Table 4.4 presents results, for the same designs, but with the specified blockage pattern included. The table presents relative TWL, TD comparisons, between the displacement functions, and the SRA and SRR approaches, compared against single-cell SRA. Multi-cell total exhibits the worst results overall, both with SRA and SRR. On the other hand, multi-cell mean reduces both TWL and TD, by about 2%, when using the SRA approach. As for the SRR approach, multi-cell mean reduces TWL and TD, by 9% and 2% respectively. The reason why multi-cell mean exhibits better results when blockages are present is due to high density regions. These are created by the blockage pattern. Such regions are likely to produce place-

ment artifacts, horizontal for the single-cell, and vertical for the multi-cell total displacement functions. Artifacts will thus lead to increased TWL. However, multi-cell mean, being artifact free, correlates TWL better to TD.

Benchmark	Single-cell		Multi-cell Total				Multi-cell Mean			
	SRR		SRA		SRR		SRA		SRR	
	TWL	TD	TWL	TD	TWL	TD	TWL	TD	TWL	TD
ind1	0.98	0.98	1.26	2.54	1.17	2.14	0.89	1.03	0.89	1.03
ind2	0.98	0.97	1.64	1.95	1.30	1.52	1.09	0.98	1.05	0.98
bridge32_1	0.97	0.97	1.77	1.91	1.32	1.52	1.15	0.98	1.04	0.97
fft	0.98	0.94	1.53	2.80	1.29	2.46	0.78	0.99	0.78	1.00
cordic_I4	0.98	0.99	1.41	2.09	1.15	1.71	0.93	1.05	0.93	1.05
des_perf_1	1.04	0.97	1.99	1.92	1.40	1.55	1.26	1.01	0.84	1.00
edit_dist_1	0.96	0.98	1.95	1.94	1.44	1.50	1.03	0.90	1.03	0.90
matrix_mult	0.96	0.97	1.69	2.39	1.32	1.80	0.82	0.98	0.82	0.97
b19	0.84	1.09	1.47	1.54	1.26	1.41	0.87	0.90	0.85	0.90
Average	0.97	0.99	1.63	2.12	1.29	1.73	0.98	0.98	0.91	0.98

Table 4.4: Displacement Cost Functions with Blockages Comparison to Single-cell SRA

Figures 4.3, 4.4 and 4.5 show the GP and legalization of cordic_I4 benchmark, with the three displacement cost functions, for SRA and SRR approaches. The artifacts described in Section 3.4 are presented in this legalization example. Multi-cell total and single-cell DF create vertical and horizontal artifacts, respectively. On the other hand, multi-cell legalizes the cells uniformly.

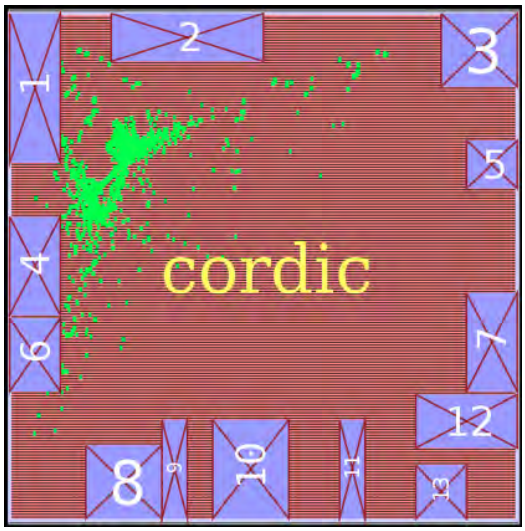
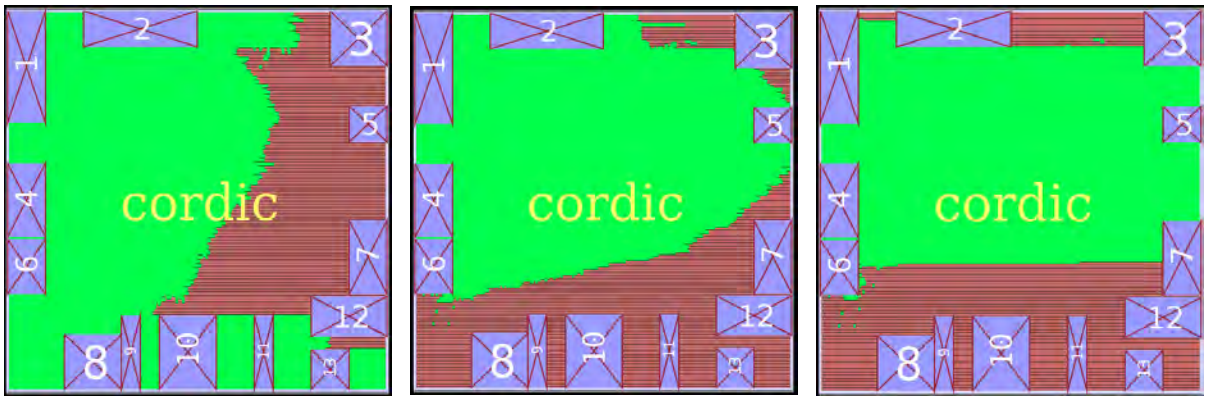
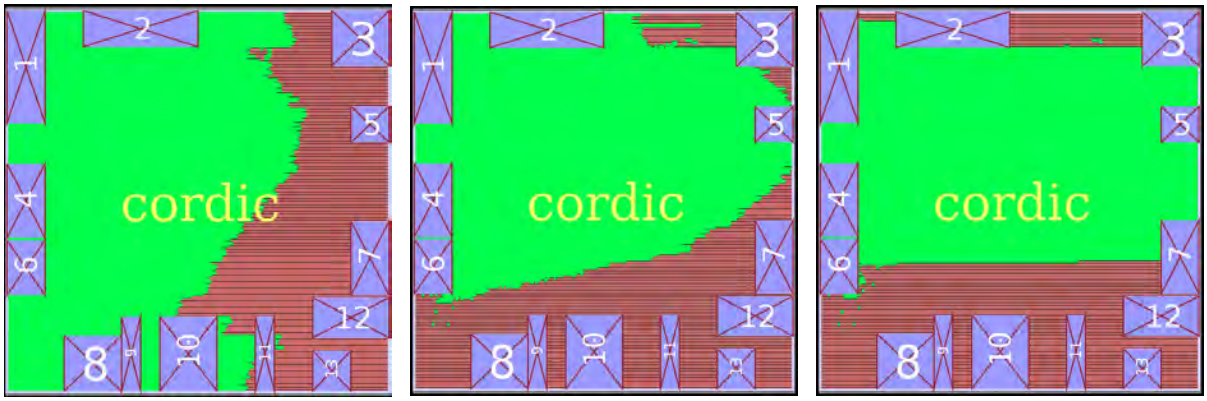


Figure 4.3: *cordic_I4* Benchmark GP Example



(a) Multi-cell Total DF LP (b) Multi-cell Mean DF LP (c) Single-cell DF LP

Figure 4.4: *cordic_I4* Benchmark SRA Displacement Function Example



(a) Multi-cell Total DF LP (b) Multi-cell Mean DF LP (c) Single-cell DF LP

Figure 4.5: *cordic_I4* Benchmark SRR Displacement Function Example

4.3 SRA vs SRR Comparison

This section presents the compared results between SRA and SRR approaches. As we can see in the previous Figures 4.4a and 4.5a, the SRA may place cells far away, so as to minimize the total cell displacement. On the other hand, SRR makes the necessary recursive moves, proceeding better legal placement, in terms of TWL and TD. The same trend can be observed when SRA and multi-cell mean or single-cell are used. However, the specified blockage patter does not reinforce this feature.

The best achieved SRR and SRA results are compared to each other, as depicted in Figures 4.6, 4.7 and 4.8. The relative TWL, TD and Execution Time of the best SRR solution are compared to the best SRA solution. It can be observed that SRR results are significantly better and achieve an average 8% reduction in both TWL and TD. This reduction stems from

the fact that SRR maintains the original, relative cell order, and has the ability to move cells across sub-rows. However, SRR's execution time is 23% slower, on average, compared to that of SRA. In terms of absolute execution time requirements, the multi-cell mean legalization of `b19` requires 183 minutes, with SRR, and 175 minutes, with the SRA approach.

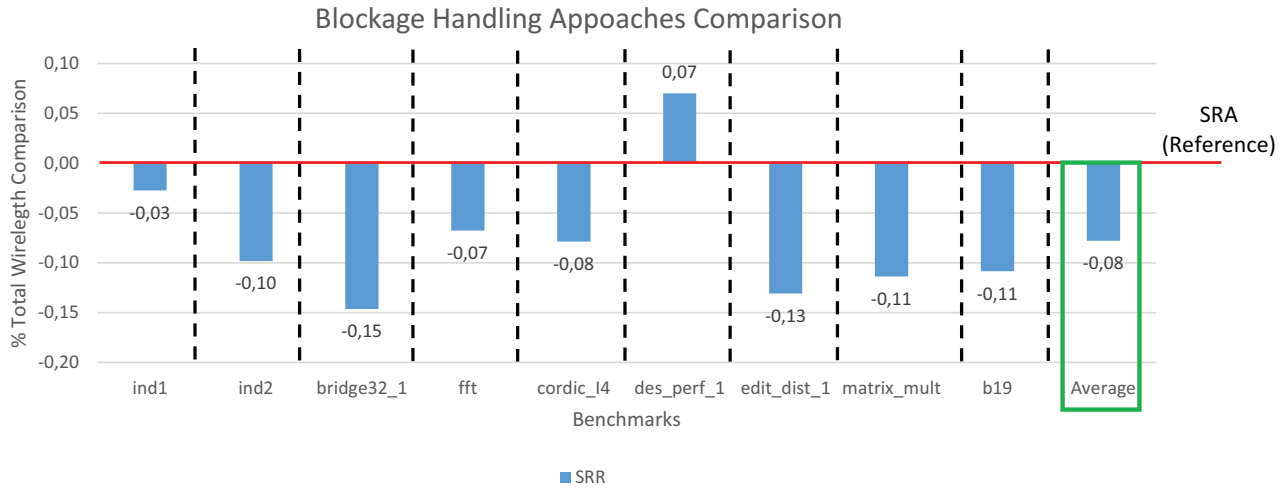


Figure 4.6: SRA vs SRR TWL Comparison

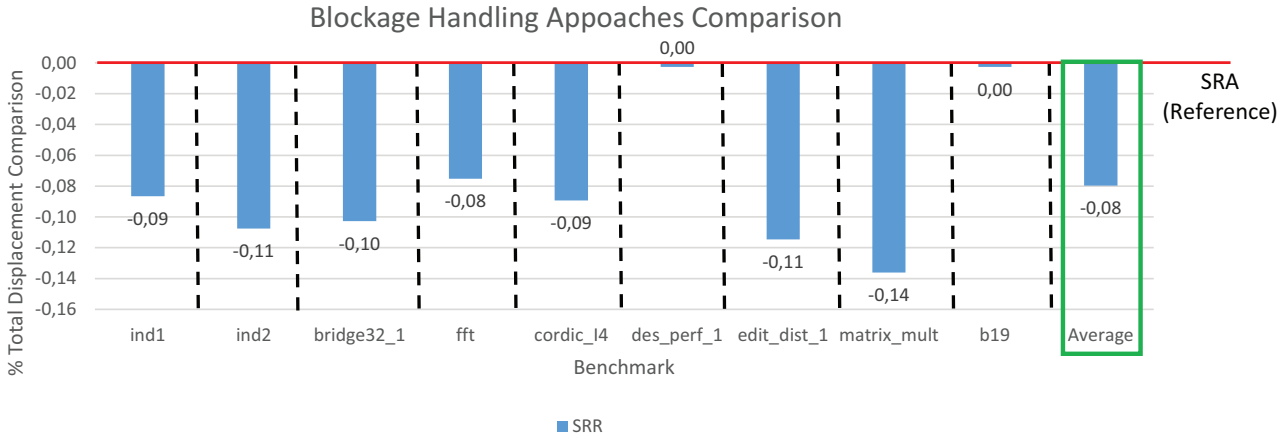


Figure 4.7: SRA vs SRR TD Comparison

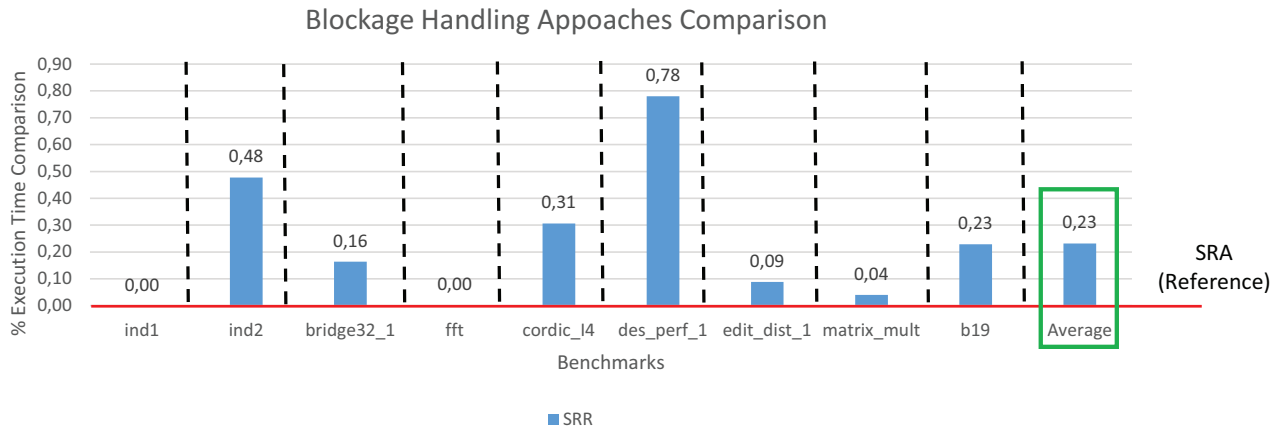


Figure 4.8: SRA vs SRR Execution Time Comparison

4.4 Multi-Row Height Cells

In this section we present results of the MRHC's impact on TWL, TD and execution time. Table 4.5, provides information about the testcases and the benchmarks of the experiments, `i.e.` it presents the benchmark names, the total number of standard cells, the percentage of the MRHC in each design and the initial TWL. It is worth mentioning that the number of the MRHC in a design is equal to the number of its flops and the initial GP does not differ when we increase its height.

Benchmark	Num. of Cells	MRHC %	GP TWL (μm)
ind1	2.346	15.39	1.04E+05
ind2	18.796	5.16	4.54E+05
bridge32_1	30.675	10.95	5.29E+05
fft	32.281	6.15	5.51E+06
cordic_I4	41.601	2.96	2.02E+05
des_perf_1	130.661	6.74	8.33E+05
edit_dist_1	130.661	4.33	5.12E+06
matrix_mult	155.325	1.87	2.13E+07
b19	219.268	3.01	1.51E+06

Table 4.5: MRHC Bechmarks

In order to record the influence of MRHC in legalization, we performed experiments where the cells' height is (i) the same as the placement row height, `i.e.` single-row height cell, (ii) twice ($x2$) the placement row height and (iii) triple ($x3$) the placement row height. Table 4.6, presents results for the TWL, the TD and the execution time for the benchmarks of

Table 4.5, where the three, $x1$, $x2$, $x3$, up-scale height factors are used on each flop. The results have been produced from the best cells selection order, the best dispalcement cost function and the best blockage handling approach. This table shows that not only TD, but also TWL are extremely increased when the flops heights are increased. The increase occurs for two reasons. The first is that Abacus2 fixes the MRHC to their legal position. As a consequence, it forces many cells, MRHC or not, to be legalized far away from the optimal GP position. Moreover, the GP of the testcases is the solution of a quadratic placement without minimizing the cells density. In this way, many cells overlap and are then forced to be legalize to sup-optimal positions. On the other hand, the execution time of legalization is significantly decreased.

Benchmark	Flops Up-scale Height Factor								
	x1			x2			x3		
	TWL (μm)	TD (μm)	Exec. Time (m:s)	TWL (μm)	TD (μm)	Exec. Time (m:s)	TWL (μm)	TD (μm)	Exec. Time (m:s)
ind1	1.80E+05	8.45E+03	0:01	2.70E+05	2.08E+04	0:01	3.07E+05	2.43E+04	0:01
ind2	3.17E+06	2.22E+05	1:05	4.24E+06	4.97E+05	0:33	4.69E+06	6.04E+05	0:29
bridge32_1	1.07E+07	1.23E+06	3:46	1.31E+07	2.49E+06	1:49	1.42E+07	2.99E+06	1:46
fft	8.23E+06	4.70E+05	2:07	9.30E+06	7.48E+05	1:15	9.62E+06	8.35E+05	1:10
cordic_I4	1.58E+07	2.07E+06	7:20	1.88E+07	2.82E+06	5:30	2.16E+07	3.16E+06	5:26
des_perf_1	7.54E+07	8.59E+06	56:00	9.79E+07	1.59E+07	39:16	1.02E+08	1.86E+07	41:13
edit_dist_1	6.08E+07	8.73E+06	71:44	8.66E+07	1.50E+07	48:00	8.80E+07	1.70E+07	48:02
matrix_mult	6.82E+07	1.02E+07	99:11	9.57E+07	1.36E+07	67:21	9.26E+07	1.46E+07	64:34
b19	2.68E+08	3.24E+07	284:05	3.16E+08	4.12E+07	261:20	3.27E+08	4.61E+07	259:01

Table 4.6: MRHC Up-scaling Results

Moreover, the charts in Figures 4.9, 4.10 and 4.11, present the compared percentages between the three cases (i) where only single-row height cells, i.e. $x1$ up-scale height factor, is used, (ii) where up-scale factor $x2$ is used and (iii) where up-scale factor $x3$ is used. The results of $x1$ up-scale height factor are used as a reference. Figures 4.9 and 4.10 depict the TWL and TD % comparison between the three up-scaling factors. The former, shows that TWL is increasing approximately 29% and 38%, when the flops' height is doubled and tripled respectively. The latter, presents the increment of TD. The cells displacement is increased due to the reasons that have previously been described. Figure 4.11 shows that execution time is decreased when MRHC are used. Execution time reduction occurs because the number of cells that are moved in each iteration become less and less, as MRHC are fixed.

Finally, Figures 4.12, shows the GP and the legal placement of des_perf_1 benchmark, when $x3$ upscale-factor is used. Figures c, d, e and f depict different stage of Figure b magnification. As we can see in Figures c and d, illustrate the legal placement of MRHC, as a bubble. The cells' overlapping density is great so many cells overlap and as a consequence MRHCs are packed in neighbouring positions. Figure f shows some MRHC with height three

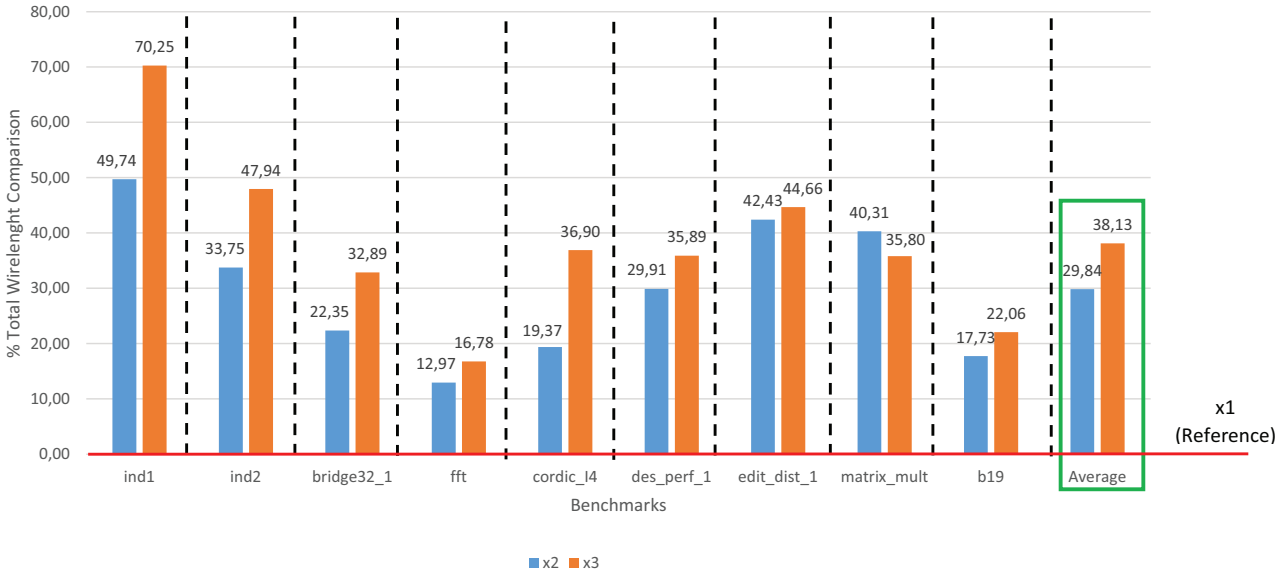


Figure 4.9: MRHC TWL Comparison

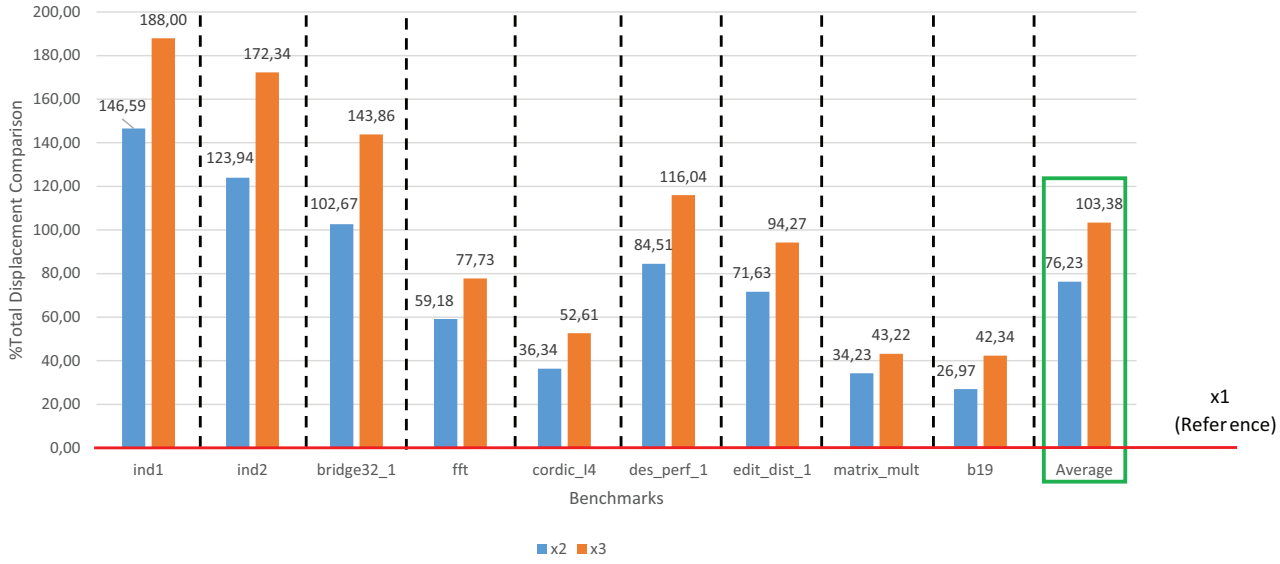


Figure 4.10: MRHC TD Comparison

times greater than the placement row height. Moreover, we can see that cells have been legalized in the gaps (white-spaces) between the MRHC.

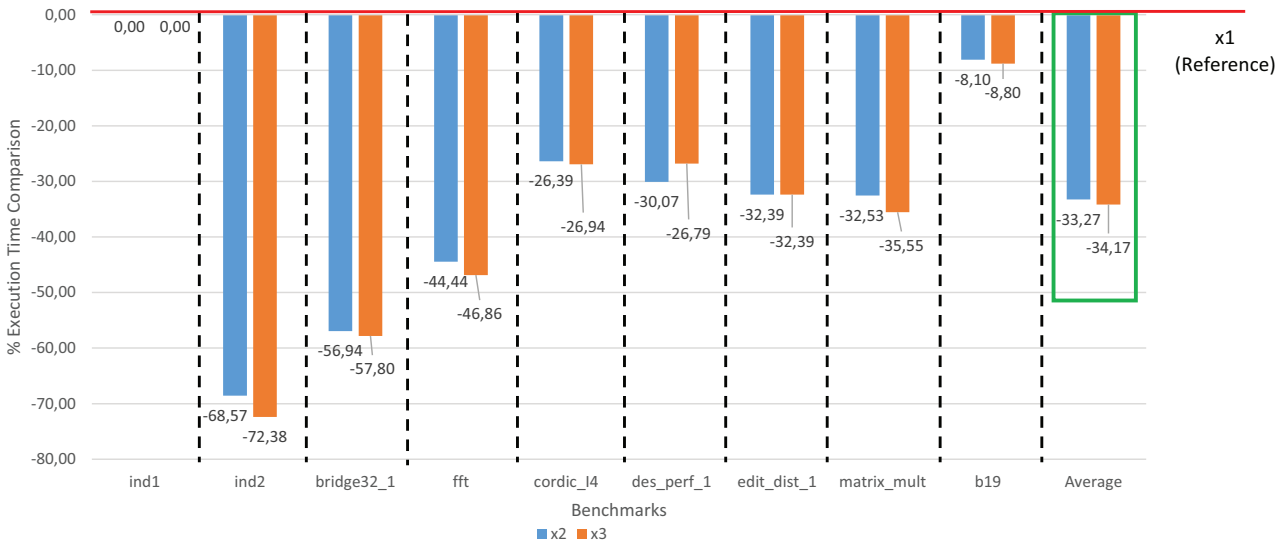


Figure 4.11: MRHC Execution Time Comparison

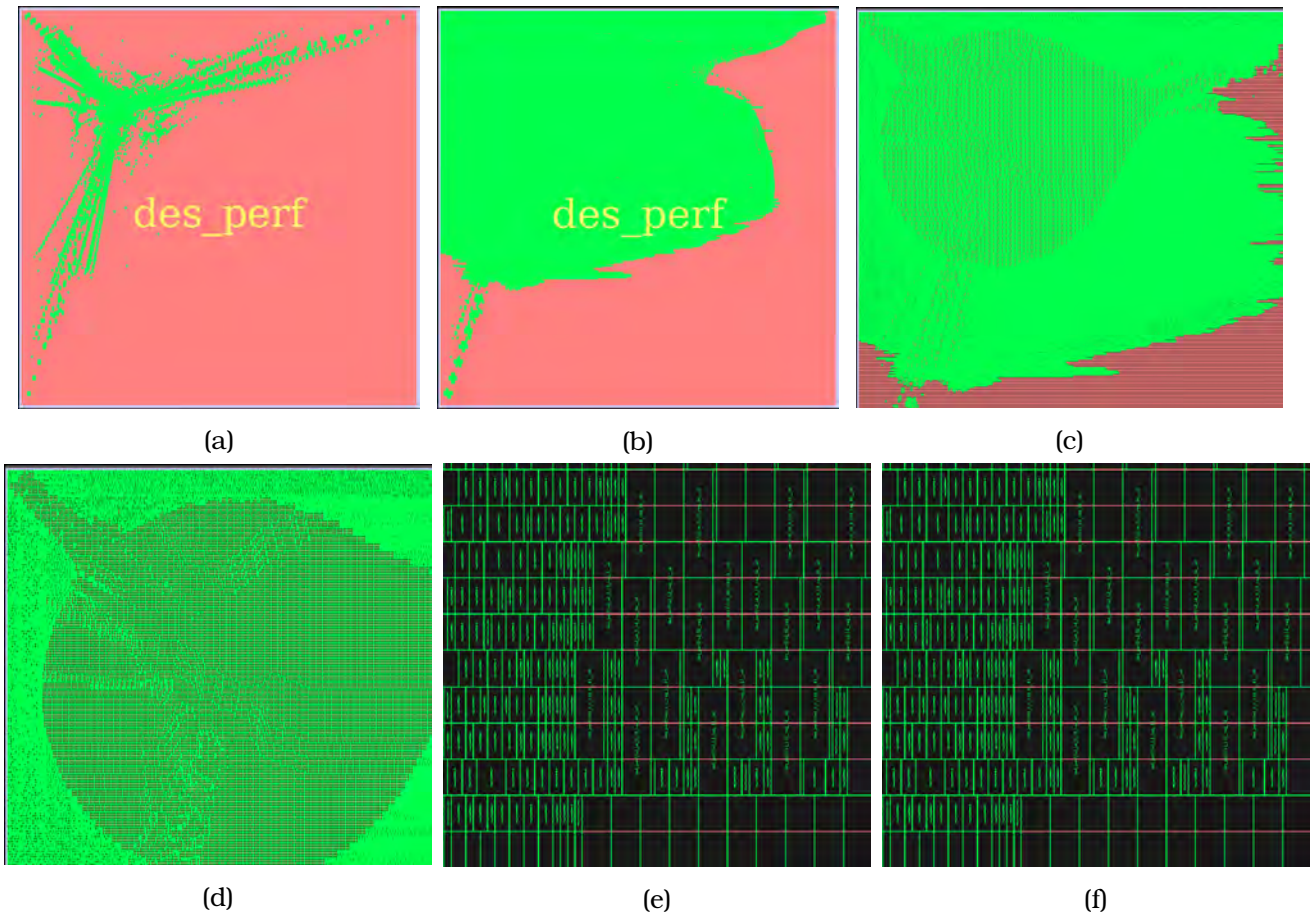


Figure 4.12: des_perf_1 Benchmark MRHC Legalization Example

CHAPTER 5

Conclusions and Future Work

In this work, we propose an evolution of the well-known Abacus placement legalization algorithm, which is called Abacus2 and it's aim is to achieve minimum cells' displacement.

Legalization is one of the three tasks of standard cell placement. The first task is global placement, which aims to generate a "rough" placement solution that violates some design constrains, such as cells' overlapping and cells' not aligning to the chip grid. The second placement task, legalization, is covered in this work. The last task, detailed placement, further improves the legalization placement solution.

Legalization must have as little impact as possible, between the first and the last tasks of placement. For this reason, Abacus2 legalizer aims to minimise the total displacement of the cells. Abacus2 has a supperior set of features than the fundamental Abacus legalizer, which lead to better legal results. Abacus2 supports three displacement cost functions and three cell sorting orders. Multi-cell mean displacement cost function is artifact free, and produces an average of 9% in reduction and a 5% increase on average, in TWL, for designs with and without blockages respectively. Moreover, Abacus2 handles placement blockages, based on Sub-Row Assign (SRA) and Sub-Row Re-assign (SRR) approaches. In SRA, a cell may only be moved within its assigned sub-row, in contrast to SRR, which allows cells to be re-assigned to other sub-rows, recursively. The SRR approach maintains the initial cells order, exhibiting an average 8% reduction, in both TWL and TD, compared to the SRA approach. Abacus2, is also capable of handling standard cells with heights integral multiple of the placement row height.

Our future goals include using Abacus2 as a look-ahead legaliser in a global placement flow, so as to exploit it's good behaviour in overlapping dense designs, and to modify its greedy approach to legalise multiple cells simultaneously.

Bibliography

[1] Charles J Alpert, Dinesh P Mehta, and Sachin S Sapatnekar. *Handbook of algorithms for physical design automation*. CRC press, 2008.

[2] Ulrich Brenner. Vlsi legalization with minimum perturbation by iterative augmentation. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1385–1390. IEEE, 2012.

[3] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1228–1240, 2008.

[4] Konrad Doll, Frank M Johannes, and Kurt J Antreich. Iterative placement improvement by network flow methods. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(10):1189–1200, 1994.

[5] Dwight Hill. Method and system for high speed detailed placement of cells within an integrated circuit design, April 9 2002. US Patent 6,370,673.

[6] Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.

[7] Sung Kyu Lim. *Practical problems in VLSI physical design automation*. Springer Science & Business Media, 2008.

[8] Igor L Markov, Jin Hu, and Myung-Chul Kim. Progress and challenges in vlsi placement research. *Proceedings of the IEEE*, 103(11):1985–2003, 2015.

[9] Majid Sarrafzadeh, Maogang Wang, and Xiaojian Yang. *Modern placement techniques*. Springer Science & Business Media, 2013.

[10] Peter Spindler, Ulf Schlichtmann, and Frank M Johannes. Abacus: fast legalization of standard cell circuits with minimal movement. In *Proceedings of the 2008 international symposium on Physical design*, pages 47–53. ACM, 2008.