UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

JULIA CASARIN PUGET

Jezz: An Effective Legalization Algorithm For Minimum Overall Displacement

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engeneering

Advisor: Prof. Dr. Ricardo Reis
Coadvisor: Msc. Guilherme Flach

Porto Alegre
July 2015

“Failure is the condiment
That gives success its flavor”
— TRUMAN CAPOTE
“Walk tall
Or, baby, don’t walk at all"
— BRUCE SPRINGSTEEN

ACKNOWLEDGEMENTS

Agradeço aos meus pais pelo exemplo e investimento em mim, aos meus amigos e familiares que sempre me apoiaram, especialmente, o meu co-orientador e amigo Guilherme Flach, que investiu seu tempo no meu desenvolvimento e não desistiu, e à sempre amiga Jozeanne.

Agradeço também às funcionárias Carmem e Ângela e aos professores Liane Loder e Marinho Barcellos, seu apoio quando precisava me ajudou a acreditar mais no meu futuro na minha profissão.

Por fim, muito obrigada ao meu orientador, Ricardo Reis, que acreditou em no meu potencial e me proporcionou grandes oportunidades de desenvolvimento profissional e pessoal durante meu período como bolsista no laboratório de microeletrônica do Instituto de Informática.

A Study On Standard-Cell CircuitLegalization And Jezz: An EffectiveLegalization Algoritm For MinimumOverall Displacement

RESUMO

Legalização é um dos três estágios em que se subdivide o posicionamento de portas lógicas na síntese física de um circuito integrado. Esse estágio consiste na seleção de posições consideradas válidas para as portas lógicas, ou seja, posições que estejam alinhadas às bandas divisoras do circuito, e em que não haja sobreposições.

Além de organizar as portas lógicas em posições válidas, uma legalização bem feita necessita prover uma transição suave entre o posicionamento global, o primeiro estágio do posicionamento, e o posicionamento detalhado, o último estágio, de forma que a solução alcançada no posicionamento global seja modificada o mínimo possível.

Neste trabalho, é feito um estudo sobre os algoritmos de legalização presentes na atualidade, suas diferenças, e também é proposto um algoritmo de legalização chamado Jezz. Para comprovar a eficiência de tal algoritmo, foi realizada uma comparação entre ele e outros dois algoritmos de legalização, Tetris (HILL, 2002), que é um algoritmo clássico, e Abacus (SPINDLER; SCHLICHTMANN; JOHANNES, 2008), que é um algoritmo semelhante a Jezz, que havia sido proposto como um algoritmo superior a Tetris no tocante ao distanciamento total das portas lógicas em relação às suas posições originais. Jezz é um algoritmo semelhante ao Abacus, sendo a diferença básica entre eles de que Jezz usa uma função linear (distância de Manhattan) para calcular o custo de mover células, enquanto Abacus usa uma função quadrática.

Palavras-chave: Legalização, Perturbação, Posicionamento, Standard-cell, Microeletrônica.

ABSTRACT

Legalization is one of the three stages in which logic gate placement is subdivided in the physical synthesis of an integrated circuit. This stage consists of selecting positions considered to be valid for the logic gates, that is, positions that are alligned to the rows that divide the circuit area and where there is no overlapping among the gates.

In addition to organizing the logic gates in valid positions, a well made legalization needs to provide a smooth transition between global placement, the first stage of placement, and detailed placement, the last stage, in such a way that the solution that had been reached in global placement is modified the least possible.

In this work, a study on the legalization algorithms is performed, covering algorithms present today and a classic one. Also, a legalization algorithm called Jezz is proposed.

To verify the effectiveness of it, a comparison between it and two others has been made, being these two Tetris (HILL, 2002), which is a classical algorithm, and Abacus (SPINDLER; SCHLICHTMANN; JOHANNES, 2008), which is an algorithm that is simmilar to Jezz, and that had been proposed as an algorithm superior to Tetris when it comes to overall displacement of the logic gates with respect to their original positions pre-legalization. Jezz is similar to Abacus, being the main difference between them that, whilst Jezz uses a linear function (Manhattan distance) to calculate the cost of displacing cells, Abacus uses a quadratic function.

Keywords: Legalization. Incremental. Disturbance. Placement. Minimum. Standard-cell. Microelectronics. Legalization. Disturbance. Placement.

LIST OF ABBREVIATIONS AND ACRONYMS

CAD	Computer Aided Design
EDA	Electronic Design Automation
FURG	Universidade Federal do Rio Grande
HPWL	Half Perimeter Wire-length
IC	Integrated Circuit
ICCAD	International Conference on Computer Aided Design
ISPD	International Symposium on Circuit Design
RTL	Register Transfer Level
UFRGS	Universidade Federal do Rio Grande do Sul
VLSI	Very Large Scale Integrated

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The design flow of a VLSI circuit consists of many steps, a flowchart can bee seen in figure 1.1.

In this work, the main focus is on physical design, a very complex process which is broken into steps, one of which is placement, when component positions are selected.

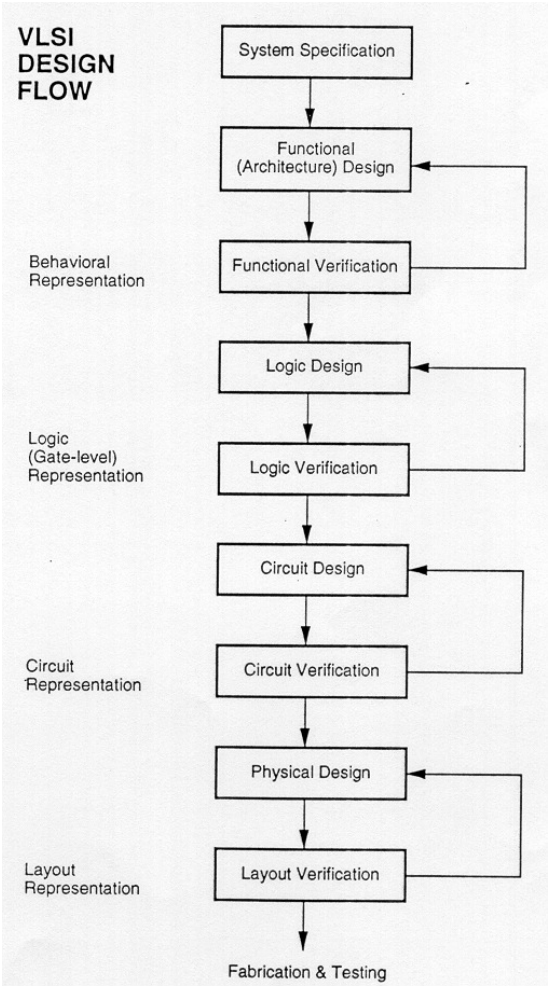

Figure 1.1 – Typical VLSI design flow. (MLYNEK; LEBLEBICI, 1998)

Physical design, as noted by (SHERWANI, 1993), is very complex, even being divided into conceptually easier steps. Concerning nowadays market requirements, that demand quick time-to-market, restricted design styles are used to reduce its complexity.

One of these design styles is standard-cell, a simpler style that consists of rectangular cells of same height, which is the type of cell that Jezz handles in legalization stage, however, the original concept of legalization is not limited to standard-cell, as all circuits with other design styles must be legalized.

With physical design complexity in mind, EDA tools come in handy. These kind of tools are CAD for electronics, in our case, microelectronics, that aid in the process of the conception of an integrated circuit. Algorithms are produced based on heuristics for executing the steps of the physical design. The main proposition of this work is an algorithm for EDA to execute the legalization step.

Within the physical synthesis of an integrated circuit, the placement step is responsible for selecting places to insert the circuit components, minimizing the wire-length connecting them. Moreover, placement algorithms also should try to avoid creating congested areas where the number of connections is higher than the available space to route them.

Typically, the placement step is divided into there phases: global placement, legalization and detailed placement. During the global phase, the overlapping restriction is relaxed, and a position for the components is first set. Although overlapping is allowed, the task of the global placement method is to spread cells reducing it, while minimizing wire-length, among other objectives.

The remaining overlapping is removed in the legalization step, where cells are also moved to legal positions within the circuit area, aligned to rows. Usually, the legalization process tries to displace the cells as little as possible as a way to preserve the initial solution obtained from global placement. After legalization, detailed placement makes fine adjustments to cell positions, performing optimizations that are hard to be seen during the global phase. This work presents a legalization method called Jezz, named after the 1992 game JezzBall (WIKIPEDIA, 2014), and its three versions, as it was improved over time. The first version of Jezz, 1.0, was used in our placement flow, UFRGS/FURG Brazil, which took the 1st place in ICCAD 2014 Incremental Timing-Driven Placement Contest.

Jezz can perform both full and incremental legalization, indicating the shift impact caused by inserting a cell in a row, because other cells might have to be shifted to make room for the new one. It intrinsically handles cell-to-site alignment and has blockage support. A cache system is used to allow fast look-up during incremental legalization, allowing Jezz to support detailed placement algorithms. The main contributions of this work are

- A full and incremental legalization method able to select the minimum displacement required to insert a cell in a row;
- support to incremental legalization;
- support to obstacles.

1.1 Outline of this thesis

This work is organized in 8 chapters. Chapter 2 emphasizes physical synthesis, for providing a brief background on the design flow step where legalization is inserted. Chapter 3 describes the nature of the legalization problem, while Chapter 4 describes the sate-of-the-art that we have in legalization. Chapter 5 describes the whole functioning of the legalization approach Jezz, Chapter 6 presents the results of the comparison made between Jezz and two other legalization approaches, Chapter 7 analyzes the complexity of the final version of Jezz after obtaining the runtime of the experimental results and Chapter 8 finishes with a conclusion.

2 THE PROCESS OF IC DESIGN AND PHYSICAL SYNTHESIS

As seen in Figure 2.1, there are many steps in the design flow of a VLSI circuit. This work focuses on legalization, which is a substage of placement, itself, a stage of physical design.

Physical design flow is also called the place and route because of its two main stages, that are interconnected. It is the back-end process of finding the physical location for the circuit components and interconnecting them.

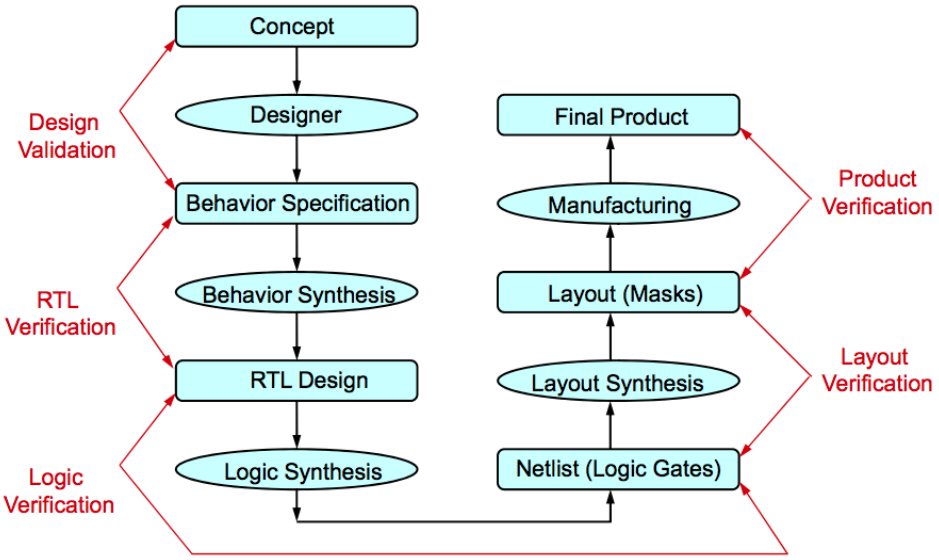

Figure 2.1 – VLSI design flowchart (NADA, 2011)

First, the circuit is partitioned, to form a number of macro-blocks. Then, in the floorplanning step, equivalently to making the floor plan of an apartment, the macro-blocks are placed on the layout surface to minimize area and interconnection length.

Then, comes the placement step, in which the components' physical locations are selected, within the available area in the circuit.

It can be driven to timing, area, power, and so on. The most important thing about placement, after selecting positions for the cells, naturally, is to reduce the wire-length of the cell interconnections. Mostly, this is made having the HPWL in mind, a most widely used approximation, which is half of the perimeter of the minimum bounding box that encloses all the pins of the cell net to be connected.

A poorly made placement can lead to unfeasible routing if one does not take into account that the cells must be nicely spread in such a way that there is little wire congestion, as shown in Figure 2.2.

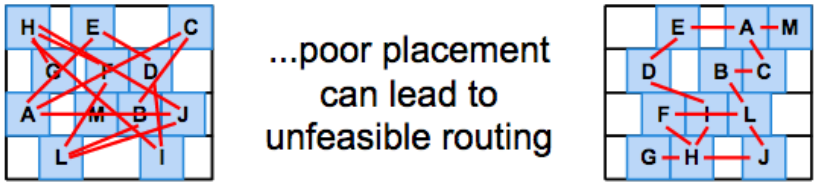

Figure 2.2 – On the left, a bad placement solution, full of congestion, and, on the right, a good one

After placement, global routing is performed. It determines coarsely the routes for the interconnections between the circuit cells.

Then, in the detailed routing step, the exact routes for the interconnection wires are determined.

The last step is the verification, to make sure everything is in agreement with the design rules, if not, manual fixes are performed.

Iteration an backtracking can be performed until the goals are reach for each individual step of the flow.

3 PROBLEM DEFINITION

Legalization is the process of aligning cells to valid positions and removing overlaps among them, while reducing the total cell movement, providing a smooth transition between global (rough) placement and detailed. The total cell movement, also the legalization cost, is defined by the sum of the Manhattan distance from the original cell positions (i.e. pre-legalization) to the final cell positions (i.e. pos-legalization) as shown in Equation (3.1).

$$\sum_{\forall cell} \left(|x^{cell}_{original} - x^{cell}_{legalized}| + |y^{cell}_{original} - y^{cell}_{legalized}| \right) \tag{3.1}$$

The circuit's placement area is divided in same-height rows, and each row, in same-width sites, as depicted in Figure 3.1. Cell widths are multiple of site widths, and only cells with height of one row are handled in this work, which is the most common case for standard-cell designs. To be considered at a valid position, a cell must be aligned vertically and horizontally to the sites within a row. The position of a cell is represented by its bottom-left corner.

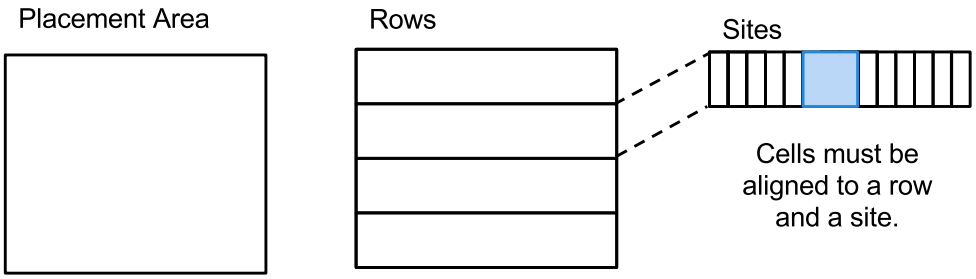

Figure 3.1 – Placement area is divided into rows and sites which define valid positions where cells can be placed.

3.1 Full and Incremental Legalization

Legalization can be made either fully or incrementally. Typically the input solution for full legalization is provided by the global placement step, whose goal is to roughly spread cells while optimizing wirelength, among other objectives. As the solution might contain overlaps among cells, legalization is performed circuit-wide, to eliminate all invalidity.

The output from the legalization process is given to detailed placement as an input, which, at last, modifies the solution to reach more refined results based on what the placement

step was primarily driven to. For example, placement for the ICCAD 2014 contest was timing-driven. Incremental legalization is the one that can be made during detailed placement, making fine adjustments to the former solution, switching one cell position with another's. Figure 3.2 shows that. The first circuit area on the upper left contains non-aligned cells in non-valid positions. The middle one, on the right, contains cells that are fully legalized, and the lower left one contains cells that are being incrementally legalized.

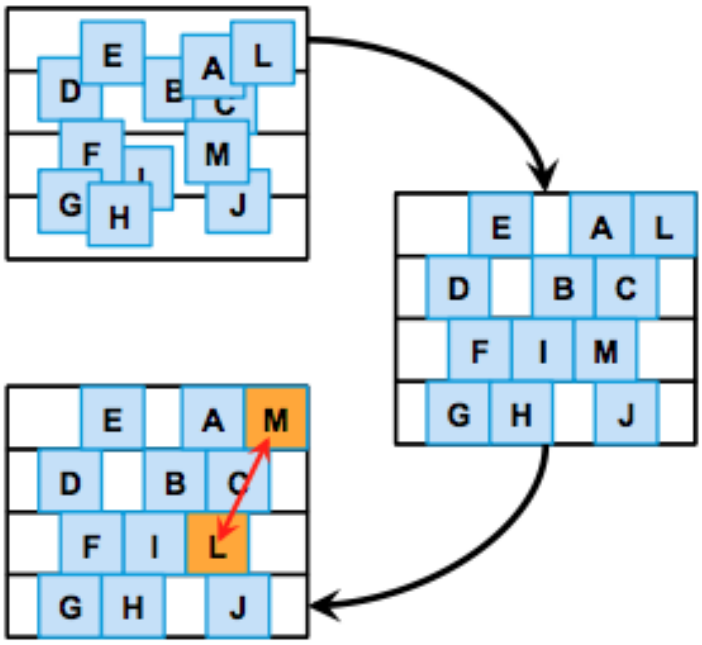

Figure 3.2 – An unorganized set of cells within a circuit and full and incremental legalization.

4 RELATED WORK

Tetris (HILL, 2002) is a classic greedy algorithm that legalizes one cell at a time and does not move cells that have already been legalized.

Abacus (SPINDLER; SCHLICHTMANN; JOHANNES, 2008) is greedy and moves cells that have already been legalized, using a quadratic function to calculate the best position for the cell. Jezz has a similar approach to Abacus, but it uses a linear function to calculate the cost of moving cells, which makes it simpler and more accurate, as it counts with the y position not only when comparing the cost of inserting in rows, but also uses the y position when calculating the displacement cost, because it uses the Manhattan distance.

There are many other methods for legalizing a circuit, as the main objective in a legalization is to remove node overlaps by displacing the nodes as little as possible. Two examples of which are the HiBin Legalizer (LEE; WU; CHIANG, 2010), which is a legalization based on bins and is greedy as well as the others, and BonnPlace (BRENNER, 2013), which is flow based.

4.1 Tetris Legalizer

Tetris (HILL, 2002) algorithm starts by grouping the cells in a vector and then sorting them in ascending order by their x-coordinate. After that, for each cell, it calculates the cost of moving it to each of the possible rows in the circuit. The cost is obtained by getting the Manhattan distance between the original cell position and the position it would occupy within the row. The best cost is changed dynamically by iterating over all the possible rows to insert the cell in, so it receives the smallest cost value found from iterating through the rows. After a cell is placed in the best row, the row's leftmost X position is increased by the added cell's length.

The process of legalizing a cell with this approach is outlined in algorithm 1. One advantage of it is that it is fast, one disadvantage of it may lead to very uneven row lengths, then, may fail to pack all components inside the placement region.

Algorithm 1: Tetris Legalizer

```
1  C = all cells to be legalized;
2  l_j = left-most position of each row j;
3  for each cell i in C sorted by x-position do
4      best_cost = ∞
5      for each row j in the circuit do
6          x = max{x_i, l_j}
7          cost = |x − x_i| + |y_j − y_i|;
8          if cost < best_cost then
9              best_cost = cost
10             best_row = j
11         end
12     end
13     x_i = max{x_i, l_best_row}
14     l_best_row = x_i + width_i
15 end
```

4.2 Abacus Legalizer

Abacus algorithm starts by grouping the cells in a vector and sorting them just as well as Tetris algorithm (see lines 1-2 in Algorithm 2). Sorting the cells according to their xposition can be done either in increasing or decreasing order. Both of them should be tested, because the results of each direction can be different. Experiments showed that the difference in the total movement between both sort directions is about 0.5%.

After that, it legalizes one cell at a time, keeping the cell as close as possible to the original position given by global placement, tentatively inserting the cell in certain rows until the best one is found, and then selecting a position for the cell horizontally. Each cell does not need not be moved over all rows, it is first moved to the nearest row according to the global position and then moved above and below it. If the lower bound exceeds the minimal cost of an already found legal position, then the movement of the cell over the rows can be stopped. This improves the runtime a lot. Jezz also stops when it finds that the cost is not improving while moving the cells to upper or lower rows.

Abacus also moves already legalized cells, which yields a lower overall displacement than Tetris, but using a quadratic function to calculate the new position a cell should occupy.

The process of legalizing a cell with the least displacement possible is outlined in algorithm 2. $PlaceRow$ is how they call the optimization of the total (quadratic) movement of all cells within one row, being the core of the approach, where they use dynamic programming.

Algorithm 2: Abacus Legalizer

```
1  C = all cells to be legalized;
2  l_j = left-most position of each row j;
3  for each cell i in C sorted by x-position do
4      best_cost = ∞;
5      for each row j in the circuit do
6          Insert cell i into row j;
7          PlaceRow j (trial);
8          Determine cost c;
9          if c < best_cost then
10             best_cost = c;
11             best_row = j;
12         end
13     end
14     Insert cell i in best_row;
15     Remove cell i from row j;
16     PlaceRow best_row(final);
17 end
```

Abacus is quite similar to Jezz in its approach, the main difference between them is that Abacus uses a quadratic function.

If the circuit has macros (non-standard cells), they assume that the macros are already placed overlap free. In addition, the rows blocked by macros are sliced in new rows (subrows), such that they are not blocked by macros anymore. Figure 4.1 shows that, for the comparison that they made with Tetris, more cells in Abacus move much less, whilst more cells in Tetris move much farther from their original positions.

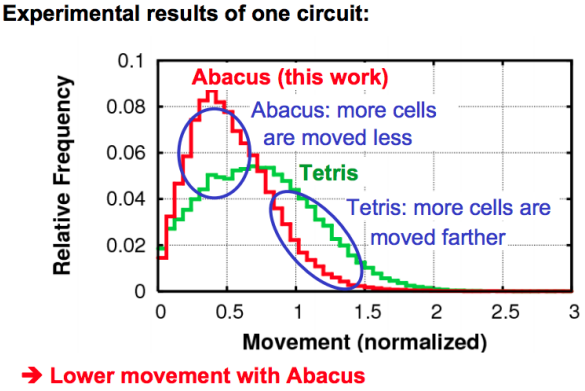

Figure 4.1 – Histogram for the comparison of Abacus and Tetris (SPINDLER; SCHLICHTMANN; JOHANNES, 2008)

4.3 HiBin Legalizer

HiBin is a hierarchical bin-based legalizer that is said to provide minimum disturbance that handles standard-cells and does not handle macro-blocks. For this approach, first, a chip is divided into several bins with equal size. Then, starting with the most crowded not yet legalized bin, they use a merging procedure to integrate the bins into a cross-shape or square-shape region until cell density in that region is less than a specific threshold value. Last, legalization is performed efficiently, preserving cell orders in each row and minimizing the weighted sum of movement distances. In order to minimize movement, HiBin, as well as Jezz and Abacus, moves already legalized cells. HiBin was compared to the state-of-the-art method Abacus and was better in terms of total overall displacement of cells by an average of 48%. It also reduces HPWL by 47%. As HPWL To accurately estimate the movement, the weighted sum of Manhattan distance movements of all cells is chosen as the disturbance metric instead of the Abacus metric, the total weighted quadratic movement of all cells.

4.4 BonnPlace

BonnPlace, part of the BonnTools (KORTE; RAUTENBACH; VYGEN, 2007) is a placement tool used in industry that has, in its legalization process, minimum cost flow and dynamic programming algorithms. It handles standard-cells and macro-blocks. It has a complex implementation, as it is also a placement tool. For legalization, it uses a sophisticated minimum cost flow approach and dynamic programming algorithms for the multiple knapsack problem (MARTELLO; TOTH, 1990). For macros, it legalizes small groups of up to four at a time, attempting to put macros close to each other in common groups. It considers the legalization process as part of the detailed placement, like an integrated phase, as shown in Section VIII (BRENNER, 2013, p 8). It uses the concept of zones, which are the maximal parts of the rows that are not blocked by fixed cells. The first phase of legalization ascertains that no zone contains more cells than can fit into it. The second places the cells within each zone in the given order. The third phase is the one of refining, switching positions of two given cells and doing more complicated actions, which is the incremental optimization.

4.5 Other Legalization Approaches

Domino (DOLL; JOHANNES; ANTREICH, 1994) is an approach that breaks the cells into subcells, all with same height and width, and breaks rows into places, and assigns the subcells to places in rows by solving a min-cost max flow.

The legalization in (BRENNER; VYGEN, 2004) acts similarly, althouth it assigns sets of modules to row regions.

Fractional Cut approach (AGNIHOTRI et al., 2003) improves recursive bisection based placement. To handle cut lines that are not limited to the row boundaries of the circuit, they developed a legalization algorithm based on dynamic programming. This combination lowered the wire-lengths produced by their placement tool. Hence, legalization is a complimentary algorithm to their fractional cut approach. The fractional cut is bisectional based and results in an uniform distribution of cell area.

Legalization has, as well, to assign cells to rows after rough placement, because the bi-sectional based approach, as mentioned, has cut lines that are not limited to the row boundaries. It is done by using dynamic programming. The approach operates on a row-by-row basis. Then, the cells of each row are packed from left to right.

5 JEZZ LEGALIZER

Jezz, the method proposed in this work, is a standard-cell circuit legalization approach that minimizes overall displacement. It has been improved through time. There are actually four versions of Jezz, but the first one is not going to be addressed here. This one, that we may call version 0.0, was used in our winning flow for the ICCAD 2014 Timing Driven Placement Contest (ICCAD..., 2014). This version was part of the placement tool uPlace that we developed for this contest.

There is one thing one must bear in mind, and it is that this is a much simpler version than the others covered here, as it does not even have the concept of calculating the best shifting cost by sweeping left and right vectors, as we are going to show in this section.

Hence, only three versions of Jezz will be presented here. Yet, there are some implementation concepts that have not been changed since version 1.0.

In Jezz, the sites mentioned in Chapter 3, seen in Figure 3.1, are unit-wide, that is, the sites that subdivide the rows are considered to have the width of 1 unit.

The concept of insertion of the node in a row is the same in all versions, that is, when there are nodes in the way, they have to be shifted to make room for the new node. What changes through the versions is the way the cost for moving the nodes and the optimum cost for choosing the best new location is calculated. Therefore, the structures and concepts that have not been altered are shown in separate sections.

5.1 Data Structures

The logic gates to be legalized are called nodes. Jezz defines three types of nodes: (1) *white-space*, (2) *blockage* and (3) *cell*. Blockage and cell nodes have constant widths, while white-space widths can be dynamically adjusted.

White-space and cell nodes are free to be moved, while blockage nodes are fixed, as they are macro-blocks or nodes that somehow cannot be moved to another position in any circumstance. Macro-blocks, actually, are not mandatorily fixed, but, generally, they have their positions decided prior to placement, hence, they are seen as fixed during legalization.

Cell nodes model movable standard-cells, and blockage model fixed standard-cells or obstacles (e.g. macro-blocks, uneven row widths).

A row is represented using a double-linked list of nodes. Every list has, at least, one node, and the sum of the node widths equals the row width. An "empty" row has one and only

one white-space node, with the same width of the row.

By construction, no two white-space nodes can be neighbors within a row. If this happens during the legalization process, the white-space nodes are merged into one. Also, a zero-width white-space is not allowed, so that any zero-length white-space is automatically deleted.

Jezz works only with integer positions (i.e. row and site indexes), which intrinsically handle the cell-to-site alignment.

5.2 Cache Memory

All the versions count with a cache memory system that has not been changed since version 1.0. The goal of this system was to aid in the process of looking up a certain node within a row. For that, the row itself is divided into regions, and, to ease the process of executing the algorithm and making shifts.

Rows are subdivided into same-width regions, to allow fast look-up to a node at certain x position. It reduces sequential search overhead, since the original cell list does not divide the row in regions. For each region, a pointer to a node inside that regions is stored. There is one cache vector for each row, and they are computed during full legalization. The way Jezz uses this system is to use the cached node as the starting point for the sequential search. Each reagion keeps a pointer to a node that is potentially within it, and, as Jezz does not keep the cache always consistent to avoid unnecessary overhead, this pointer may be invalid.

If the node is displaced horizontally, the cache pointer may point to a node outside of its respective region. For little displacement, that should be fine, as we still get a node close to the aimed region anyway.

If the node was moved to another row, the pointer gets invalid, and Jezz looks at the pointer in neighboring regions. The cache pointer to a region is updated when a look-up at that region is performed.

Figure 5.1 shows the cache system, pointing inside the row regions.

However this may help when an incremental legalization is being made, for a full legalization, when cells are inserted by x-position, the cache system does not need to be used, as it is easier to search nodes sequentially from the extremity of the linked list, since they are very likely to be the last/first one.

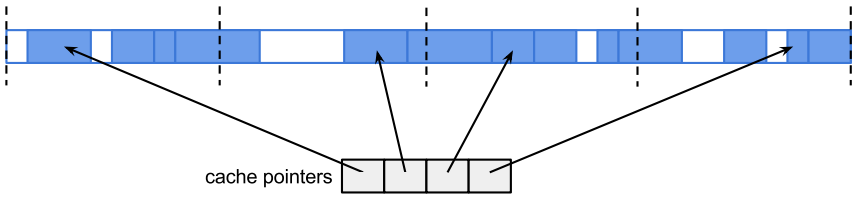

Figure 5.1 – Cache System

5.3 Legalization

Jezz legalizes one cell at a time, being a greedy algorithm (SARRAFZADEH; WONG, 1996, p 21). Given a cell and a row, the impact caused by the insertion of the cell in such row is computed. Jezz uses the impact information to select the best row to actually place the cell.

In a *full legalization*, all cells are processed in a specific order, in this case, from left to right (increasing x position).

For this work, we consider only processing cells by increasing original x position, as usually is done by legalization methods, however, Jezz could process cells in any order.

In full legalization, we only insert the cell in the rightmost position available in the row, making shifts to the left if necessary. In incremental legalization, there might be neighboring cells to the right as well as left.

In an *incremental legalization*, cells are assumed to be already legalized, and fine adjustments are performed iteratively to the full legalization already made, so shifts to the right can also be performed. This is typically used by detailed placement algorithms.

Since the cells are not necessarily inserted in any specific order of x, the impact on already legalized cells must be computed to left and right.

5.3.1 Node Insertion

There are three main types of insertion cases that can occur during legalization. When the cell's new position happens to be located in between a white-space node, the white-space is broken into two new white-spaces, and the cell is placed between them, as shown in Figure 5.2.

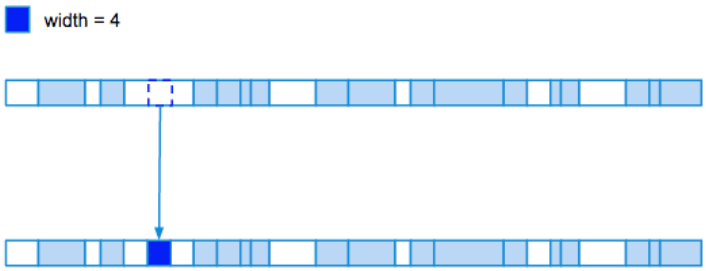

Figure 5.2 – Insertion of a cell in between a white-space node.

Another case is when there is a blockage node enclosing the position for the cell, as shown in Figure 5.3. As a consequence, the length of the blockage node in the way is added to the distance to be shifted.

Figure 5.3 – Insertion of a cell enclosed by blockage.

The last case is when there are movable nodes in the way, either occupying the whole space where the cell should be inserted or just part of it, as Figure 5.4 shows.

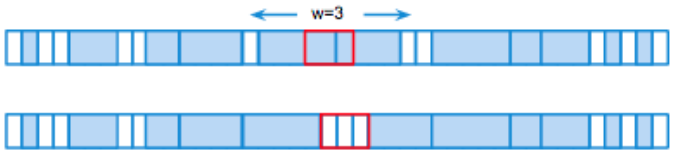

Figure 5.4 – Insertion of a cell when there are other nodes in the way.

In that case, we divide the row between its left and right sides. The cell can be shifted to either sides.

Jezz computes the cost of positioning the cell, because other cells may have to be shifted by one or more sites in order to make room for the new one.

Two cost vectors (left/right) are computed. The kth element in the vector represents the accumulated cost necessary to open k sites to such side.

The size of the side vector is always the width of the cell to be inserted plus 1, because the vector goes from zero shifts needed to a number of shifts needed equal to the width of the cell.

The total cost is the summation of the shifts to the right and to the left. Note that the summation of the displacement to the left and to the right is chosen in such a way that the number o white-space units equals the width of the node to be inserted.

The case in Figure 5.4 is the one in which the concept is still the same regarding the room that needs to be made for the new cell to be inserted, but what changes through Jezz versions is the way it chooses the best combination of shifts and which cells are to be moved to make room for the new one, which will be presented in the next sections. For this case, as we are considering movable cells, the terms *cell* and *node* will be used interchangeably, while a white-space node will be explicitly referred to as $white - space$. All the versions use two cost vectors, one for each left/right side in which the row is divided, and populate the vectors based on different calculations.

5.4 Jezz Version 1.0

Version 1.0 is a simple one compared to the next two, but already deals with the concept of combining accumulated costs for determining the best option of shifts for node insertion.

It provides a simple calculation for the costs for the case cited previously in 5.3.1, in which one or more cells need to be shifted to make room for the new one. The new node is supposed to be inserted in between the node that overlaps its left edge and the immediate right neighbor (in the case of an incremental legalization, as there will be no right neighbor in full legalization). Starting at the left and right nodes, two sequential searches are then performed to check the cost of shifting nodes to open the required room.

This version counts with the concept of weight of the node, which means that one node to be inserted can be of priority to move, or, prior to legalization, the designer could have decided that the node must preferably not move, or be as close as possible to its original position, hence, the concept of weight could come in handy, for it can be inserted in the calculation in order to give preference to moving one node in detriment of another.

The full legalization algorithm used is outlined in 3.

Algorithm 3: Full Legalization

```
1  for each cell c sorted by increasing x-position do
2      r_0 = nearestRow(c)
3      best_impact = computeImpact(c, r_0)
4      best_row = r_0
5      for each row r above r_0 do
6          impact = computeImpact(c, r);
7          if impact < best_impact then
8              best_impact = impact
9              best_row = r
10         else
11             break;
12         end
13     end
14     for each row r below r_0 do
15         // see lines 6-12
16     end
17     insertCell(cell, best_row);
18 end
```

During full legalization, if we sort cells in ascending order of X-positions, we will be making legalization from left to right, so, when legalizing a node, overlaps may only be happening on the left side, as there will be no right neighbors for that node. As a consequence of it, full legalization ends up being the process of appending new nodes and removing overlaps to the left, hence, the cost vector for the right direction may be empty.

For each cell, the impact is first computed for the nearest row with respect to the original cell position. Then, rows above and below the initial row are sequentially analyzed from the closest to the farthest to the initial row. Finally, the cell is placed in the row with the least impact on the legalization cost.

5.4.1 Impact Computation

The search algorithm sequentially sweeps the nodes seeking for white-spaces. The search stops when the accumulated width of white-spaces is greater or equal to the width of the new node or when there is no more space left. During the search, a cost vector is filled such that the element at index k indicates the cost of shifting nodes by k units. The left and right cost vectors are then used to define the optimum shifting amount for each direction. This procedure is outlined in Algorithm 4. The term e in line 13 represents the weight of the node.

Figure 5.5 depicts the right and left cost vector computation to open room to the red node assuming the node weights are all one. To open a single space to left, only one node needs to be shifted. Therefore the shifting cost is one as reported by $left[1]$. Similarly when two spaces are required to left, three nodes need to be shifted. However note that the first node will shift two spaces while the last two nodes only need to be shifted one space. So that, the overall cost is 4 as reported by $left[2]$.

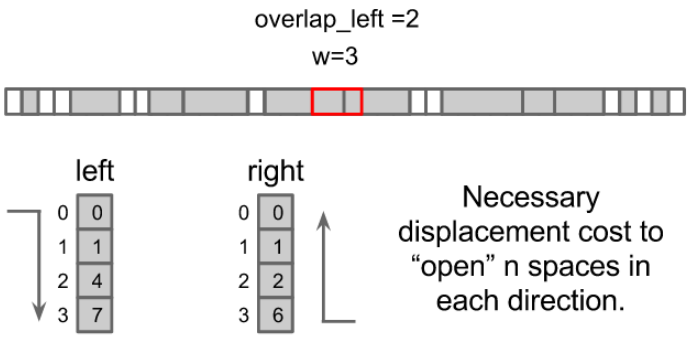

Figure 5.5 – Calculation for minimum displacement cost.

Algorithm 4: Cost Vector Computation

```
1  node = reference node;
2  disrupt[0..w] = {0, · · · , 0};
3  overflow = w;
4  e = 0;
5  k = 0;
6  while (node and overflow > 0) do
7      if node is blockage then
8          break;
9      end
10     if node is whitespace then
11         offset = width(node) > overflow? overflow : width(node);
12         for i = 0; i < offset; i++ do
13             disrupt[k+1] = e + disrupt[k];
14             k++;
15         end
16         overflow = overflow - offset;
17     else
18         e += weight(node);
19     end
20     node = previous(node);
21 end
```

5.4.2 Optimum Cost

Once the cost vectors are computed, one needs to choose the optimum amount to shift left and right. This is performed by sweeping the two vectors in opposite directions as presented in Algorithm 5. Note that, by traversing the vectors in opposite directions, the amount of open spaces always matches the new node width. The combination of left and right shifts with the lowest cost, plus the cost of inserting the node itself (represented by $e \times max(0, |overlap\_left - i|)$), is then selected, as seen in line 9.

Algorithm 5: Optimum Shift

```
1  max_l = w − overflow_l // maximum available space to left (up to w);
2  max_r = w − overflow_r // maximum available space to right (up to w);
3  if max_l + max_r < w then
4  |  overflow;
5  end
6  best_cost = ∞;
7  best_i = −1;
8  for i = w − maxr; i ≤ max_l; i++ do
9  |  cost = left[i] + e × max(0, |overlap_left − i|) + right[w − i];
10 |  if cost < best_cost then
11 |  |  best_cost = cost;
12 |  |  best_i = i;
13 |  end
14 end
15 displace left cells by best_i to left;
16 displace right cells by w − best_i to right;
17 displace current cell by overlap_left − best_i; // negative displacement means a left
   displacement
```

5.4.3 Experimental Results

Jezz version 1.0 was only compared with Tetris in full legalization, Abacus was not present in the work yet. Both Tetris and Jezz algorithms follow the same general premise, in which the cells are ordered by ascending x coordinate and reinserted in the circuit one by one. Each cell is inserted within the row that generates the lowest cost at the time.

Table 5.1 summarizes features that Jezz and Tetris have or not. Node weighting, in line 4, is the possibility to make a node more important to be place with more priority than the others. It is represented by the $weight$ term e in Algorithm 4.

Table 5.1 – Jezz and Tetris comparison in features

Feature	**Jezz**	**Tetris**
superfast	no	yes
incremental	yes	no
node weighting	yes	no
overflow control	yes	no
blockage support	yes	no

For the experiment, each benchmark has been primarily randomly placed and then tested for the legalizers. This procedure has been made a hundred times for each benchmark. The tests were performed in a machine with 64 bit processor Intel Core i7, 3.4GHz, running Ubuntu

version 14.04. The tool used was written in language C++. The benchmarks used were the ones available from ICCAD 2014 Incremental Timing-Driven Placement Contest. Information about each of the benchmarks is shown in Table 5.2. Table 5.3 shows a comparison between the legalizers Tetris and Jezz, after the experiment, in terms of elapsed time during the execution of the algorithm and the total displacement of the cells within the circuit for different benchmarks.

Table 5.2 – Information about the benchmarks

Benchmark	**#Cells**	**#I/O Pins**	**Has macro-blocks**	**Dimensions (μm)**	**Density**
b19	219268	47	No	1187.2x1188	0.76
leon2	794286	700	No	2086.4x2086	0.7
leon3mp	649191	333	No	1989.2x1990	0.7
vga_lcd	164891	184	No	898.6x898	0.7

The table above shows that, even though Jezz supports fixed macro-blocks, the comparison has only been done with standard-cells, because Tetris only does not provide support for macro-blocks.

Table 5.3 – Comparison between Tetris and Jezz legalizers

	Tetris		Jezz	
benchmark	**execution time (s)**	**displacement (μm)**	**execution time**	**displacement**
b19	50.999	2.33727e+07	10.0963	2.97269e+07
leon2	326.41	9.1051e+07	46.0866	1.13496e+08
leon3mp	250.128	6.93014e+07	32.9324	8.64554e+07
vga_lcd	30.3778	1.63691e+07	7.70431	1.95958e+07

As we can see from Table 5.3, Jezz's performance was worse than Tetris in terms of displacement, yet its runtime was much faster than Tetris, especially, for circuits with a large number of cells. This may be due to the fact that Tetris searches for the best row by parsing every possible row, not only parsing the adjacent rows. Jezz, on the other hand, only parses the adjacent rows and it stops when it finds the best one and also it splits the available options for placing the new cell in left and right sides. Tetris has a better displacement of all the cells within the circuit, while Jezz has a better overall knowledge of the circuit's available space, and it keeps a cache memory.

One explanation for Tetris having better displacement is that it does not move cells that had already been legalized. Jezz does, hence, when one cell is to be inserted within a row, some other cells may be moved as well. Albeit they move by small distances, only to make room for the new one, this counts as overhead in displacement.

Another explanation is that Tetris does not take overflow into account, while Jezz does, displacing the cells to avoid overflow.

5.5 Jezz Version 2.0

In this new version of Jezz, that is in (PUGET et al., 2015), the way it populates the cost vectors and computes the minimum shifting cost is different from version 1.0.

So far, only the distance to be shifted was taken into account. Now, if, when shifting the blue node shown in the lowest row in Figure 5.6, it happens to be getting nearer its original position that it had in global placement, the cummulated cost for shifting k sites is decreased by the amount with which the distance to the original position diminishes. For example, if it is getting two sites closer to its original position, the cost for shifting k sites is decreased by two. On the other hand, if it is getting farther to its original position, the cost is increased by the amount, so, in the example, if it were to get two sites farther from its original position, the cost would be increased by two.

Figure 5.7 shows with more clarity how a node might be getting nearer or farther from its global position pre-legalization. The green node gets to be 4 positions away from its original position, hence, the cost of making such change ends up in increasing the cost by 4 units.

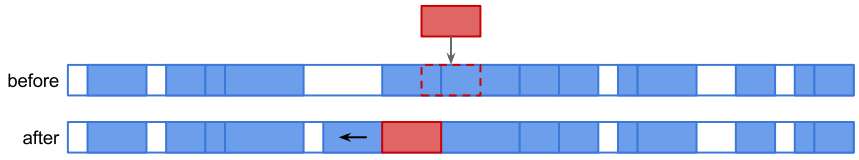

Figure 5.6 – Cell insertion

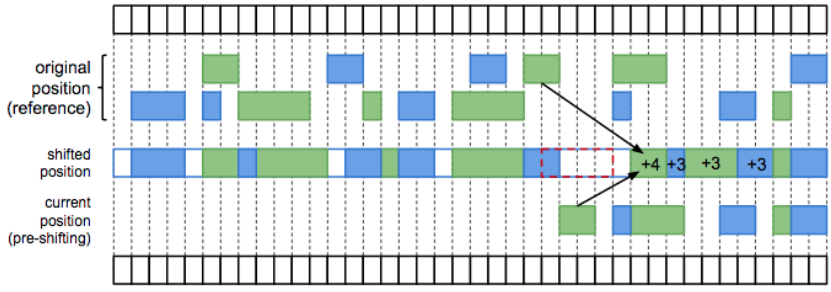

Figure 5.7 – Cell moving away from its original position during legalization.

5.5.1 Impact Computation

The impact is the change in the cost function due to the insertion of a cell in a row as measured by Equation (3.1).

The node is assumed to be inserted just after the node that encloses its left edge as shown in Figure 5.6. This assumption keeps the relative ordering of cells, which is a well known property in terms of maintaining the placement quality. As we make the decision of inserting the node in that certain position, it breaks the combinatorial nature of the legalization problem. Still, multiple choices arise when inserting a cell in between two other cells. One may shift all cells to the left or to the right or choose any combination of left and right shifts. The main contribution of Jezz is to select the optimum combination of left and right shifts.

To do so, Jezz computes two impact vectors that indicate the impact of shifting nodes to left ($impact_l$) and to right ($impact_r$), as in the previous version. The k^{th} element of each vector reports the impact of opening k sites to the left/right, meaning that other nodes might have to be shifted as well to make room for the new cell. The size of the vector is the same as the width of the cell, w, being inserted, plus one, as the maximum number of spaces required in each direction is bounded by the cell width.

The impact computation for the right direction is outlined in Algorithm 6. A similar algorithm is used for the left direction. Nodes are sequentially processed from the reference node, which is the immediate neighbor of the node to be legalized. In Figure 5.8, being the node in red border the one to be legalized, the reference node to the left would be the blue one that overlaps with it.

Every time a cell node is visited, the impact of shifting it left by 0 up to k sites is accumulated in the impact vector (lines 11-13). The search stops when the accumulated width of whitespaces is greater or equal to the width of the cell being inserted, or when there is no more space left.

Algorithm 6: Impact Computation

```
1  node = reference node;
2  impact_r[0..w] = {0, ..., 0};
3  k = 0;
4  while (node and k < w) do
5      if node is blockage then
6          break;
7      end
8      if node is whitespace then
9          k += min{width(node), w − k}
10     else
11         for i = k + 1, d = 1; i ≤ w; i++, d++ do
12             impact_r[i] +=
               weight^node × (|x^node_original − (x^node_current + d)| − |x^node_current − x^node_original|)
13         end
14     end
15     node = next(node);
16 end
17 overflow_r = w − k
```

Figure 5.8 shows an example of the impact vector computation made in incremental legalization, because of the neighbouring cells on the right side. In full legalization, it would be the same concept, but there would be no cells to the right.

At the top, the original positions of the nodes, $x^{node}_{original}$, are represented. At the bottom, the current positions of nodes, $x^{node}_{current}$, are depicted. In the middle, the shifted position of nodes at the right side of the cell being inserted is shown for a displacement of 4 sites.

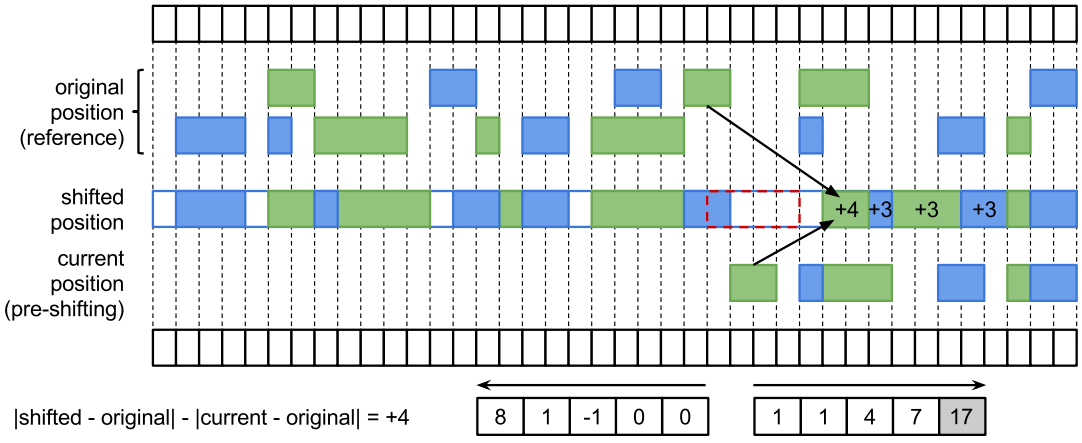

Figure 5.8 – Impact vector computation

To open a single space to the right, only one node needs to be shifted. Therefore, the shifting impact is one, as reported by $impact_r[1]$. If, when shifting by one site to make room for

the new node, the node being shifted is approaching its original position, the cost is decreased by one. If, on the other hand, the node is getting farther from its original position, the cost is increased by one, and so on, that is why we see an impact of +4 when moving the green node pointed by arrows 4 sites to the right from its previous position.

Still as in version 1.0, each index in the cost vector for each side is computed by the summation of the costs of moving each of the cells in the way aside.

5.5.2 Optimum Shift

Again, as in version 1.0, once the cost vectors are computed, one needs to choose the optimum amount to shift left and right. This is performed by sweeping the two vectors in opposite directions, as presented in Algorithm 7. What differs from the previous version is the way it is done. As explained, the algorithm now questions if a node is approaching or not its original position, so the shifting cost can not only be increased, but also dynamically reduced.

Note that, by traversing the vectors in opposite directions, the amount of open spaces always matches the new node width. The combination with the lowest cost of left and right shifts plus the movement of the cell being inserted itself is then selected.

Algorithm 7: Optimum Shift

```
1  max_l = w − overflow_l // maximum available space to left (up to w);
2  max_r = w − overflow_r // maximum available space to right (up to w);
3  if max_l + max_r < w then
4  |  overflow;
5  end
6  best_cost = ∞;
7  best_i = −1;
8  for i = w − maxr; i ≤ max_l; i++ do
9  |  cost = left[i] + e × max(0, |overlap_left − i|) + right[w − i];
10 |  if cost < best_cost then
11 |  |  best_cost = cost;
12 |  |  best_i = i;
13 |  end
14 end
15 displace left cells by best_i to left;
16 displace right cells by w − best_i to right;
17 displace current cell by overlap_left − best_i; // negative displacement means a left
   displacement
```

5.5.3 Experimental Results

For the experiment, different benchmarks were used from the ones in the experimental results of version 1.0. The benchmarks used were the ones available from ISPD 2002 benchmark suite.

These other benchmarks were chosen over the previous ones because this suite provides 18 different benchmarks, which is a good quantity of circuits for the experiment, and, most of all, because the number of cells in the benchmarks increases almost linearly, starting from over 12k to 210k cells. This contributes to a good knowledge of Jezz's performance in terms of runtime feasibility and overall displacement improvemet for smaller and larger circuits that we would not have had we chosen the previous benchmarks.

The procedure of primarily randomly placing each benchmark and testing it has been made a hundred times for both legalizers. The tests were performed in a machine with 64 bit processor Intel Core i7, 3.4GHz, running Ubuntu version 14.04. Jezz was implemented using language C++.

This comparison does not contain Abacus as well as the previous one, because the main focus here was first to fix the bad overall displacement results and compare again with the same algorithm to see if the new implementation worked out.

As previously remarked, all the results from all the versions have been obtained from full legalization, not incremental. Table 5.4 shows the number of cells each benchmark circuit has and a comparison between the legalizers Tetris and Jezz, after the experiment, in terms of elapsed time during the execution of the algorithm and the total displacement of the cells within the circuit for each different benchmark.

Table 5.4 – Comparison between Jezz and Tetris

Bench	# Cells	Jezz		Tetris		Improv.
		Time (s)	Total Disp (μm)	Time (s)	Total Disp (μm)	
ibm01	12506	0.029	1.57E+05	0.001	2.50E+05	37.47%
ibm02	19342	0.033	2.03E+05	0.001	3.29E+05	38.23%
ibm03	22853	0.046	2.76E+05	0.002	4.53E+05	39.07%
ibm04	27220	0.055	3.21E+05	0.002	5.20E+05	38.26%
ibm05	28146	0.050	2.96E+05	0.002	4.70E+05	36.95%
ibm06	32332	0.059	3.29E+05	0.002	5.55E+05	40.67%
ibm07	45639	0.089	4.92E+05	0.003	8.21E+05	40.12%
ibm08	51023	0.095	4.97E+05	0.004	8.22E+05	39.55%
ibm09	53110	0.107	5.99E+05	0.004	9.93E+05	39.67%
ibm10	68685	0.151	8.33E+05	0.005	1.33E+06	37.51%
ibm11	70152	0.147	7.77E+05	0.006	1.30E+06	40.03%
ibm12	70439	0.157	8.66E+05	0.005	1.38E+06	37.33%
ibm13	83709	0.182	9.12E+05	0.007	1.52E+06	40.11%
ibm14	147088	0.316	1.53E+06	0.014	2.54E+06	39.57%
ibm15	161187	0.319	1.61E+06	0.014	2.68E+06	39.85%
ibm16	182980	0.391	1.90E+06	0.018	3.10E+06	38.78%
ibm17	184752	0.406	2.12E+06	0.017	3.40E+06	37.74%
ibm18	210341	0.408	2.07E+06	0.020	3.39E+06	38.88%
					Average	38.88%

As we can notice from Table 5.4, Jezz 2.0 has a much longer execution time than Tetris, but it does not take more than half a second to execute, even for larger circuits. Also, the overall displacement of cells from their original position is diminished by almost 40% comparing to Tetris, so it performs a smooth transition between global and detailed placement, as well, if it must shift cells, it prefers to shift the ones that will be closer to their original positions, which is an addendum to the previous version optimum position computation.

The histogram in Figure 5.9 shows that most of the cells from one of the benchmarks are displaced, at most, by 10 μm, so most cells are displaced by small distances with Jezz. Tetris displaces many cells by larger distances, being up 50k displaced by at least 20 μm.

This version, consequently, seems much better than the previous one in terms of displacement, though it cannot be fully asserted because different benchmarks were used for the experimental results in the previous version.

Nonetheless, it now has a much longer runtime. Because of the improvements in the algorithm, it has more criteria to ascertain during execution, such as the increasing of the accumulated cost by approaching an original position.

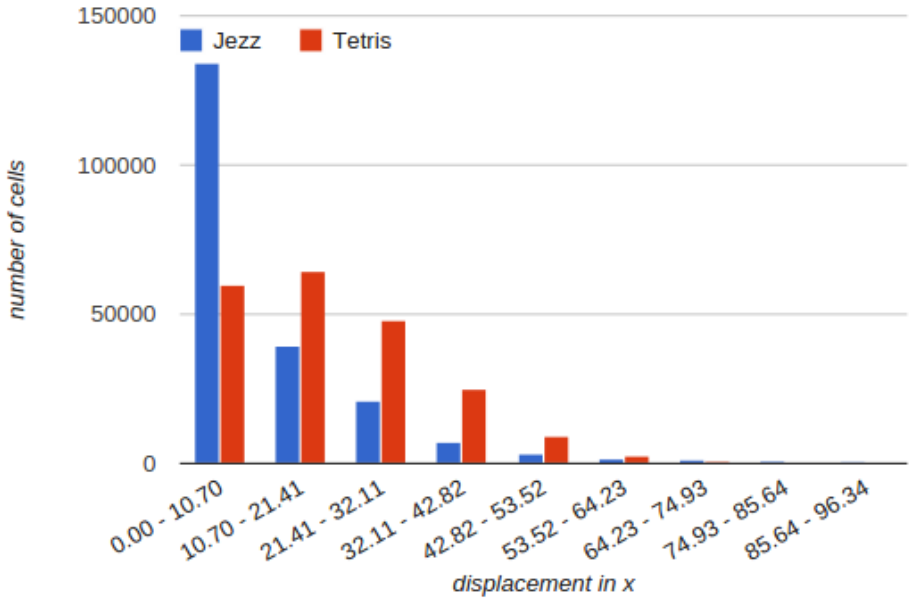

Figure 5.9 – Histogram of cell displacement for a benchmark

5.5.4 Complexity Analysis

The problem of finding a global optimum position for each cell is very difficult. It is part of the placement flow, known to be NP-complete. Depending on the size of a circuit, reaching a global minimum can become unfeasible. We propose a method for reaching a good local minimum by calculating the overall displacement that must be made in order to make room for the new cell within a row. The nested loops that Algorithm 6 has can give us slower runtime for some cases and a posisbility of quadtratic complexity in worst case, but, on average, as we can notice in the graph of Figure 5.10, the runtime increases almost linearly with number of cells.

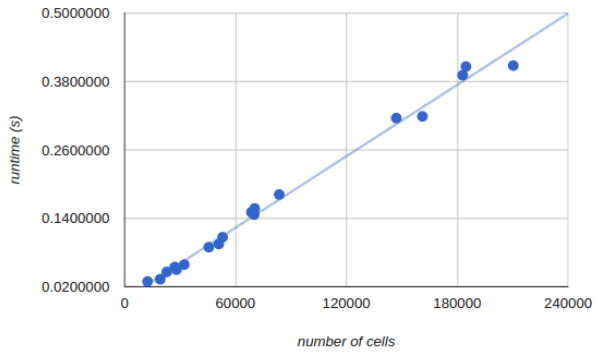

Figure 5.10 – Jezz runtime increases almost linearly with the number of cells

5.6 Jezz Version 3.0

The full legalization approach used in the final version is outlined in Algorithm 8. The difference from the previous versions is noted in lines 11-16, where a tie-break criterion is inserted. If the best impact so far is the same as the current impact, the row that gives this impact is chosen as best row, just as would happen if its impact were smaller than the current best impact.

To avoid analyzing too many rows, the analysis is stopped (line 19) if the impact of a row does not improve the lowest impact found up until now. This has little influence in quality, but improves the runtime significantly.

The impact on cells already legalized in the row is computed as shown in Algorithm 8. If the cost of inserting the cell within a row is the same as another, Jezz uses a tie-break criterion of choosing the one with the least maximum displacement.

Algorithm 8: Full Legalization

```
1  for each cell c sorted by increasing x-position do
2      r_0 = nearestRow(c)
3      best_impact = computeImpact(c, r_0)
4      best_row = r_0
5      for each row r above r_0 do
6          impact = computeImpact(c, r);
7          if impact < best_impact then
8              best_impact = impact
9              best_row = r
10         end
11         else if impact = best_impact then
12             if curr_max_disp < smallest_max_disp then
13                 best_impact = impact
14                 best_row = r smallest_max_disp = curr_max_disp
15             end
16         end
17     end
18     else
19         break;
20     end
21 end
22 for each row r below r_0 do
23     // see lines 6-20
24 end
25 insertCell(cell, best_row);
```

5.6.1 Impact Computation

The impact computation for the right direction is outlined in Algorithm 6 in Section 5.5. A similar algorithm is used to the left direction. It has not changed since version 2.0.

Figure 5.11 shows an example of the impact vector computation. At the top, the original positions of the nodes, $x^{node}_{original}$, are represented. At the bottom, the current positions of nodes, $x^{node}_{current}$, are depicted. In the middle, the shifted position of nodes at the right side of the cell being inserted is shown for a displacement of 4 sites.

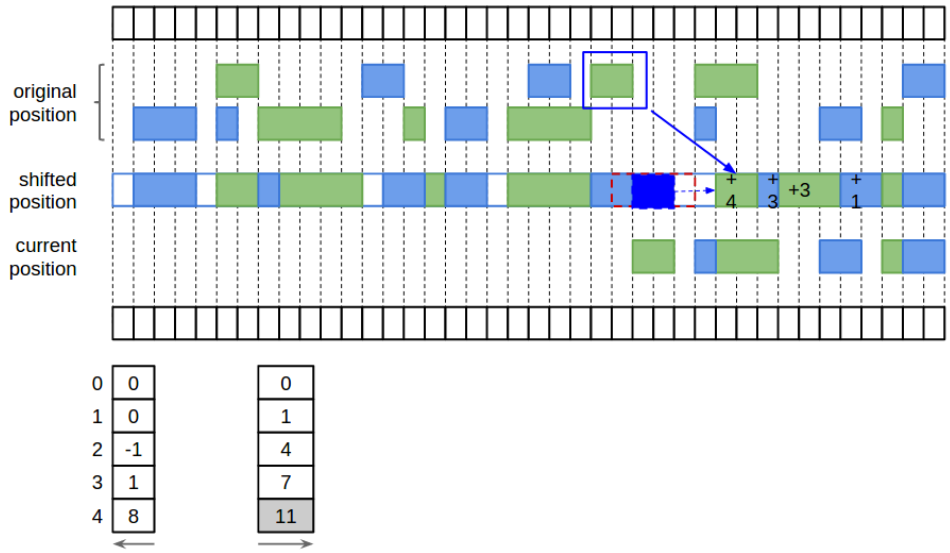

Figure 5.11 – Impact vector computation for the right side

5.6.2 Optimum Shift

As explained before, once the cost vectors are computed, one needs to choose the optimum amount to shift left and right, so, the two cost vectors are sweeped in opposite directions, as presented in Algorithm 9.

If the full legalization is being performed, the right cost vector does not count, as it only might be necessary to move already legalized cells to the left in order to make space for the new one, which is always assigned to the first rightmost available position.

Algorithm 9: Optimum Shift

$max_l = w - overflow_l$ // max available sites to left
$max_r = w - overflow_r$ // max available space to right
if $max_l + max_r < w$ **then**
    exit(overflow);
end
$best\_cost = \infty$;
$smallest\_max\_disp = \infty$;
for $i = max_l$; $i \geq w - max_r$; i- - **do**
    $curr\_cost = |(x_{original} + w) - i|$;
    $cost = impact_l[i].cost + weight \times curr\_cost + impact_r[w - i].cost$;
    $curr\_max\_disp = max(max(impact_l[i].disp, impact_r[w - i].disp, |node\_y\_disp + curr\_cost \times step|)$;
    if $cost < best\_cost$ **then**
        $best\_cost = cost$;
        $shifts\_left = i$
    end
    if $cost = best\_cost$ **then**
        if $curr\_max\_disp = smallest\_max\_disp$ **then**
            $shifts\_left = i$ $smallest\_max\_disp = curr\_max\_disp$;
        end
    end
end
displace left nodes by $shifts\_left$ to left;
displace right nodes by $shifts\_left - i$ to right;

The optimum shift calculation in version 3.0 takes into account diminishing the maximum cell displacement when choosing the best cost. In each circuit, there is a cell that is displaced more than the others, so we compute the displacement for each cell and store it in the second position of the now double impact vector (each index of the vecor stores two values). The first value, that appears in Algorithm 9 as $impact\_l.cost$ is the one that we use to compute accumulated cost, and the second value, that appears as $impact\_l.displ$ is the one where we store the maximum displacement that we got so far (for each side), and it chances dynamically.

We then compute the biggest displacement by calculating the maximum one for both cost vectors and the node being inserted itself. It is shown in line 11, and $|node_{yd}isp + curr_{c}ost \times step|$ mean the node displacement in y-axis and its x-axis displacement, given by the current cost, but multiplied by the step factor that is used in the algorithm, as all the distances that we work with for legalization are integer ones and the row-alligned sites are unit-wide, so, for acquiring the maximum displacement correct value, we must do this multiplication.

The maximum displacement calculation is used in the algorithm as a criterion for cost tie-break and it helps in achieving a smaller maximum displacement. This is the main difference of the optimum cost calculation in version 3.0, as the other ones did not have tie-break criteria for optimum cost (version 2.0 had for the cost computation) and do not try to diminish maximum displacement.

The experimental results are shown in the next section, as this is the final version of the algorithm.

6 EXPERIMENTAL RESULTS

In order to show the effectiveness of the latter version of the algorithm Jezz, 3.0, an experiment of comparing it with two other algorithms, Tetris and Abacus, was performed, using 18 benchmark circuits for comparison. The Abacus version used here is also the one implemented for the ICCAD Timing Driven Placement Contest 2014.

Abacus was chosen because it and Jezz have a similar approach towards legalization, and Tetris was chosen because it is a classic algorithm, whereas a study on legalization algorithms was performed, and Tetris is sort of an historic root for legalization.

As in the other versions, the comparison was made with full legalization, as detailed placement was not performed, hence, no incremental legalization has been made. We considered the legalization step after global placement, so the comparison only takes into account the first two stages of placement.

For the experiment, each benchmark has been primarily globally placed using mPL6 (CHAN et al., 2005) placement algorithm, from the placement package mPL, and then tested. The tests were performed in a machine with 64 bit processor Intel Core i7, 3.4GHz, running Ubuntu version 14.04.

The benchmarks used were the ones available from ISPD 2002. They previously contained macro-blocks (that we call blockage nodes), but not anymore after a variation of them was made with FastPlace(VISWANATHAN; CHU, 2005), which were the ones used here, because a comparison with Tetris algorithm would be useless, as it does not provide support for them.

6.1 In Terms of Runtime

Table 6.1 shows the number of cells each circuit from the benchmarks has and a comparison between the legalizers Tetris, Jezz and Abacus in terms of elapsed time during the execution of each algorithm.

Table 6.1 – Runtime for Jezz, Tetris and Abacus

Benchmark	#Cells	Jezz	Tetris	Abacus
ibm01	12506	0.038755	0.000905	0.004535
ibm02	19342	0.075944	0.001409	0.007149
ibm03	22853	0.100474	0.001705	0.008749
ibm04	27220	0.140916	0.003052	0.011423
ibm05	28146	0.13774	0.002067	0.011268
ibm06	32332	0.126951	0.002347	0.013185
ibm07	45639	0.200308	0.003432	0.021171
ibm08	51023	0.23508	0.003814	0.021967
ibm09	53110	0.197032	0.004259	0.029005
ibm10	68685	0.390358	0.005276	0.033493
ibm11	70152	0.361115	0.005777	0.035308
ibm12	70439	0.333709	0.005418	0.040669
ibm13	83709	0.390846	0.006478	0.041311
ibm14	147088	0.702293	0.01374	0.10258
ibm15	161187	0.631168	0.013228	0.114065
ibm16	182980	0.877742	0.016378	0.129681
ibm17	184752	0.923253	0.016558	0.12982
ibm18	210341	1.06586	0.019803	0.144765

Although it is clear to see that Jezz takes much longer to execute than the other two, it shows itself to take a little longer than a second to execute, even for a larger circuit, which has around 200k cells, ibm18.

One of the main reasons why Tetris is much faster is that it does not move already legalized cells, hence its simplicity. Jezz moves already legalized cells within a row to make room for the new cell to be inserted by combining movements to the left and right.

Note the nesting of for loops in the algorithms of Jezz outlined in the previous section. This gives us the almost certain possibility of having a slower runtime than Abacus, which finds the optimum cost by solving a quadratic program which can be transformed to a fast dynamic program.

Graph 7.1 shows an almost linear relationship between runtime and number of cells for Jezz, the runtime scales well when the number of cells grows, yet this may not be the case for a big number of cells used in the industry.

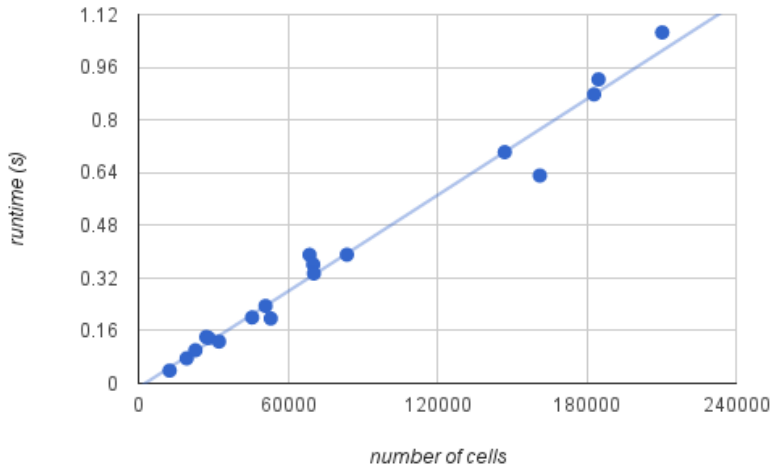

Figure 6.1 – Jezz runtime increases almost linearly with the number of cells.

6.2 In Terms of Minimizing Displacement

As previously mentioned, a good algorithm for legalization needs to provide a smooth transition between the first and last placement stages, global placement and detailed placement, which means that the solution reached after global placement that is given as input to the legalization stage must change as little as possible during this stage.

The latter version of Jezz, 3.0, has produced the following results seen in Table 6.2 and Table 6.3 in comparison with Tetris and Abacus in terms of displacement.

Table 6.2 shows the overall displacement of the whole, being that the summation of all the shifts made by the cells within the circuit.

Table 6.2 – Overall displacement for Jezz, Tetris and Abacus

Benchmark	Jezz	Tetris	Abacus	Jezz/Tetris	Jezz/Abacus
ibm01	95553	159663	102275	40.15%	6.57%
ibm02	159081	269146	160152	40.89%	0.67%
ibm03	215381	372928	224151	42.25%	3.91%
ibm04	185665	318344	196780	41.68%	5.65%
ibm05	182408	333974	183292	45.38%	0.48%
ibm06	210849	383065	220412	44.96%	4.34%
ibm07	336495	570545	351739	41.02%	4.33%
ibm08	337813	607089	336468	44.36%	-0.40%
ibm09	397210	679333	414455	41.53%	4.16%
ibm10	604523	1030000	608412	41.31%	0.64%
ibm11	578916	987000	602938	41.32%	3.98%
ibm12	632063	1050000	632967	39.80%	0.14%
ibm13	745599	1240000	763361	39.87%	2.33%
ibm14	1010000	1850000	1020000	45.41%	0.98%
ibm15	1020000	1860000	1050000	45.16%	2.86%
ibm16	1260000	2320000	1280000	45.69%	1.56%
ibm17	1400000	2450000	1410000	42.86%	0.71%
ibm18	1450000	2630000	1450000	44.87%	0.00%
			avg	42.69%	2.38%

As we can see from table 6.2, Jezz has been improved since the first version and now reaches a gain of almost 43% in terms of overall displacement when compared to Tetris, and 2.38% when compared to Abacus.

The overall displacement is a good measure to verify the propensity of the algorithm to provide the smooth transition from global to detailed placement, because a smaller overall displacement means that the whole solution has been less altered.

Besides that, to refine even more the property of the algorithm of altering the solution as little as possible, some other results can be verified.

Table 6.3 shows the average displacement, which means the average distance that a cell shifts during the legalization stage. The average values show that Jezz is better than Tetris in almost 40% and very timidly better than Abacus. Jezz is actually much better than the classical algorithm Tetris in overall and average displacement and still manages to be better than Abacus, even though it is by a very small factor, so Abacus' advantage in runtime has not been overcome by Jezz improvements in displacement.

Table 6.3 – Average displacement for Jezz, Tetris and Abacus

Benchmark	**Jezz**	**Tetris**	**Abacus**
Benchmark	Jezz	Tetris	Abacus
ibm01	7.64057	12.7669	8.17807
ibm02	8.22464	13.9151	8.28001
ibm03	9.42463	16.3186	9.80838
ibm04	6.8209	11.6952	7.22924
ibm05	6.48078	11.8658	6.51219
ibm06	6.52137	11.8479	6.81715
ibm07	7.37297	12.5013	7.70698
ibm08	6.6208	11.8983	6.59444
ibm09	7.47901	12.7911	7.80371
ibm10	8.80138	15.0319	8.858
ibm11	8.25231	14.0627	8.59474
ibm12	8.9732	14.9246	8.98603
ibm13	8.90704	14.8599	9.11922
ibm14	6.84717	12.584	6.94302
ibm15	6.32908	11.5336	6.537
ibm16	6.90109	12.662	7.02094
ibm17	7.5522	13.2716	7.61113
ibm18	6.87771	12.4853	6.91497
avg.	7.557	13.167	7.750

Describing the behavior seen in table 6.3, Figure 6.2 shows that most of the cells from the benchmark $ibm05$ (chosen for no specific reason) are displaced no more than 10 μm, so most cells are displaced by small distances with Jezz, which contributes to a small average displacement. The benchmark $ibm05$ has 28146 cells, and more than 18k cells move up to as much as 10 μm after full legalization. Table 6.3 shows that, for all the benchmarks, the average displacement for Jezz is actually under 10 μm, for Abacus as well, but Tetris does not have one average value that's lower than 11.5 μm.

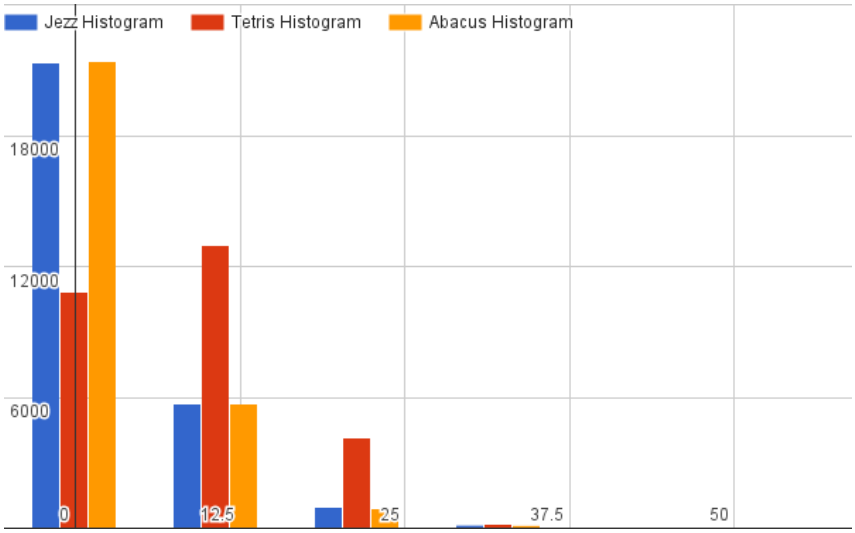

Figure 6.2 – Histogram of cell displacement for $ibm05$

Abacus has a similar performance concerning the displacement the biggest number of cells make, being worse than Jezz by a small factor, but Tetris is much more distributed than both of the former, as we can notice in the graph by seeing the red bars. It means that Tetris alters the original solution locally, in cell shifts, more than the other two. Jezz and Abacus show that more cells move less, and Tetris shows that more cells move more than in the other two approaches.

Naturally, it happens because Tetris algorithm is focused only on the cell to be inserted and the way it is going to disturb the solution, but it does not take into account its neighbouring cells, whereas the neighbouring cells that have already been legalized shall not be moved. This simpler approach helps the algorithm in being superfast, as noticed in Table 6.1.

Table 6.4 shows the maximum displacement seen for a cell in the circuit after the legalization process. Despite the fact that Jezz is better than Tetris and Abacus in terms of overall displacement and provides an overall minimum displacement, maximum displacement is a comparison term in which Jezz did not come out much better than the others. One justification for that is that Jezz addresses maximum displacement in such a manner that it gives preference to choosing the cost that comprises the smallest maximum displacement, but it might not have been done in the most proper way, or at least in a way that reduces the values by a big factor.

What is being done is that Jezz stores the biggest maximum displacement it finds in each vector index, changing it iteratively whenever one of the nodes in the way would give us a bigger maximum displacement.

Then, it gives preference to choosing the combination of vector indexes (optimum cost calculation) that gives us the smallest maximum displacement value. So, this improves maximum displacement for certain benchmarks, but may also be worse for some of them, because

there might be an incongruity in storing the biggest maximum displacement value and choosing the smallest one of them all. Maybe this approach should be done differently, having some other aspect in mind.

Table 6.4 – Maximum displacement for Jezz, Tetris and Abacus

Benchmark	**Jezz**	**Tetris**	**Abacus**
Benchmark	Jezz	Tetris	Abacus
ibm01	58.0	59.0	57.0
ibm02	57.0	60.0	63.0
ibm03	63.0	57.0	85.0
ibm04	66.0	62.0	67.0
ibm05	50.0	38.0	48.0
ibm06	61.0	55.0	57.0
ibm07	61.0	61.0	58.0
ibm08	61.0	50.0	57.0
ibm09	97.0	59.0	71.0
ibm10	64.0	70.0	57.0
ibm11	66.0	63.0	72.0
ibm12	63.0	72.0	63.0
ibm13	75.0	66.0	79.0
ibm14	60.0	61.0	57.0
ibm15	68.0	58.0	61.0
ibm16	70.0	64.0	68.0
ibm17	80.0	69.0	67.0
ibm18	71.0	59.0	56.0
avg.	66.1667	60.1667	63.5

One can think that, as Jezz is just slightly better than Abacus in overall displacement and is far slower in runtime, it should show much better results than Abacus in displacement to make up for the fact that it is slower. Although Jezz uses a linear function (Manhattan distance) to calculate cost and Abacus uses a quadratic one, there is still a big disparity when it comes to runtime, so simplicity of implementation and better displacement results might be not enough as advantages after seeing runtime results depending on the need of the designer.

Jezz scales well with the increase of number of cells, so its runtime is still feasible, but it is certain that Abacus leaves Jezz behind in runtime aspect. This may have to do with the nested conditionals that Jezz uses in its algorithms, for example, in the main loop of impact computation Algorithm 6. However, Jezz is simple to implement and still managed to show a slightly better result than Abacus. Thus, in a matter of choosing the one with best displacement results, Jezz could be chosen, though, if runtime is very important, Abacus would be better.

6.3 Further Possible Improvements

A change could be made for the tie-break criterion when there is a tie between two positions for inserting the cell, as mentioned in this section. In such way, Jezz could be even better than both other approaches in displacement measures, however, legalization is still supposed to be a transition stage. A full legalization algorithm does not have the task of being driven to something such as timing, area, power, and so on. It is actually the opposite, as it receives a solution that has been coarsely reached in global placement in order to fulfill some objectives such as these ones, and it must not alter it while legalizing.

Hence, there is only so much one can do to improve full legalization, as, in reality, it must change as little as possible the original solution to get it to the detailed placement, so optimizations can be made, but one can hit a certain plateau.

When legalization is integrated to a detailed placement algorithm, incremental legalization can be made (which, as previously said, Jezz provides) being timing-driven, for example. As said, “In order to exploit the global placement solution, detailed placement approaches only make local changes on the current placement.” in (SARRAFZADEH; WANG; YANG, 2003, p. 37), being incremental legalization part of the local changes.

7 COMPLEXITY ANALYSIS

Graph in Figure 7.1 shows that, in version 3.0 of Jezz, runtime increases almost linearly with the number of cells. VLSI cell placement problem is known to be NP-complete, and legalization is a part of it. As this is a kind of algorithm based on heuristics, it reaches a good local minimum by calculating the overall displacement that must be made in order to insert a new cell within a row, as well as it prefers to shift cells that are reaching their original position pre-legalization, contributing to the process of finding a good local minimum, reducing the average displacement of cells.

On average, as we can notice in the graph shown in figure 7.1, the runtime increases almost linearly with number of cells.

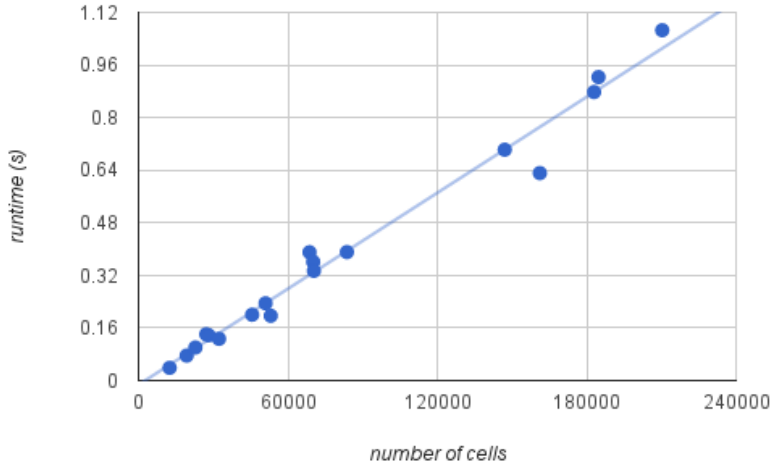

Figure 7.1 – Jezz runtime increases almost linearly with the number of cells

8 CONCLUSION

In this work, we proposed an approach called Jezz for legalizing cells within a circuit. Legalization is the middle of three stages in the placement of the cells of an integrated circuit.

The first stage is global placement, which aims at generating a rough placement solution that relaxes some design constraints, e. g. there may be overlaps among cells. The middle stage, legalization, was the covered in this work. The last stage is detailed placement, which further improves the legalized placement solution in an iterative manner by rearranging cells locally while keeping others fixed. In this stage, an incremental legalization can be made.

Legalization must provide a smooth transition between the first and the last stages of placement. For that reason, the main aim of Jezz is to provide minimum overall displacement of cells in its legalization. The way Jezz executes the insertion of a cell in a new position is dividing the region for positioning in its left and right sides, and searching for a better place to insert the cell in both sides. White-spaces are considered as a type of node, as well as obstacles, and the former are split when the cell is to be placed in the center of them. We also compared this approach with Tetris and Abacus in terms of runtime and total displacement.

For the sake of showing the effectiveness of the algorithm, a comparison has been made for 18 benchmarks between Jezz and two other algorithms, Tetris and Abacus, and Jezz showed itself to be better than both in overall displacement, almost 43% better than Tetris and 2.38% better than Abacus.

As a disadvantage, Jezz is worse in runtime, yet its runtime scales well with circuits with larger numbers of cells. This approach has a linear complexity for runtime, because runtime grows much close to linearity with the growth in number of cells, as shown in Chapter 6.

One can think that, as Jezz is just slightly better than Abacus in overall displacement and is far slower in runtime, it should show much better results than Abacus in displacement (as it did with Tetris) to make up for the fact that it is slower. In a matter of choosing the one with best displacement results, Jezz could be chosen, though, if runtime is very important, Abacus would be better. Also, whichever of them could be chosen over Tetris, that was worse than both of them in displacement.

REFERENCES

SARRAFZADEH, M. and WANG, M. and YANG, X. 2011. Available from Internet: <<http://www.ee.ncu.edu.tw/~jfli/vlsi21/lecture/ch01.pdf/>>. Visited 2015 Jul 5.

AGNIHOTRI, A. R. et al. Fractional cut: Improved recursive bisection placement. In: **ICCAD**. IEEE Computer Society / ACM, 2003. p. 307–310. ISBN 1-58113-762-1. Available from Internet: <http://dblp.uni-trier.de/db/conf/iccad/iccad2003.html#AgnihotriYKMOM03>.

BRENNER, U. Bonnplace legalization: Minimizing movement by iterative augmentation. **IEEE Trans. on CAD of Integrated Circuits and Systems**, v. 32, n. 8, p. 1215–1227, 2013. Available from Internet: <http://dblp.uni-trier.de/db/journals/tcad/tcad32.html#Brenner13>.

BRENNER, U.; VYGEN, J. Legalizing a placement with minimum total movement. **IEEE Trans. on CAD of Integrated Circuits and Systems**, v. 23, n. 12, p. 1597–1613, 2004. Available from Internet: <http://dblp.uni-trier.de/db/journals/tcad/tcad23.html#BrennerV04>.

CHAN, T. F. et al. mpl6: a robust multilevel mixed-size placement engine. In: GROENEVELD, P.; SCHEFFER, L. (Ed.). **ISPD**. ACM, 2005. p. 227–229. ISBN 1-59593-021-3. Available from Internet: <http://dblp.uni-trier.de/db/conf/ispd/ispd2005.html#ChanCRSSX05>.

DOLL, K.; JOHANNES, F. M.; ANTREICH, K. Iterative placement improvement by network flow methods. **IEEE Trans. on CAD of Integrated Circuits and Systems**, v. 13, n. 10, p. 1189–1200, 1994. Available from Internet: <http://dblp.uni-trier.de/db/journals/tcad/tcad13.html#DollJA94>.

Dwight Hill. **Method and system for high speed detailed placement of cells within an integrated circuit design**. 2002. US 6370673 B1.

ICCAD 2014 Timing Driven Placement Contest. 2014. Available from Internet: <<http://cad_contest.ee.ncu.edu.tw/CAD-contest-at-ICCAD2014/problem_b/results/ICCAD2014_Contest_P2_Results.pdf/>>. Visited 2015 Jul 5.

KORTE, B.; RAUTENBACH, D.; VYGEN, J. Bonntools: Mathematical inno- vation for layout and timing closure of systems on a chip. In: **Proc. of IEEE**. [S.l.]: IEEE, 2007. v. 95, n. 3, p. 555–572.

LEE, Y.-M.; WU, T.-Y.; CHIANG, P.-Y. A hierarchical bin-based legalizer for standard-cell designs with minimal disturbance. In: **ASP-DAC**. IEEE, 2010. p. 568–573. ISBN 978-1-60558-837-7. Available from Internet: <http://dblp.uni-trier.de/db/conf/aspdac/aspdac2010.html#LeeWC10>.

MARTELLO, S.; TOTH, P. **Knapsack Problems**. [S.l.]: John Wiley and Sons, 1990. Available from Internet: <<http://www.or.deis.unibo.it/kp/Chapter6.pdf/>>. Visited 2015 Jul 5.

MLYNEK, D.; LEBLEBICI, Y. **Design of VLSI Systems**. 1998. Available from Internet: <<http://emicroelectronics.free.fr/onlineCourses/VLSI/index.html>>. Cited 2015 Jul 5.

PUGET, J. et al. Jezz: An effective legalization algorithm for minimum displacement. In: **South Symposium on Microelectronics, SIM 2015**. [S.l.: s.n.], 2015.

SARRAFZADEH, M.; WANG, M.; YANG, X. **Modern Placement Techniques**. 1. ed. [S.l.]: Kluwer Academic Publishers, 2003.

SARRAFZADEH, M.; WONG, C. K. **An Introduction to VLSI Physical Design**. [S.l.]: McGraw-Hill, 1996.

SHERWANI, N. **Algorithms for VLSI Physical Design Automation**. 1. ed. [S.l.]: Kluwer Academic Publishers, 1993.

SPINDLER, P.; SCHLICHTMANN, U.; JOHANNES, F. M. Abacus: Fast legalization of standard cell circuits with minimal movement. In: **Proceedings of the 2008 International Symposium on Physical Design**. New York, NY, USA: ACM, 2008. (ISPD '08), p. 47–53. ISBN 978-1-60558-048-7. Available from Internet: <http://doi.acm.org/10.1145/1353629.1353640>.

VISWANATHAN, N.; CHU, C. C. N. Fastplace: efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. **IEEE Trans. on CAD of Integrated Circuits and Systems**, v. 24, n. 5, p. 722–733, 2005. Available from Internet: <http://dblp.uni-trier.de/db/journals/tcad/tcad24.html#ViswanathanC05>.

WIKIPEDIA. **JezzBall**. 2014. [Online; accessed 08-Nov-2014]. Available from Internet: <http://en.wikipedia.org/wiki/JezzBall>.

APPENDIX - Trabalho de Conclusão I
Jezz: An Effective Legalization Algorithm For Minimum Displacement

Julia Casarin Puget[1], **Ricardo Reis**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)

Abstract. *This work has the objective of introducing the subject chosen for the proposed graduation work, a legalization algorithm for standard cells, presenting what has been implemented, which relies on constructing the algorithm Jezz for legalizing standard cells using a linear function, and comparing it to two other legalization algorithms, Tetris, a classic algorithm, and Abacus, an algorithm that has a similar approach to Jezz, but uses a quadratic function. Jezz showed itself to be almost 43% better than Tetris and 2.38% better than Abacus in terms of overall displacement, although it is slower in runtime, yet, feasible, presenting an average complexity of O(n). As this work is strongly based on the legalization stage of a full placement step within a physical synthesis, a study on legalization algorithms and their different approaches towards solving the legalization problem is also presented.*

1. Introduction

Within the physical synthesis, the placement step is responsible for selecting places to insert the circuit components, minimizing the wirelength connecting them. Moreover, placement algorithms also should try to avoid creating congested areas where the number of connections is higher than the available space to route them.

Typically, the placement step is divided into there phases: global placement, legalization and detailed placement. During the global phase, the overlapping restriction is relaxed, and a position for the components is first set. Although overlapping is allowed, the task of the global placement method is to spread cells reducing it, while minimizing wirelength, among other objectives.

The remaining overlapping is removed in the legalization step, where cells are also moved to legal positions within the circuit area, aligned to rows. Usually, the legalization process tries to displace the cells as little as possible, since the initial solution is supposed to correlate well with the final solution. After legalization, detailed placement makes fine adjustments to cell positions, performing optimizations that are hard to be seen during the global phase.

This work presents a legalization method called Jezz, used in our placement flow, UFRGS/FURG Brazil, which te ook the 1st place in ICCAD 2014 Incremental Timing-Driven Placement Contest. Jezz is named after the 1992 game JezzBall [**?**]. Jezz can perform both full and incremental legalization, indicating the shift impact caused by inserting a cell in a row, because other cells might have to be shifted to make room for the new one. It intrinsically handles cell-to-site alignment and has blockage support. A cache system is used to allow fast look-up during incremental legalization, allowing Jezz to support detailed placement algorithms.

The main contributions of this work are

- A full and incremental legalization method able to select the minimum displacement required to insert a cell in a row;
- support to incremental legalization;
- support to obstacles.

2. Related Work

Tetris [Hill 2002] is a classic greedy algorithm that legalizes one cell at a time and does not move cells that have already been legalized, therefore, not fit for use during detailed placement, a stage that often requires incremental legalization. It also iterates over all the available rows searching for the best position for each cell. Abacus is greedy and moves cells that have already been legalized, using a quadratic function to calculate the best position for the cell. Jezz has a similar approach to Abacus, but it uses a linear function to calculate the cost of moving cells, which makes it simpler and more accurate, as it counts with the y position not only when comparing the cost of inserting in rows, but also uses the y position when calculating the displacement cost, because it uses the Manhattan distance. There are many other methods for legalizing a circuit, as the main objective in a legalization is to remove node overlaps by displacing the nodes as little as possible. Two examples of which are the HiBin Legalizer [Lee et al. 2010], which is a legalization based on bins and is greedy as well as the others, and BonnPlace [Brenner 2013], which is flow based. There is, as well, [Cong and 0004 2008], which handles mixed-size cell legalization.

3. Problem Definition

Legalization is the process of aligning cells to valid positions and removing overlaps among them, while reducing the total cell movement. The total cell movement, also the legalization cost, is defined by the sum of the Manhattan distance from the original cell positions (i.e. pre-legalization) to the final cell positions (i.e. pos-legalization) as shown in Equation (1).

$$\sum_{\forall cell} \left(|x^{cell}_{original} - x^{cell}_{legalized}| + |y^{cell}_{original} - y^{cell}_{legalized}| \right) \tag{1}$$

The circuit's placement area is divided in same-height rows, and each row, in same-width sites, as depicted in Figure 1. Cell widths are multiple of site widths, and only cells with height of one row are handled in this work, which is the most common case for standard-cell designs. To be considered at a valid position, a cell must be aligned vertically and horizontally to the sites within a row. The position of a cell is represented by its bottom-left corner.

4. Data Structures

Jezz defines three types of nodes: (1) *whitespace*, (2) *blockage* and (3) *cell*. Blockage and cell nodes have constant widths, while whitespace widths can be dynamically adjusted. Whitespace and cell nodes are free to be moved, while blockage nodes are fixed. Cell nodes model movable standard-cells, and blockage model fixed standard-cells or obstacles (e.g. macro-blocks, uneven row widths).

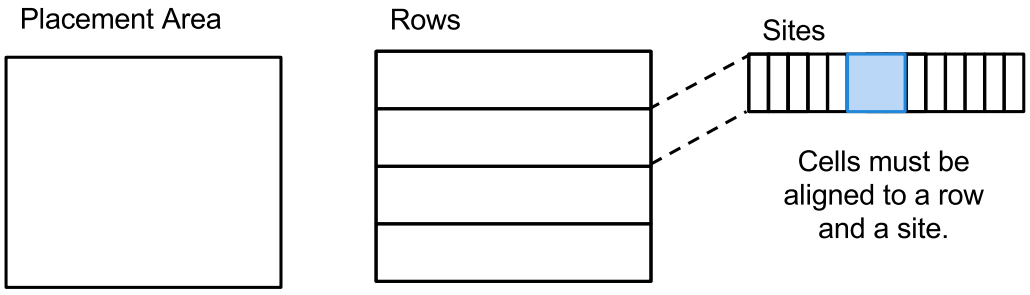

Figure 1. Placement area is divided into rows and sites which define valid positions where cells can be placed.

A row is represented using a double-linked list of nodes. Every list has, at least, one node, and the sum of the node widths equals the row width. An ”empty” row has one and only one whitespace node, with the same width of the row.

By construction, no two whitespace nodes can be neighbors within a row. If this happens during the legalization process, the whitespace nodes are merged into one. Also, a zero-width whitespace is not allowed, so that any zero-length whitespace is automatically deleted.

Jezz works only with integer positions (i.e. row and site indexes), which intrinsically handle the cell-to-site alignment.

5. Jezz Legalization

Jezz legalizes one cell at a time. Given a cell and a row, the impact caused by the insertion of the cell in such row is computed. Jezz uses the impact information to select the best row to actually place the cell. The impact computation is detailed in Section 6.

In a *full legalization*, all cells are processed in a specific order. For this work, we consider only processing cells by increasing original x position, as usually is done by legalization methods, however, Jezz could process cells in any order.

The full legalization approach used in this work is outlined in Algorithm 1. For each cell, the impact is first computed for the nearest row w.r.t. the original cell position. Then, rows above and below the initial row are sequentially analyzed from the closest to the farthest to the initial row. Finally, the cell is placed in the row with the least impact on the legalization cost. The impact on cells already legalized in the row is computed as shown in Algorithm 2. If the cost of inserting the cell within a row is the same as another, Jezz chooses the one with less maximum displacement.

To avoid analyzing too many rows, the analysis is stopped (line 19) if the impact of a row does not improve the lowest impact found up until now. This has little influence in quality, but improves the runtime significantly.

In full legalization, we only insert the cell in the rightmost position available in the row, making shifts to the left if necessary, in order to move the cell as little as possible related to its position from global placement. In an *incremental legalization*, cells are assumed to be already legalized, and fine adjustments are performed iteratively to the full legalization already made, so shifts to the right can also be performed. This is typ-

Algorithm 1: Jezz Full Legalization

```
1  for each cell c sorted by increasing x-position do
2      r_0 = nearestRow(c)
3      best_impact = computeImpact(c, r_0)
4      best_row = r_0
5      for each row r above r_0 do
6          impact = computeImpact(c, r);
7          if impact < best_impact then
8              best_impact = impact
9              best_row = r
10         end
11         else
12             if impact = best_impact then
13                 best_impact = impact
14                 best_row = r
15             end
16         end
17         else
18         end
19         break;
20     end
21     for each row r below r_0 do
22         // see lines 6-18
23     end
24     insertCell(cell, best_row);
25 end
```

ically used by detailed placement algorithms. Since the cells are not necessary inserted in any specific order of x, the impact on already legalized cells must be computed to left and right. After that, the optimum cost for inserting the cell is computed in algorithm 3 selecting the amount of shifts to left and to right.

6. Impact Computation

The impact is the change in the cost function due to the insertion of a cell in a row as measured by Equation (1).

The node is assumed to be inserted just after the node that encloses its left edge as shown in Figure 2. This assumption keeps the relative ordering of cells, which is a well known property in terms of maintaining the placement quality. As we make the decision of inserting the node in that certain position, it breaks the combinatorial nature of the legalization problem, which makes it become NP-complete.

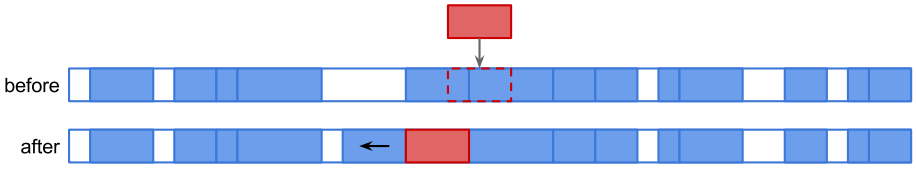

Figure 2. Cell Insertion

Still, multiple choices arise when inserting a cell in between two other cells. One may shift all cells to the left or to the right or choose any combination of left and right shifts. The main contribution of Jezz is to select the optimum combination of left and right shifts.

To do so, Jezz computes two impact vectors that indicate the impact of shifting nodes to left ($impact_l$) and to right ($impact_r$). The k^{th} element of each vector reports the impact of opening k sites to the left/right, meaning that other nodes might have to be shifted as well to make room for the new cell. The size of the vector is the same as the width of the cell, w, being inserted, plus one, as the maximum number of spaces required in each direction is bounded by the cell width.

The impact computation for the right direction is outlined in Algorithm 2. A similar algorithm is used to the left direction. From the reference node, nodes are sequentially processed. Every time a cell node is visited, the impact of shifting it left by 0 up to k sites is accumulated in the impact vector (lines 11-13). The search stops when the accumulated width of whitespaces is greater or equal to the width of the cell being inserted or when there is no more space left.

Figure 3 shows an example of the impact vector computation. At the top, the original positions of the nodes, $x^{node}_{original}$, are represented. At the bottom, the current positions of nodes, $x^{node}_{current}$, are depicted. In the middle, the shifted position of nodes at the right side of the cell being inserted is shown for a displacement of 4 sites.

To open a single space to the right, only one node needs to be shifted. Therefore, the shifting impact is one, as reported by $impact_r[1]$. If, when shifting by one site to make room for the new node, the node being shifted is approaching its original position, the cost is decreased by one. If, on the other hand, the node is getting farther from its

Algorithm 2: Impact Computation

1 $node$ = reference node;
2 $impact_r[0..w] = \{0, \cdots, 0\}$;
3 $k = 0$;
4 **while** *($node$ **and** $k < w$)* **do**
5 **if** *$node$ is blockage* **then**
6 break;
7 **end**
8 **if** *$node$ is whitespace* **then**
9 k += $\min\{width(node), w - k\}$
10 **else**
11 **for** $i = k + 1$, $d = 1$; $i \leq w$; i++, d++ **do**
12 $impact_r[i]$ += $weight^{node} \times (|x^{node}_{original} - (x^{node}_{current} + d)| - |x^{node}_{current} - x^{node}_{original}|)$
13 **end**
14 **end**
15 $node = next(node)$;
16 **end**
17 $overflow_r = w - k$

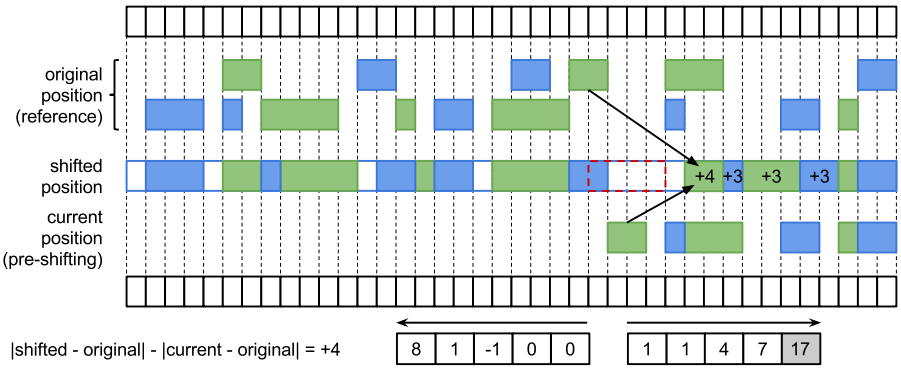

Figure 3. Impact vector computation for the right side.

original position, the cost is increased by one, and so on, that is why we see an impact of +4 when moving the green node pointed by arrows 4 sites to the right from its previous position. Each index in the cost vector for each side is computed by the summation of the costs of moving each of the cells in the way aside, as shown.

6.1. Optimum Shift

Once the cost vectors are computed, one needs to choose the optimum amount to shift left and right. This is performed by sweeping the two vectors in opposite directions, as presented in Algorithm 3. Note that, by traversing the vectors in opposite directions, the amount of open spaces always matches the new node width. The combination with the lowest cost of left and right shifts plus the movement of the cell being inserted itself is then selected. If the full legalization is being performed, the right cost vector does not count, as it only might be necessary to move already legalized cells to the left in order to make space for the new one, which is always assigned to the first rightmost available position.

Algorithm 3: Optimum Shift

```
1  max_l = w − overflow_l // max available sites to left
2  max_r = w − overflow_r // max available space to right
3  if max_l + max_r < w then
4  |  exit(overflow);
5  end
6  best_cost = ∞;
7  for i = max_l; i ≥ w; i- - do
8  |  cost = impact_l[i] + weight × |(x_original + w) − i| + impact_r[w − i]
9  |  if cost < best_cost then
10 |  |  best_cost = cost
11 |  |  shifts_left = i
12 |  end
13 end
14 displace left nodes by shifts_left to left;
15 displace right nodes by shifts_left − i to right;
```

6.2. Cache System

To allow fast look-up of the reference node, Jezz implements a cache system. Each row is divided into same-width regions, and, for each region, a pointer to a node inside that regions is stored. Jezz does not keep the cache always consistent to avoid unnecessary overhead. If the node is displaced horizontally, the cache pointer may point to a node outside of its respective region. For little displacement, that should be fine, as we still get a node close to the aimed region anyway. If the node was moved to another row, the pointer gets invalid, and Jezz looks at the pointer in neighboring regions. The cache pointer to a region is updated when a look-up at that region is performed.

Note, however, that, for a full legalization, when cells are inserted by x-position, the cache system does not need to be used, as it is easier to search nodes sequentially from the extremity of the linked list, since they are very likely to be the last/first one.

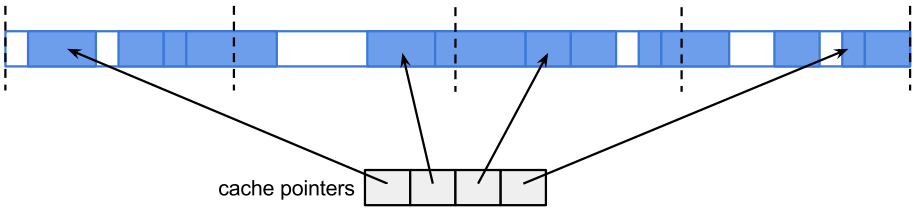

Figure 4. Cache System

7. Tetris Legalizer

Tetris [Hill 2002] algorithm starts by grouping the cells in a vector and then sorting them in ascending order by their x-coordinate. After that, for each cell, it calculates the cost of moving it to each of the possible rows in the circuit. The cost is obtained by getting the Manhattan distance between the original cell position and the position it would occupy within the row. The best cost is changed dynamically by iterating over all the possible rows to insert the cell in, so it receives the smallest cost value found from iterating through the rows. After a cell is placed in the best row, the row's leftmost X position is increased by the added cell's length.

8. Abacus Legalizer

Abacus [Spindler et al. 2008] algorithm starts by grouping the cells in a vector and sorting them just as well as Tetris algorithm, and, after that, it legalizes one cell at a time, keeping the cell as close as possible to the original position given by global placement, tentatively inserting the cell in certain rows until the best one is found, and then selecting a position for the cell horizontally. It also moves already legalized cells, which yields a lower overall displacement than Tetris, but using a quadratic function to calculate the new position a cell should occupy.

9. Experimental Results

For the experiment, each benchmark has been primarily randomly placed and then tested a hundred times for both legalizers. The tests were performed in a machine with 64 bit processor Intel Core i7, 3.4GHz, running Ubuntu version 14.04. Jezz was implemented using language C++. The benchmarks used were the ones available from ISPD 2002. Table 1 shows the number of cells each benchmark circuit has and a comparison between the legalizers Tetris and Jezz, after the experiment, in terms of elapsed time during the execution of the algorithm and the total displacement of the cells within the circuit for each different benchmark.

As we can notice from Table 1, the overall displacement of cells from their original position is diminished by almost 43% comparing to Tetris and it is almost 3% better than Abacus, which uses a quadratic function , so it performs a smooth transition between global and detailedplacement, as well, if it must shift cells, it prefers to shift the ones that will be closer to their original positions, and by using a linear function. Also, 2 shows that Jezz has a much longer execution time than Tetris and Abacus, but it does not take more than a second to execute, even for larger circuits.

Figure 5 shows that most of the cells from the benchmark ibm05 are displaced, at most, by 10 μm, so most cells are displaced by small distances with Jezz.

Table 1. maximum displacement for Jezz, Tetris and Abacus

Benchmark	Jezz	Tetris	Abacus	Jezz/Tetris	Jezz/Abacus
ibm01	95553	159663	102275	40.15%	6.57%
ibm02	159081	269146	160152	40.89%	0.67%
ibm03	215381	372928	224151	42.25%	3.91%
ibm04	185665	318344	196780	41.68%	5.65%
ibm05	182408	333974	183292	45.38%	0.48%
ibm06	210849	383065	220412	44.96%	4.34%
ibm07	336495	570545	351739	41.02%	4.33%
ibm08	337813	607089	336468	44.36%	-0.40%
ibm09	397210	679333	414455	41.53%	4.16%
ibm10	604523	1030000	608412	41.31%	0.64%
ibm11	578916	987000	602938	41.32%	3.98%
ibm12	632063	1050000	632967	39.80%	0.14%
ibm13	745599	1240000	763361	39.87%	2.33%
ibm14	1010000	1850000	1020000	45.41%	0.98%
ibm15	1020000	1860000	1050000	45.16%	2.86%
ibm16	1260000	2320000	1280000	45.69%	1.56%
ibm17	1400000	2450000	1410000	42.86%	0.71%
ibm18	1450000	2630000	1450000	44.87%	0.00%
			avg	42.69%	2.38%

Table 2. runtime for Jezz, Tetris and Abacus

Benchmark	#Cells	Jezz	Tetris	Abacus
ibm01	12506	0.038755	0.000905	0.004535
ibm02	19342	0.075944	0.001409	0.007149
ibm03	22853	0.100474	0.001705	0.008749
ibm04	27220	0.140916	0.003052	0.011423
ibm05	28146	0.13774	0.002067	0.011268
ibm06	32332	0.126951	0.002347	0.013185
ibm07	45639	0.200308	0.003432	0.021171
ibm08	51023	0.23508	0.003814	0.021967
ibm09	53110	0.197032	0.004259	0.029005
ibm10	68685	0.390358	0.005276	0.033493
ibm11	70152	0.361115	0.005777	0.035308
ibm12	70439	0.333709	0.005418	0.040669
ibm13	83709	0.390846	0.006478	0.041311
ibm14	147088	0.702293	0.01374	0.10258
ibm15	161187	0.631168	0.013228	0.114065
ibm16	182980	0.877742	0.016378	0.129681
ibm17	184752	0.923253	0.016558	0.12982
ibm18	210341	1.06586	0.019803	0.144765

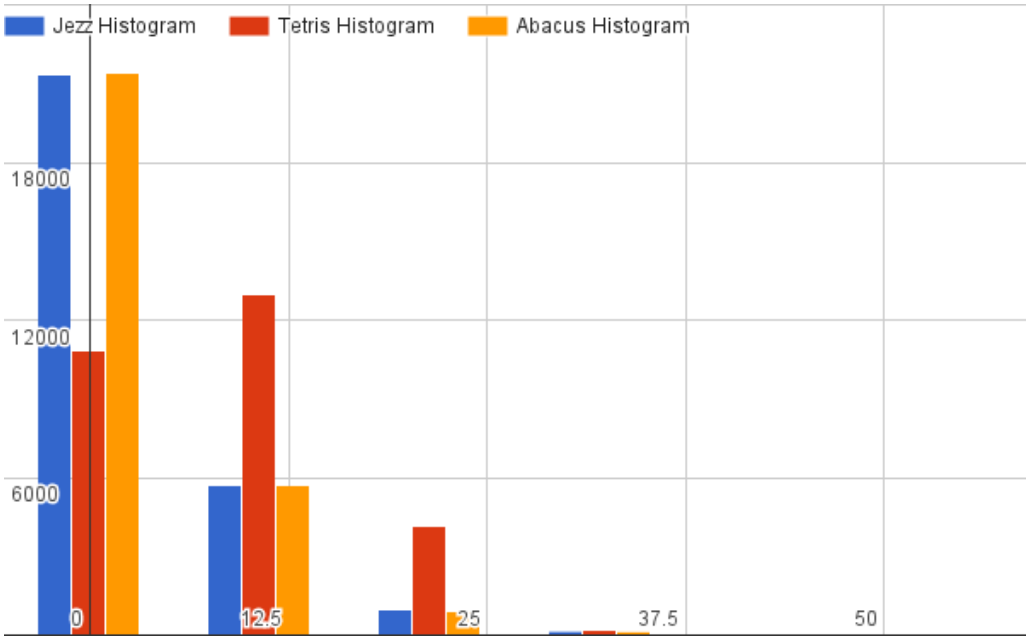

Figure 5. Histogram of Cell Displacement for $ibm01$

10. Complexity Analysis

The problem of finding a global optimum position for each cell is NP-complete and can become unfeasible, but, in this paper, we propose a method for legalizing cells trying to shift their positions as little as possible, reaching a good local minimum by calculating the overall displacement that must be made in order to make room for the new cell within a row. This can be performed with quadractic complexity for the worst case, but, on average, as we can notice in the graph shown in figure 6, the runtime increases almost linearly with number of cells.

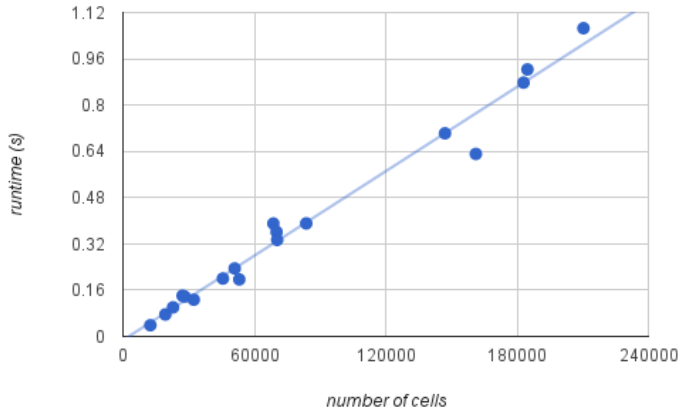

Figure 6. Jezz runtime increases linearly with the number of cells.

11. Conclusions

In this paper, we presented Jezz, an approach for legalizing cells within a circuit. For each cell, it divides the region for positioning in its left and right sides, and searches for a better place to insert it in both sides. Whitespaces are considered as a type of node, as well as obstacles, and the former are split when the cell is to be placed in the center of them. We also compared this approach with Tetris and Abacus in terms of runtime and total displacement, and Jezz showed itself to be almost 43% better than Tetris and 2.38% better than Abacus. Although being much slower than the other two in runtime, a legalization of more than 200k cells (benchmark ibm18 in table 2) runs in about a second, an acceptable amount of time, and the runtime increases almost linearly with the number of cells, as seen in graph 6. For future work, we plan on implementing a detailed placement tool using Jezz and reducing the linear search time in it, and making it handle mixed size cells as well as standard cells.

12. Publications

Previous versions of this work have been published in [Flach et al. 2015a] and [Puget et al. 2015], and a previous version of Jezz was part of the placement flow that achieved the first place in the ICCAD timing driven placement contest in 2014 [Flach et al. 2015b]. The current version is in SBCCI 2015, yet to appear.

References

Brenner, U. (2013). Bonnplace legalization: Minimizing movement by iterative augmentation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 32(8):1215–1227.

Cong, J. and 0004, M. X. (2008). A robust mixed-size legalization and detailed placement algorithm. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(8):1349–1362.

Flach, G., Puget, J., Monteiro, J., Foga¸ca, M., Johann, M., Butzen, P., and Reis, R. (2015a). Jezz: An incremental legalizer. In *Iberchip*.

Flach, G., Puget, J., Monteiro, J., Johann, M., and Reis, R. (2015b). Jezz: An effective legalization algorithm for minimum displacement.

Hill, D. (2002). Method and system for high speed detailed placement of cells within an integrated circuit design.

Lee, Y.-M., Wu, T.-Y., and Chiang, P.-Y. (2010). A hierarchical bin-based legalizer for standard-cell designs with minimal disturbance. In *ASP-DAC*, pages 568–573. IEEE.

Puget, J., Flach, G., Johann, M., and Reis, R. (2015). Jezz: An effective legalization algorithm for minimum displacement. SIM/EMICRO.

Spindler, P., Schlichtmann, U., and Johannes, F. M. (2008). Abacus: Fast legalization of standard cell circuits with minimal movement. In *Proceedings of the 2008 International Symposium on Physical Design*, ISPD '08, pages 47–53, New York, NY, USA. ACM.