Abacus: Fast Legalization of Standard Cell Circuits with Minimal Movement

Peter Spindler, Ulf Schlichtmann and Frank M. Johannes
Insitute for Electronic Design Automation, Technische Universitaet Muenchen
Munich, Germany

ABSTRACT

Standard cell circuits consist of millions of standard cells, which have to be aligned overlap-free to the rows of the chip. Placement of these circuits is done in consecutive steps. First, a global placement is obtained by roughly spreading the cells on the chip, while considering all relevant objectives like wirelength, and routability. After that, the global placement is legalized, i.e., the cell overlap is removed, and the cells are aligned to the rows. To preserve the result of global placement, cells should be moved as little as possible during legalization.

This paper presents "Abacus", which is a fast approach to legalize standard cell circuits with minimal movement. The approach is based on sorting the cells according to their position first, and legalizing the cells one at a time then. Legalizing one cell is done by moving the cell from row to row until the optimal place with the lowest movement is found. Whenever a cell is moved to a row, the cells already aligned to the row are placed by dynamic programming to minimize their total movement. Therefore, our approach Abacus moves already legalized cells during legalization. In contrast to this, Tetris [1], which uses a similar legalization technique, does not move already legalized cells. Consequently, the average movement is about 30% lower in Abacus than in Tetris. On the other hand, the CPU time of the whole placement process is increased by only 7% with our legalization approach. Applying Abacus to routability-driven placement results in 1% improvement in routed wirelength.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and Routing*

General Terms

Design, Algorithms

Keywords

Legalization, Standard Cell Circuits, Minimal Movement, Dynamic Programming

1. INTRODUCTION

Typical circuits today are standard cell circuits with millions of standard cells. Standard cells are for example inverters, nand, nors. These cells all have the same height but different widths. The chips of standard cell circuits are organized in rows, and the standard cells have to be aligned to these rows.

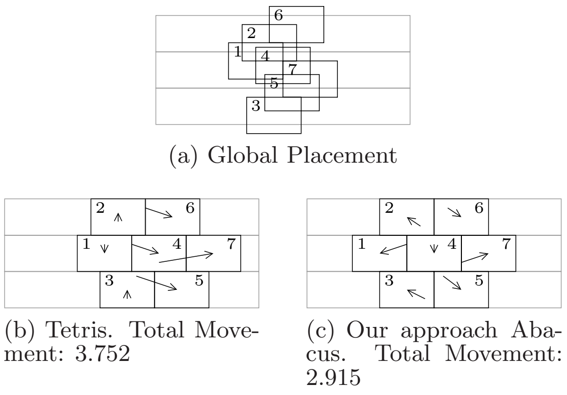
Due to the high number of standard cells per circuit, physical design of modern circuits is done in two main steps: placement of the cells, and routing of the nets, which connect the cells. Placement itself is also done in separate steps. First, the cells are roughly spread on the chip. This results in a global placement with few cell overlap, and the cells not aligned to the rows. Several approaches exist for global placement today [2] [3] [4] [5] [6] [7], just naming a few. The global placement is legalized then. There, legalization means to align the cells overlap-free to the rows. After legalization, detailed placement is used to improve the legal placement.

This paper focuses on the import step of legalizing a global placement. The global placement is obtained either by a global placement approach, or after changing a placed circuit (for example after gate sizing). Moreover, it is assumed that the global placement was optimized considering different objectives like total wirelength, or routability. Hence, during legalization, the placement is to be disturbed as little as possible, i.e., the movement of the cells is to be minimized.

1.1 Related Work

Various approaches exist for legalizing standard cell circuits. Domino [8] shreds cells in sub cells (all having the same height and width) and rows in places, and assigns sub cells to places by solving a min-cost-max-flow. The authors of [9] present a similar method to Domino, but assign sets of modules to row regions. Fractional Cut [10] is a two stage approach: first the cells are assigned to rows by dynamic programming, then the cells of each row are packed from left to right. The authors of [11] also present a two stage approach: first the cells are assigned to the rows by cell juggling, then the cells of each row are placed by finding a shortest path in a graph. Mongrel [12] uses a greedy heuristic to move cells from overflowed bins to under capacity bins in a ripple fashion based on total wirelength gain. Diffusion based placement migration is presented in [13] to remove cell overlap. In [14], computational geometry is used to spread the cells, and to align them to rows. NRG [15] uses simulated annealing for legalization.

Tetris [1] is a fast greedy heuristic, which is widely used (for example in [16] [3] [17]). The authors of [18] present a similar approach. In [17] [19], extensions to mixed size placements are described. Tetris sorts the cells first, and legalizes one cell at a time then. The legalization of one cell is done by moving the cell over the rows, and within a row by moving the cell over free sites. This movement is done until the nearest free site is found. Once a cell has been legalized, it will not be moved anymore. This results in a high total cell movement during legalization. This problem of Tetris is demonstrated in Figure 1(b). Here the global placement as shown in Figure 1(a) is legalized. The cells 1,2,3, which are legalized first, are moved small distances, the following cells 4,5,6,7 are moved long distances, resulting in a high total movement of all cells.



(a) Global Placement

(b) Tetris. Total Movement: 3.752

(c) Our approach Abacus. Total Movement: 2.915

Figure 1: Motivation for Abacus. Global placement (a) is given. (b) and (c) show legal placements. Arrows display the movement of the cells between global and legal placement.

1.2 Our Contribution

Our legalization approach "Abacus" is similar to Tetris in that the cells are sorted first, and are legalized one at a time then. However, during legalization, our approach moves already legalized cells, which yields a lower total movement of all cells (see Figure 1(c)). In detail, the cells already aligned to a row are moved such that their total movement is optimized. The optimum is found by solving a quadratic program, which can be transformed to a fast dynamic program. In addition, Abacus preserves the relative order of the cells, i.e., if two cells a and b are in the same row in the legal placement, and a is left of b, then a was left of b in the global placement.

1.3 Detailed Placement

After legalization, detailed placement approaches can be used to improve the legal placement. For example, the cells in each row can be placed optimal (while preserving their ordering) by a dynamic program to reduce wirelength [20] [21]. Another common technique is to use a sliding window to capture a small number of cells (about 10), and to apply different transformations to these cells [22] [19] [16] [23] [24]. The transformations are for example rotating, flipping or moving single cells, exchanging pair of cells, or changing the ordering of all cells in the window. It should be noted that this paper addresses legalization; detailed placement is not covered here.

1.4 Outline

The rest of paper is organized as follows: Section 2 presents our legalization approach; Section 3 describes how the cells in one row are placed during legalization; Section 4 explains our dynamic programming approach; Section 5 analyzes the worst-case computational complexity; Section 6 demonstrates various experimental results; Section 7 concludes the paper.

2. LEGALIZATION

Legalizing a standard cell circuit is to align the standard cells overlap-free to the rows of the chip. Legalization is one placement step. The input of legalization is a global placement, where the cells are roughly spread on the chip, the cells have some overlap, and the cells are not aligned to the rows. In this paper, it is assumed that different objectives like wirelength, length of the critical path (timing), routability, or the temperature profile are already optimized in the global placement. Consequently, to preserve the global placement as far as possible, the objective of legalization is to move the cells as little as possible.

Moreover, it is assumed that the rows are oriented horizontally, and it is assumed that the standard cells all have the same height. If the circuit has macros (non-standard cells), then it is assumed that the macros are already placed overlap free. In addition, it is assumed that the rows blocked by macros are sliced in new rows (subrows) such that all new rows are not blocked by macros anymore.

Algorithm 1: Our legalization approach "Abacus"

```
1  Sort cells according to x-position;
2  foreach cell $i$ do
3      $c_{best} \leftarrow \infty$;
4      foreach row $r$ do
5          Insert cell $i$ into row $r$;
6          PlaceRow $r$ (trial);
7          Determine cost $c$;
8          if $c < c_{best}$ then $c_{best} = c$, $r_{best} = r$;
9          Remove cell $i$ from row $r$;
10     end
11     Insert Cell $i$ to row $r_{best}$;
12     PlaceRow $r_{best}$ (final);
13 end
```

Algorithm 1 describes our legalization approach "Abacus". Similar to Tetris, the cells are sorted first according to their x-position (line 1). Then, the cells are legalized one a time (line 2-13). The legalization of one cell i is done by moving it over the rows (line 4-10). In each row, the cell is inserted according to its x-position in the global placement (line 5). Then, "PlaceRow" (line 6) places all cells of the row such that their total movement is minimal and they are not overlapping. PlaceRow is described in the next section. It is the major difference between our legalization approach and Tetris. After PlaceRow is called, the cost of the new position of cell i is determined (line 7), e.g., by the movement of cell i between its position in the global placement and its new position in the current row. At last, the cell i is removed from the current row (line 9).

After the cell i is moved over the rows, it is inserted to the best row (line 11). The best row is the row with the lowest cost (line 8). During the movement of the cell i over the rows (line 4-10), i.e., during the trial mode, the results of PlaceRow are treated as temporary positions, which means that the cells are not really moved to these positions. Hence, the best row needs to be placed again (line 12), and the results of PlaceRow are treated as the final legal positions. This means, the cells are actually placed to these positions.

Different issues about Abacus should be noted here. First, the sorting of the cells according to their x-position can be done either in increasing order, or in decreasing order. Both directions should be tested because the result of each direction can be different, and the best result should be used. Experiments showed that the difference in the total movement between both sort directions is about 0.5%.

Another issue is that each cell needs not to be moved over all rows of the chip (line 4-10). Rather, each cell is first moved to the nearest row (according to the global position). Then, it is moved above and below this row. For each row, a lower bound of the cost is computed by assuming that the cell is only moved vertically. If the lower bound exceeds the minimal cost of an already found legal position, then the movement in this direction (up or down over the rows) can be stopped. This method limits the movement to a few rows, and improves the CPU time of legalization drastically.

At last, it should be noted that the cells are inserted into the rows in order of their x-position in the global placement. Since the cells are processed according to their global x-position (line 1 and 2), inserting a cell into a row means either to append the cell as the last cell in the row (if sorted in increasing order), or as the first cell (if sorted in decreasing order).

3. PLACEROW

The core of our approach is to optimize the total (quadratic) movement of all cells within one row. This optimization is called PlaceRow, and it is used several times for each cell during legalization (see Algorithm 1). In the following, PlaceRow is described.

First, it is assumed that the row has N_r standard cells, indexed from 1 to N_r. Table 1 shows different properties of one cell i. Given is the position (of the lower left corner of the cell) in global placement $(x'(i), y'(i))$, the width $w(i)$, and the weight $e(i)$. The weight can be for example one, the area of the cell, or the number of pins of the cell. PlaceRow determines the legal x-position $x(i)$ of each cell. The legal y-position $y(i)$ is obtained beyond PlaceRow by moving the cell over the rows.

Property	**Explanation**
$x'(i), y'(i)$	Position in global placement
$x(i), y(i)$	Position in legal placement
$w(i)$	Width
$e(i)$	Weight (e.g., number of pins)

Table 1: Properties of cell i. Position refers to the lower left corner of the cell.

Moreover, it is assumed that all cells are sorted by their global x-position, i.e., $x'(i) \geq x'(i-1)$. Based on these definitions, PlaceRow is described by the following quadratic program:

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

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

The objective (1) describes the total, weighted, and squared movement of all cells between the global position $x'(i)$ and the legal position $x(i)$. By the way, the objective is convex since all weights $e(i)$ are positive. The constraint (2) assures that there is no overlap between the cells. In addition, the constraint preserves the ordering of the cells, i.e., cell a is placed left of cell b if a is left of b in the global placement.

The quadratic program (1) s.t. (2) can be solved with OOQP [25] for example. However, solving quadratic programs with "$\geq$" constraints is time consuming in general. If the same solution of the quadratic program (i.e., the same legal placement) is found by "$=$" constraints, then the quadratic program is solved quite fast by solving one linear equation. The situation that "$=$" constraints are sufficient is given if all cells of one row are abutting in the legal placement. There, two cells are "abutting" if there is no free space between them in the legal placement.

With only "$=$" constraints, (2) is transformed to:

$$x(i) = x(1) + \sum_{k=1}^{i-1} w(k) \quad i = 2, ..., N_r \tag{3}$$

Inserting (3) in (1) gives a quadratic function, only depending on $x(1)$. The minimum of this function is obtained by setting its derivative with respect to $x(1)$ to zero, which gives:

$$\underbrace{\sum_{i=1}^{N_r} e(i)}_{e_c} x(1) - \underbrace{\left[e(1)x'(1) + \sum_{i=2}^{N_r} e(i) \left[x'(i) - \sum_{k=1}^{i-1} w(k) \right] \right]}_{q_c} = 0 \tag{4}$$

Init	**Iteration** $(i = 1, 2, ..., N_r)$
$e_c = 0$	$e_c \leftarrow e_c + e(i)$
$q_c = 0$	$q_c \leftarrow q_c + e(i)\,[x'(i) - w_c]$
$w_c = 0$	$w_c \leftarrow w_c + w(i)$

Table 2: Iterative calculation.

Table 2 shows the iterative calculation of e_c, w_c, and q_c, which depends only on given properties of the cells: $x'(i)$, $w(i)$, and $e(i)$. Executing the iterations up to $i = N_r$, e_c is the total weight of all N_r cells, and w_c is the total width of all N_r cells. q_c is used in (4) and gives the optimal position $x(1)$ of cell $i = 1$:

$$x(1)\, e_c - q_c = 0 \quad \Leftrightarrow \quad x(1) = \frac{q_c}{e_c} \tag{5}$$

Using (3), the optimal positions $x(i)$ of the remaining cells $(i = 2, ..., N_r)$ in the row are determined. At this point, the quadratic program, and thus PlaceRow, is solved based on one linear equation (5) — assuming "$=$" constraints.

4. IMPLEMENTATION BY DYNAMIC PROGRAMMING

However, the “=” constraints are just allowed for abutting cells, and not in general for all cells in one row. Therefore, a method is necessary to detect clusters of cells, where all cells in the clusters are abutting, and the clusters themselves do not abut. The optimal position of a cluster is found then by solving (5) for this cluster. Here it should be noted that (5) is obtained by assuming that all cells $i = 1, ..., N_r$ in the row are in one cluster.

The clustering method, and solving PlaceRow by dynamic programming are shown in this section.

The properties of one cluster c are summarized in Table 3. $n_{first}(c)$ is the first cell in the cluster c, and $n_{last}(c)$ is the last cell in the cluster c. If just one cell is in the cluster, $n_{last}(c) = n_{first}(c)$; otherwise $n_{last}(c) = n_{first}(c) + N_c(c) - 1$ with $N_c(c)$ the number of cells in cluster c. $x_c(c)$ is the left x-position of cluster c, $e_c(c)$ represents the total weight of the cells in the cluster c, and $w_c(c)$ is the total width of the cluster c.

Property	Explanation
$n_{first}(c)$	First cell of cluster
$n_{last}(c)$	Last cell of cluster
$N_c(c)$	Number of cells in cluster, $N_c(c) = n_{last}(c) - n_{first}(c) + 1$
$x_c(c)$	Optimal position (lower left corner)
$e_c(c), w_c(c), q_c(c)$	Values similar to Table 2
$e_c(c)$	Total weight
$w_c(c)$	Total width
$q_c(c)/e_c(c)$	Optimal position

Table 3: Properties of cluster c

Algorithm 2: PlaceRow. Places all cells in one row optimal, i.e., with minimal total movement. Solves (1) s.t. (2).

```
   // Determine clusters and their optimal positions $x_c(c)$:
1  for $i = 1, ..., N_r$ do
2      $c \leftarrow$ Last cluster;
       // First cell or cell $i$ does not overlap with last cluster:
3      if $i = 1$ or $x_c(c) + w_c(c) \leq x'(i)$ then
4          Create new cluster $c$;
5          Init $e_c(c), w_c(c), q_c(c)$ to zero;
6          $x_c(c) \leftarrow x'(i)$;
7          $n_{first}(c) \leftarrow i$;
8          AddCell($c, i$);
9      else
10         AddCell($c, i$);
11         Collapse($c$);
12     end
13 end
   // Transform cluster positions $x_c(c)$ to cell positions $x(i)$:
14 $i \leftarrow 1$;
15 for all clusters $c$ do
16     $x = x_c(c)$;
17     for $i \leq n_{last}(c)$ do
18         $x(i) \leftarrow x$;
19         $x \leftarrow x + w(i)$;
20     end
21 end

22 Function AddCell($c, i$):
23 $n_{last}(c) \leftarrow i$;
24 $e_c(c) \leftarrow e_c(c) + e(i)$;
25 $q_c(c) \leftarrow q_c(c) + e(i) \cdot (x'(i) - w_c(c))$;
26 $w_c(c) \leftarrow w_c(c) + w(i)$;

27 Function AddCluster($c, c'$):
28 $n_{last}(c) \leftarrow n_{last}(c')$;
29 $e_c(c) \leftarrow e_c(c) + e_c(c')$;
30 $q_c(c) \leftarrow q_c(c) + q_c(c') - e_c(c') \cdot w_c(c)$;
31 $w_c(c) \leftarrow w_c(c) + w_c(c')$;

32 Function Collapse($c$):
   // Place cluster $c$:
33 $x_c(c) \leftarrow q_c(c)/e_c(c)$;
   // Limit position between $x_{min}$ and $x_{max} - w_c(c)$
34 if $x_c(c) < x_{min}$ then $x_c(c) = x_{min}$;
35 if $x_c(c) > x_{max} - w_c(c)$ then $x_c(c) = x_{max} - w_c(c)$;
   // Overlap between $c$ and its predecessor $c'$?:
36 $c' \leftarrow$ Predecessor of $c$;
37 if $c'$ exists and $x_c(c') + w_c(c') > x_c(c)$ then
       // Merge cluster $c$ to $c'$:
38     AddCluster($c', c$);
39     Remove cluster $c$;
40     Collapse($c'$);
41 end
```

Algorithm 2 shows the implementation of PlaceRow by dynamic programming. The algorithm starts in line 1-13 with iteratively clustering the cells, and determining the optimal position of each cluster. Here, the cells $i = 1, ..., N_r$ are processed in increasing order (line 1) according to their global x-position $x'(i)$, i.e., $x'(i) \geq x'(i-1)$. If cell i is the first cell, or if it does not overlap with the last cluster (line 3), then a new cluster is created containing the cell i (line 4-8). Otherwise, the cell i is added to the last cluster (line 10), and the last cluster is recursively collapsed with its predecessor cluster as long as the clusters are overlapping (line 11, and line 37-41 respectively).

During the clustering, the iterative calculation of $e_c(c)$, $w_c(c)$, and $q_c(c)$ is done in the functions “AddCell” (line 24-26) and “AddCluster” (line 29-31). The calculation is similar to Table 2. The optimal position $x_c(c)$ of cluster c is determined in line 33. This is similar to (5). In line 34 and 35, the position of a cluster is limited such that the left corner $x_c(c)$ is right of the starting position x_{min} of the row, and the right corner $x_c(c) + w_c(c)$ is left of the ending position x_{max} of the row.

In line 14-21 of Algorithm 2, the optimal positions $x(i)$ of all cells are determined based on the optimal positions $x_c(c)$ of the clusters to which the cells belong. After that, PlaceRow, and the quadratic program (1) s.t. (2) are solved.

The presented dynamic program is partly similar to the dynamic program published in [20] [21]. The latter dynamic program is used to place the cells in one row such that the total linear wirelength is minimized. Both dynamic programs (ours, and the one presented in [20] [21]) place the cells in one row without overlap, they do not change the ordering of the cells, and they optimize a convex objective. Thus, the optimality proof presented in [20] can be used to state that also our dynamic program is optimal.

The main difference to [20] [21] is that our dynamic program minimizes the quadratic cell movement. Because of the quadratic norm, the minimum is found quickly by solving one linear equation (5). The low CPU time is necessary because PlaceRow is called for each cell of the circuit several times (see line 6 and 12 in Algorithm 1). Minimizing the linear cell movement would result in a more complicated way in finding the minimum. In principle, the objective is a piecewise linear function then, and the minimum is found by visiting the bends of the function, which needs more time than just solving one linear equation. Another difference to [20] [21] is that our dynamic program rely on simple data structures (namely arrays); sophisticated data structures like red-black trees used in [20] are not necessary in our approach.

5. WORST-CASE COMPUTATIONAL COMPLEXITY

This section analyzes the worst-case computational complexity of PlaceRow (Algorithm 2), and of Abacus (Algorithm 1). N represents the total number of cells in a circuit, and N_r is the number of cells in one row.

The worst-case complexity of PlaceRow is given by the number of calls to the functions “AddCell” (line 22-26) and “AddCluster” (line 27-31). The functions themselves have constant computational complexity. AddCell is called once for each cell (line 8 and 10), which gives a linear complexity $O(N_r)$. During recursive collapsing, AddCluster is called everytime a cluster overlaps with its predecessor (line 37-41). With N_r cells in the row, there can be at most $N_r - 1$ clustering processes. Hence, the number of calls to AddCluster depends also linearly on the number of cells, and gives a complexity of $O(N_r)$ for PlaceRow. Another critical part for the complexity of PlaceRow is line 18-19. However, this part is executed only once per cell, which gives also a linear complexity for PlaceRow. In summary, PlaceRow has a linear worst-case complexity: $O(N_r)$.

Based on this complexity of PlaceRow, the worst-case computational complexity of Abacus (Algorithm 1) can be analyzed. With N cells in the circuit, the “foreach loop” in line 4-10 of Algorithm 1 is called N times. With R the number of rows, one “foreach loop” has R cycles. In each cycle, PlaceRow is called. With at most $\hat{N}_r$ cells in one row, the complexity of PlaceRow is limited by $O(\hat{N}_r)$. Since all of this is executed in a nested way, the worst case complexity of Abacus is $O(N\,R\,\hat{N}_r)$.

To obtain a complexity of Abacus, which just depends on N, approximations for R and $\hat{N}$ are necessary. Assuming that the standard cells are quadratic (same width and height), and the chip area is also quadratic, the number of rows is $R \approx \sqrt{N}$. The upper bound for the number of cells in one row is the same $\hat{N}_r \approx \sqrt{N}$. This gives the complexity of Abacus by $O(N^2)$.

6. EXPERIMENTAL RESULTS

In this section, different experimental results are presented. All of our experiments were executed on an AMD Opteron 248 machine with 8 GB RAM running at 2.2 GHz. Although PlaceRow targets the quadratic movement, the term “movement” in this section refers to the Euclidean movement. That means, the movement of one cell is the Euclidean distance between the position of the cell in the global placement, and the position of the cell in the legal placement. If not mentioned differently, the total movement is the sum of the movement of all cells in one circuit. The average movement is the total movement divided by the number of cells. Hence, total movement and average movement can be used interchangeable.

6.1 Movement Histogram

Figure 2 shows the histogram of the movement. The perfect histogram would be a peak with a relative frequency of one at a movement of zero, representing that no cells is moved. However, the cells in the global placement are overlapping, and are not aligned to the rows. Therefore, the cells are moved during legalization. Compared to Tetris, the histogram of the movement using our approach Abacus is better, the cells are moved less, and the peak is nearer to zero movement. The average movement is about 30% lower in Abacus than in Tetris.

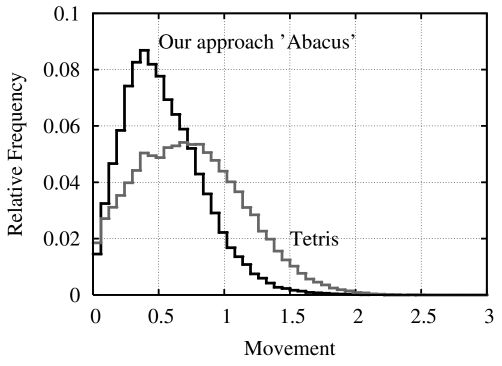


Figure 2: Movement histogram of Tetris and of our approach Abacus. Based on ibm12e of the IBM-PLACE 2.0 benchmark suite. Movement is normalized to the average dimension of the cells.

6.2 Average-Case Computational Complexity

Figure 3 displays the CPU times of legalizing various circuits with Abacus. N represents the number of standard cells per circuit. The results with $N < 10^6$ are based on the IBM-PLACE 2.0 benchmark suite [26], the other results are based on the ISPD 2005 contest benchmark suite [27], and on the ISPD 2006 contest benchmark suite [28]. With the almost linear average-case computational complexity of $\Theta(N^{1.19})$, Abacus can easily cope with future circuits having an increasing N. Moreover, the worst-case complexity of $O(N^2)$ shown in Section 5 is not reached by experiments.

6.3 Application in Routability-Driven Placement

This section presents experimental results in legalizing routability-driven global placements of the IBM-PLACE 2.0 benchmark suite. This suite consists of sixteen circuits (ibm03e/h-ibm06e/h do not exist). The global placements are obtained by routability-driven Kraftwerk [29], and as routability is optimized during global placement here, legalization has to be done with minimal movement in order to preserve the routability optimization. This is one application of Abacus. Table 4 shows the results of Abacus and of Tetris. After legalization (with Abacus or Tetris), detailed placement is applied by placing the cells in each row such that HPWL wirelength is minimized [20]. Wirelength

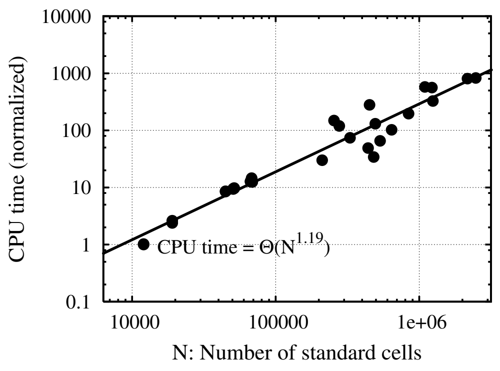


Figure 3: Average computational complexity of our approach Abacus.

is further reduced by flipping single cells, and by exchanging pairs of cells in a greedy fashion. The CPU time for the complete placement process is increased on average by about 6.6% if Abacus is used. Tetris increases the CPU time only by 0.5%. It should be noted here that Kraftwerk is a very fast placer, which means that an increase in CPU time of 6.6% with Abacus is not dramatic. Compared to Tetris, Abacus reduces the average movement by about 30%, representing that the global placement is better preserved in Abacus. Consequently, the routed wirelength, and the number of vias are decreased by about 1% if Abacus is used. Here, routing is done with Cadence WarpRoute 2.3.33, and includes final routing. The CPU time for routing is about the same in Tetris and in Abacus.

In [29], results of routability-driven Kraftwerk are published, using a legalization approach similar to Tetris. Along with this, a comparison to other state-of-the-art placers is published there. This comparison demonstrates that Kraftwerk offers excellent results for routability-driven placement. Using Abacus instead of Tetris, the results of Kraftwerk can be improved, as shown in Table 4.

7. CONCLUSION

This paper presents "Abacus", which is a placement legalization approach with minimal movement, applicable for standard cell circuits. Abacus is based on Tetris, but during each step of legalization, all cells of one row are placed optimally by minimizing their total movement. This reduces the total movement of all cells by about 30% in Abacus. The optimal placement of one row is implemented efficiently by dynamic programming, which gives an almost linear (average) computational complexity of Abacus. Furthermore, the CPU time of the whole placement process is increased by only 7%. Applied to routability-driven global placements, Abacus improves the routed wirelength by 1% compared to Tetris. Another application of Abacus would be to legalize timing-driven global placements.

8. REFERENCES

[1] Dwight Hill. Method and system for high speed detailed placement of cells within integrated circuit designs. U.S. Patent 6370673, April 2002.

[2] Jarrod A. Roy, David A. Papa, Saurabh N. Adya, Haward H. Chan, Aaron N. Ng, James F. Lu, and Igor L. Markov. Capo: Robust and scalable open-source min-cut floorplacer. In *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 224–226, 2005.

[3] Andrew B. Kahng and Qinke Wang. Implementation and extensibility of an analytic placer. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 24(05):734–747, May 2005.

[4] Tony Chan, Jason Cong, and Kento Sze. Multilevel generalized force-directed method for circuit placement. In *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 185–192, 2005.

[5] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. A high-quality mixed-size analytical placer considering preplaced blocks and density constraints. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 187–192, 2006.

[6] Ulrich Brenner and Markus Struzyna. Faster and better global placement by a new transportation algorithm. In *ACM/IEEE Design Automation Conference (DAC)*, pages 591–596, 2005.

[7] Peter Spindler and Frank M. Johannes. Fast and robust quadratic placement based on an accurate linear net model. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 179–186, 2006.

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

[9] Ulrich Brenner and Jens Vygen. Legalizing a placement with minimum total movement. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 23(12):1597–1613, December 2004.

[10] Ameya Agnihorti Mehmet Can Yildiz, Ateen Khatkhate, Ajita Mathur, Satoshi Ono, and Patrick H. Madden. Fractional cut: Improved recursive bisection placement. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 307–310, 2003.

[11] Andrew B. Kahng, Igor L. Markov, and Sherief Reda. On legalization of row-based placements. In *Great Lakes Symposium on VLSI (GLS-VLSI)*, pages 214–219, 2004.

[12] Sung-Woo Hur and John Lillis. Mongrel: Hybrid techniques for standard cell placement. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 165–170, 2000.

[13] Haoxing Ren, David Z. Pan, Charles J. Alpert, and Paul Villarrubia. Diffusion-based placement migration. In *ACM/IEEE Design Automation Conference (DAC)*, pages 515–520, 2005.

[14] Tao Luo, Haoxing Ren, Charles J. Alpert, and David Z. Pan. Computational geometry based placement migration. In *ACM/IEEE Design Automation Conference (DAC)*, pages 41–47, 2007.

[15] M. Sarrafzadeh and M. Wang. NRG: Global and detailed placement. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 532–537, 1997.

	Kraftwerk (global placement)									
	Our Approach "Abacus"					**Tetris**				
Circuit	CPU Leg	Move	CPU Route	rWL	# Vias	CPU Leg	Move	CPU Route	rWL	# Vias
ibm01e	0.79	0.913	297	0.678	118482	0.12	1.073	292	0.680	120198
ibm01h	0.88	1.175	354	0.673	119710	0.21	1.707	361	0.679	121424
ibm02e	2.43	0.721	364	1.840	253027	0.17	0.888	311	1.859	253170
ibm02h	2.06	0.854	387	1.977	265587	0.27	1.296	472	2.056	271696
ibm07e	6.44	0.542	551	3.559	469384	0.46	0.753	584	3.595	473695
ibm07h	8.66	1.000	591	3.601	483191	1.08	1.488	681	3.705	491398
ibm08e	8.75	0.569	844	3.993	559984	0.45	0.752	640	4.038	568458
ibm08h	9.57	0.579	715	3.926	567249	0.52	0.927	732	3.989	573271
ibm09e	9.80	0.618	582	2.877	484327	0.47	0.956	488	2.901	488415
ibm09h	8.73	0.620	493	2.890	487189	0.65	1.009	510	2.932	490594
ibm10e	11.23	0.543	871	5.660	759409	0.65	0.808	898	5.715	764847
ibm10h	11.35	0.542	890	5.692	761935	0.67	0.823	919	5.738	768437
ibm11e	12.52	0.536	670	4.319	629705	0.58	0.794	680	4.348	633766
ibm11h	13.07	0.554	650	4.281	629790	0.73	0.879	723	4.323	633421
ibm12e	11.16	0.535	1371	8.344	923900	0.64	0.748	1211	8.409	930654
ibm12h	11.36	0.541	1516	8.351	941797	0.66	0.794	1371	8.384	941651
Average	**6.6%$^+$**	**1.000**	**1.00**	**1.000**	**1.000**	**0.5%$^+$**	**1.456**	**0.995**	**1.012**	**1.010**

Table 4: Results in the IBM-PLACE 2.0 benchmark suite. "CPU Leg" is the CPU time of legalization. $^+$means the ratio between the CPU time of legalization and the CPU time of the complete placement process. "Move" is the average movement of the cells between global and legal placement, normalized to the average cell dimension of each circuit. "CPU Route" is the CPU time of routing. All CPU times are in seconds. "rWL" is the routed wirelength in meters after final routing.

[16] Chen Li, Min Xie, Cheng-Kok Koh, Jason Cong, and Patrick H. Madden. Routability-driven placement and white space allocation. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 26(5):858–871, May 2007.

[17] Ateen Khatkhate, Chen Li, Ameya R. Agnihotri, Mehmet C. Yildiz, Satoshi Ono, Cheng-Kok Koh, and Patrick H. Madden. Recursive bisection based mixed block placement. In *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 84–89, 2004.

[18] Chen Li and Cheng-Kok Koh. On improving recursive bipartitioning-based placement. Technical Report TR-ECE 03-14, Purdue University, December 2003.

[19] Jason Cong and Min Xie. A robust detailed placement for mixed-size ic design. In *Asia and South Pacific Design Automation Conference*, pages 188–194, 2006.

[20] Andrew B. Kahng, Paul Tucker, and Alex Zelikovsky. Optimization of linear placements for wirelength minimization with free sites. In *Asia and South Pacific Design Automation Conference*, pages 241–244, 1999.

[21] Ulrich Brenner and Jens Vygen. Faster optimal single-row placement with fixed ordering. In *Design, Automation and Test in Europe (DATE)*, pages 117–121, 2000.

[22] Andrew E. Caldwell, Andrew B. Kahng, and Igor L. Markov. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 19(11):1304–1313, November 2000.

[23] Min Pan, Natarajan Viswanathan, and Chris Chu. An efficient and effective detailed placement algorithm. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 48–55, 2005.

[24] Haoxing Ren, David Z. Pan, Charles J. Alpert, Gi-Joon Nam, and Paul Villarrubia. Hippocrates: First-do-no-harm detailed placement. In *Asia and South Pacific Design Automation Conference*, pages 141–146, 2007.

[25] Michael Gertz and Stephen Wright. Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29(1):58–81, March 2003.

[26] Xiaojian Yang, Bo-Kyung Choi, and Majid Sarrafzadeh. Routability-driven white space allocation for fixed-die standard-cell placement. In *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 42–49, 2002.

[27] Gi-Joon Nam, Charles J. Alpert, Paul Villarrubia, Bruce Winter, and Mehmet Yildiz. The ISPD2005 placement contest and benchmark suite. In *ACM/SIGDA International Symposium on Physical Design (ISPD)*, pages 216–219, May 2005.

[28] ISPD 2006 placement contest. `http://www.sigda.org/ispd2006/contest.html`, March 2006.

[29] Peter Spindler and Frank M. Johannes. Fast and accurate routing demand estimation for efficient routability-driven placement. In *Design, Automation and Test in Europe (DATE)*, pages 1226–1231, 2007.