Deterministic VLSI Block Placement Algorithm Using Less Flexibility First Principle

DONG SheQin (董社勤)\(^1\), HONG XianLong (洪先龙)\(^1\), WU YuLiang (吴有亮)\(^2\) and GU Jun (顾 钧)\(^3\)

\(^1\)Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P.R. China
\(^2\)Department of Computer Science and Engineering, The Chinese University of Hong Kong

Hong Kong, P.R. China
\(^3\)Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, P.R. China

E-mail: dongsq@tsinghua.edu.cn; ylw@csee.cuhk.edu.hk; gu@cu.ust.hk

Received May 18, 2001; revised April 29, 2002.

Abstract In this paper, a simple while effective deterministic algorithm for solving the VLSI block placement problem is proposed considering the packing area and interconnect wiring simultaneously. The algorithm is based on a principle inspired by observations of ancient professionals in solving their similar problems. Using the so-called Less Flexibility First principle, it is tried to pack blocks with the least packing flexibility on its shape and interconnect requirement to the empty space with the least packing flexibility in a greedy manner. Experimental results demonstrate that the algorithm, though simple, is quite effective in solving the problem. The same philosophy could also be used in designing efficient heuristics for other hard problems, such as placement with preplaced modules, placement with L/T shape modules, etc.

Keywords VLSI floorplanning, placement, NP-hard, VLSI physical design

1 Introduction

Floorplanning or module placement is the problem of placing a set of circuit modules on a chip to minimize the total area and interconnect wiring cost. Since it is the first stage of the VLSI physical design process, the quality of this stage is critical for the successive design stages. In practice, manual design is still quite prevailing on floorplans of full custom design, printed circuits boards, and analog circuits. For large ASIC designs, in order to expedite the process of floorplanning and placement, a lot of schemes have been proposed to automate this process\(^{[1,2]}\).

There are two kinds of floorplans: slicing and non-slicing. A slicing structure is obtained by recursive applications of "rectangle slicing" operations. Selections of horizontal or vertical slicing as well as the corresponding positions produce a wide variety of structures\(^{[3]}\). Slicing floorplan schemes possess the advantages of a faster running time, due to the reduction of search space. Meanwhile, many floorplans are non-slicing because of their larger flexibility in searching space. A non-slicing approach that employs a branch-and-bound strategy to search for an optimum solution in the whole solution space was presented in \(^{[4]}\). However, based on this approach, the maximum number of blocks that can be practically placed within a reasonable CPU time is only six.

The sequence pair representation for handling a non-slicing floorplan has been proposed in \(^{[5]}\). In the representation, two sets of permutations are used to reflect the geometric relation of the placed blocks. The \(p\)-admissible solution space is \((n!)^2\cdot 8^n\), where \(n\) is the total number of blocks. Clearly, the space is so large that there is no guarantee of finding an optimum solution in a reasonable amount of computation time. A bounded-slicing grid structure (BSG) to cope with non-slicing floorplan was proposed in \(^{[6]}\), where an \(n\) by \(n\) grid plane is used for the placement of \(n\) blocks. However, the representation itself has too much redundancy. In \(^{[7]}\), a 2D branch-and-bound cluster refinement algorithm of block placement for optimizing area and interconnection simultaneously was proposed. Given \(k\) as the cluster size, the run time complexity for each iteration in the algorithm is \(O(n^{2+k/2})\). It is clearly too CPU intensive if a larger cluster size is chosen.

It is obvious that the problem complexity increases a lot from slicing floorplan to non-slicing floorplan. Therefore some efficient representations of the non-slicing floorplans are desired. A root-
edly ordered tree, \textit{O}-tree, admissible placement representation for a non-slicing floorplan, where the blocks are compacted to the left and to the bottom edges, was proposed in [8]. The run time for transforming an \textit{O}-tree to its representing placement is linear to the number of blocks. A deterministic algorithm has been derived by perturbing \textit{O}-tree in sequence, but only external nodes can be the possible inserting positions. Recently, a more effective non-slicing floorplan topological representation is proposed. Though its combination space is slightly greater than \textit{O}-tree, it has the same computation complexity. As the problem of placement for hard blocks is an \textit{NP}-hard problem, to obtain good results, applying of heuristics would be needed.

In this paper, we will propose a quite different class of efficient heuristics, the so-called quasi-human based heuristics, based on some simple philosophy obtained from human’s accumulated experience on solving similar problems in everyday life. For example, our block placement problem is just a special case of the (irregular) polygon-shape stone packing problems facing our professional masons in their everyday work. Based on their long time (generations) accumulated experience, they happened to conclude that a simple principle, \textit{Less Flexibility First} (LFF) principle, can work well in solving their problems. Basically, the principle simply suggests using the more restricted resource and fulfilling the more restrictive requests with high priorities and putting off those less restrictive action requests till later steps. The principle was known in ancient Chinese society for generations. In this paper, we develop a set of quantitative flexibility for VLSI modules to be placed on the chip to reflect better our combined objectives of improving both the area and interconnection simultaneously for the VLSI block placement problems. Our experimental results show that this “less flexibility first” heuristic, although deterministic in nature, is quite efficient and effective.

The rest of the paper is organized as follows. In Section 2, the less flexibility first principle and definitions are developed. Section 3 describes the algorithm and its implementation. Section 4 presents our experimental results for MCNC benchmarks. We give our conclusions in Section 5.

2 Less Flexibility First Principle

For thousands of years, Chinese masons have been using the following rule-of-thumb in their daily stone packing work: “Golden are the corners; silverly are the sides; and strawy are the hollows”. Bizarre it might sound at the first look, but it simply states a priority order in selecting empty space inside the working area (e.g., bounding floor, wall, etc.) to be filled by packed stones. This filling order (corners, sides, then the rest) is observed “best” in maintaining the “packability” of the left-over working space. It tends to retain the working area “less fragmented”, “containing less holes” and “in the original form”. The principle is honored that when a new packing step is taken, the deformation of the working space should be kept as minimal as possible. There are three kinds of empty space a rectangle can choose to pack in: corners, space sides, and the rest, which are the “hollow” areas. Intuitively speaking, the deformation of the working space caused by corner packing is minimal, then the deformation caused by side packing, then that by space packing. The latter two cases should have more chances to cause fragmentation of the working space and the space packing would create more irregular space or “holes” in the work area. To pack the blocks tighter, one should use up all space in the work area by the blocks. Basically, to obtain final better results, we should try to use up the more restricted resource and fulfill the requests with more restricted requirements earlier so as to leave the more “flexible” resources or requests for the harder later stages of the processing. This schedule sequence of packing tasks can be named as the \textit{Less Flexibility First} (LFF) principle.

The major objectives of VLSI block placement are chip area minimization and interconnection wire length minimization. It is clear that the VLSI block placement is a problem more complicated than the pure rectangle packing. Based on the above LFF principle, we develop the following heuristic for our VLSI block placement problem.

2.1 Problem Definition

Given a set of rectangular blocks \( B = \{ B_1, B_2, \ldots, B_n \} \), a set of nets specifying the interconnections between pins of blocks, and a set of pads (external pins), a placement \( P = \{ (x_i, y_i)|1 \leq i \leq n \} \) is a coordinates assignment of the lower left corners of the \( n \) rectangular blocks such that there is no overlapping between blocks. Each \( B_i \) is represented by a tuple \((h_i, w_i)\), where \( h_i \) and \( w_i \) are the height and the width of block \( B_i \), respectively. The objective of the placement is to find an assignment with the chip area and interconnection wire length minimized while satisfying some given constraints, if any. The area of the smallest rectangle
that encloses the placement is called the chip area or working area of the placement.

2.2 Flexibility of Placement

2.2.1 Flexibility of Empty Space

For a rectangle to be packed, there are three kinds of packing space in the working area: corner space, side space, and central hollow part of the working area. Each of the three kinds of space has a distinct flexibility. The flexibility of a packing space is defined as its degree of freedom allowed for packing adjustment. In a working area, if the location of a block side is not fixed or adjacent to other block sides, then the block should have a flexibility to move towards the direction perpendicular to that side. For example, in Fig 1(a), the corner packed block has its two sides bounded by the boundary of the work area, so it cannot move horizontally to the right nor downward vertically; but it can move to the left, up, up-right directions. Therefore, we consider its flexibility degree as 3. Similarly, we define the flexibility of the other two cases as shown in Figs 1(b) and 1(c).

![Fig 1. Flexibility of empty space. (a) Flexibility of corner packing. (b) Flexibility of side packing. (c) Flexibility of hollow space packing.](image)

**Definition 1 (Flexibility of Empty Space: \( P_e^f \)).** The flexibility of a corner packing space is \( P_e^f = 3/8 \). The flexibility of a side packing space is \( P_s^f = 5/8 \). The flexibility of a hollow packing space is \( P_h^f = 8/8 = 1 \). It is clear that

\[
P_e^f < P_s^f < P_h^f
\]

(1)

Based on the Less Flexibility First principle, a sound packing ordering might want to arrange a packing space with lower flexibility to be packed (consumed) earlier, and hold the space with higher packing flexibility to accommodate tougher packing requirements that usually appear in later packing stages.

2.2.2 Flexibility of a Rectangle to Be Packed

Similarly, the rectangles to be packed also have different degrees of flexibility depending on their shapes and sizes. It is generally agreed that a rectangle with large size or with larger long-side/short-side ratio tend to possess a less packing flexibility. For example, in Fig. 2, block A clearly has a larger room of choices to pack in due to its smaller size and more regular shape, thus blocks B and C have less flexibility than block A.

![Fig 2. Flexibility of rectangle to be packed: A large size of or longer rectangle has less flexibility.](image)

**Definition 2 (Flexibility of a Rectangle: \( R_i^f \)).** Given \( A_i \) as the area of rectangle \( B_i, w_i \) its width and \( h_i \) its height, \( P_w \) the area of the working space, \( W \) the width of the working area and \( H \) its height, the flexibility of a rectangle is defined as:

\[
R_i^f = r_1 * \left(1 - \frac{A_i}{P_w}\right) + r_2 * \left(1 - \frac{\max(w_i, h_i)}{\max(W, H)}\right),
\]

(2)

where \( r_1 + r_2 = 1 \).

By the Less Flexibility First principle, a rectangle with less \( R_i^f \) should be packed first, i.e., a rectangle of larger size or "slimmer" shape will have a higher priority of being packed earlier.

2.2.3 Interconnect Flexibility Between Two Blocks

Interconnect wiring length optimization among VLSI blocks is also very important for VLSI block placement. As the exact wiring length is still unknown at the placement stage, we may adopt the conventional half perimeter bounding box model or other simple model for length estimation of a net. Any two VLSI building blocks may have interconnections between them. Besides, in a placement, the larger number of nets between two blocks also reflects the flexibility of their being bundled together. For example, in Fig 3, the number of nets between blocks A and C is larger than that between blocks A and B; therefore, block C will have a higher priority to be packed in the place \( P_x \) which is closer to the packed block A.
Definition 3 (Interconnect Flexibility Between Two Blocks: $C_{ij}^f$). Given $W_{ij}$ as the number of nets between block $B_i$ and block $B_j$, $W_{net}$ the number of all nets of the placement, the flexibility between the two blocks is:

$$C_{ij}^f = 1 - W_{ij} / W_{net}$$

According to our Less Flexibility First principle, a packing step involving the two blocks $B_i$ and $B_j$ that has a less $C_{ij}^f$ will be entertained first. Actually, a packing step always involves more than two blocks that will be the neighbors of the block to be packed. In Fig 3, when block B is packed in $P_L$, it will have three neighbors. In this case, it is reasonable to use interconnect flexibility among these blocks rather than interconnect flexibility between two blocks to measure the interconnect flexibility of the block to be packed in a corner position. So we use $C_{net} = C_{i1}^f + C_{i2}^f + \cdots + C_{im}^f + (1 - m)$ instead of $C_{ij}^f$ to measure the interconnect flexibility of a block to be packed which will have $m$ neighbors.

3 Algorithm and Its Implementation

3.1 Algorithm

A deterministic algorithm, named LFF, is designed based on our proposed Less Flexibility First (LFF) principle. The advantage of a deterministic algorithm is its higher speed and perhaps easier implementation.

Here we give the terminology for our algorithm description:

- $B_{BS}$: Block set to be packed;
- $U_{BS}$: Unpacked block set indexed by $i$;
- $P_{BS}$: Packed block set indexed by $j$;
- Ratio: Required ratio of length and width of the placement;
- $P_L$: Current placement configuration;
- $Area_N$: Sum of net area of all blocks in $B_{BS}$;
- $Pa$: The area of the placement.

$$\text{Alpha} = Area_N / Pa, \quad F_i$$ is the flexibility of block $i$.

$\lambda_1, \lambda_2, \lambda_3$ are the weights for the three flexibilities.

Then, we can get the width and length of $P_L$. Because $Area_N$ is already known, Ratio and Alpha can be given by the user.

Deterministic Algorithm, LFF

**Input**: array of blocks with width, height, pin positions, I/O pad positions, and netlist
**Output**: blocks with position and orientation

$U_{BS} = B_{BS}$

Update the placement with an estimated blank rectangle with required ratio of length and width;

while $U_{BS}$ is not empty and there is space for packing in $P_L$;

for each block $i$ in $U_{BS}$ under the current $P_L$;

- calculate:
  $$F_i = \lambda_1 * P_i + \lambda_2 * R_i + \lambda_3 * C_{net}$$

end for

Update $P_L$ with the block of the smallest $F_i$ packed in it;

- Add the block of the smallest $F_i$ to $P_{BS}$;
- Remove the block of the smallest $F_i$ from $U_{BS}$.

end while

Optimize the total wire length of the placement using half perimeter mode by vertical or horizontal flipping of each block in the final $P_L$.

3.2 Implementation

As discussed in the last section, it is reasonable that packed rectangle must occupy one empty corner of the current packing configuration. Therefore, there are many candidate corners for a packing step and one corner has two packing orientations for a rectangle. For example, in Fig 4 if there are totally 7 corners in the configuration, and a block B can be packed at any one of them with two possible orientations. As a result, there are 14 packing choices for block B. In terms of the flexibility definitions discussed above, we define one choice of this kind as a Candidate Corner Packing Step (CCPS). A CCPS can be represented by a six-tuple as

$$\langle \text{block_id}, \text{length}, \text{width}, \text{orientation}, x, y \rangle$$

The block_id is block name, the length and width are values of the longer and shorter sides of the packed block, orientation indicates if the rectangle is placed with its longer side laid horizontally or vertically. $(x, y)$ is the left-lower corner coordinate of the suggested location.

Firstly, based on the current packing configuration, a list of all candidate CCPSs for all unpacked blocks is generated. The candidate CCPS list is then sorted in a lexicographical order, where an
earlier CCPs possesses a higher priority. The sorting of the list is according to the definition of the flexibility of a rectangle in the last section.

![Diagram](Figure 4. Candidate packing corners for a block B)

In terms of selection among qualified candidate CCPs, we combine all the strategies shown above. For each of the rectangles to be packed, a set of its feasible CCPs, based on current packing configuration, is obtained. Then the CCPs are gathered as a list and sorted as described above. For each of these candidate CCPs, a “Fitness Value (FV)” will be calculated as follows. First, the candidate CCP is pseudo-packed into the current working space. The term “pseudo” means that it is just a test and the packing process can be reverted. The other leftover rectangles are then pseudo-packed with a greedy strategy following the list order until no rectangle can be packed. The strategy is “greedy” in the sense that it tries to pack the rectangles at the front of the list first and to pack as many rectangles as possible following the list order. The $FV_i$ of this candidate CCP $i$ is then calculated as:

$$FV_i = \mu_1 \times A_p + \mu_2 \times (1/C_{net}^i)$$

where the item $A_p$ is the total area of all rectangles packed, $C_{net}^i$ is the interconnect flexibility among the block $i$ to be packed and the already packed blocks in one corner. $\mu_1$ and $\mu_2$ are the weights. The candidate CCPs with the highest fitness value will be selected as the next packing candidate. The process repeats until all rectangles are really packed or the list becomes null while there are still unpacked rectangles. The while loop of the deterministic algorithm LFF described in last section can be rewritten as follows:

```
while $U_{BS}$ is not empty and there is space for packing in $P_L$
    for each block $i$ in $U_{BS}$
        Based on the current $P_L$, find all possible CCPs and form a list
    end for
    Sort all of the CCPs in the list in lexicographical order
    for each candidate CCP $i$ in the list
        Pseudo-pack this CCP in $P_L$
        Pseudo-pack all the remaining rectangles in the current CCPs list with a greedy approach until no more CCPs can be packed
        Calculate $FV_i = \mu_1 \times A_p + \mu_2 \times (1/C_{net}^i)$
        (Note: Before the test calculation for the next candidate CCPs is tried, the previous pseudo-packed CCPs will be restored.)
    end for
    Pick the candidate CCP with the highest $FV_i$ and update $P_L$ with really packing the corresponding block according to the CCPs with highest $FV_i$
    Add the block of the highest $FV_i$ to $P_{BS}$
    Remove the block of the highest $FV_i$ from $U_{BS}$
end while
```

3.3 More Efficient Algorithm: S-LFF

To further expedite the execution efficiency, the Less Flexibility First principle needs to be followed in a stricter manner. In each packing step, only the rectangle with the fewest CCPs will be packed. According to the definition of CCPs, we can regard it as another flexibility of a VLSI block. By “the Less Flexibility First principle”, a VLSI block with fewest CCPs will be packed first, so we can derive a stricter LFF algorithm: S-LFF. S-LFF can be derived as follows. In the rewritten while loop of the deterministic algorithm LFF, we replace the step “Pick the candidate CCP with the highest $FV_i$ and update $P_L$ with really packing the corresponding block according to its CCPs with highest $FV_i$”, with the step: “Pick the candidate block with the fewest CCPs, among these fewest CCPs, then choose the CCPs with the highest $FV_i$ and update $P_L$ with really packing the corresponding block according to its CCPs with highest $FV_i$.”

3.4 Complexity of the Algorithm

A K-D tree data structure is used in our implementation for manipulating the packed rectangles. It is actually a multidimensional binary tree which can be used for supporting area operations on 2-space. It provides fast $O(\log n)$ region searching operations which keep the time for insertion and deletion small.

A packing step will consume one or more corners and generate some new corners. After the first packing step, the corners of the current packing configuration $P_L$ will be 5, and one corner packing step will create at least one new corner in most cases. So at the $i$-th step we may have $(4 + i) \times 2 \times (n - i)$ CCPs, where $n$ is the total
number of rectangles to be packed. When taking \([n/2]\) (an integer not smaller than \(n/2\)) packing steps, the number of CCPs will be \((4 + n/2) \times n\). Therefore, for each object to be packed, the length of the list of CCPs generated will be bounded by \(O(n^2)\). For each entry in the list, a pseudo-packing for each remaining rectangle will be done. As a result, the complexity of one iteration will be \(O(n^2 \times n \times \log n)\). The process is repeated once for each rectangle packed, so the worst case complexity will be \(O(n^5 \times \log n)\). Similarly, the worst case complexity of the stricter manner packing process can be found as \(O(n^4 \times \log n)\). Actually, the average time complexity will be smaller due to the shortening of CCPs list after each successful pseudo-pack.

4 Experimental Results

We implemented the deterministic algorithm using C programming language and ran it on a Sun Spark II workstation. To examine the efficiency of the proposed algorithm, we apply our algorithm to the MCNC (Microelectronics Center of North Carolina University) benchmark circuits. This set of benchmarks was developed specifically for building block layout (BBL) design and has been widely used for floorplanning and placement algorithm testing. Table 1 describes the parameters of the five MCNC benchmark circuits.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Number of modules</th>
<th>1/0 pads</th>
<th>nets</th>
<th>pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apte</td>
<td>9</td>
<td>73</td>
<td>97</td>
<td>214</td>
</tr>
<tr>
<td>Xerox</td>
<td>10</td>
<td>107</td>
<td>208</td>
<td>606</td>
</tr>
<tr>
<td>Hp</td>
<td>11</td>
<td>43</td>
<td>83</td>
<td>264</td>
</tr>
<tr>
<td>Ani33</td>
<td>33</td>
<td>42</td>
<td>123</td>
<td>480</td>
</tr>
<tr>
<td>Ani49</td>
<td>49</td>
<td>24</td>
<td>408</td>
<td>931</td>
</tr>
</tbody>
</table>

In order to find the minimum bounding box sizes for successful solutions, we continued our experiments with the size of bounding box increased gradually. The sides of the box were increased one unit a time and the experiments were repeated until a successful solution was obtained. At present, in the combined optimization objective of chip area and total wire length, it is reasonable that the chip area is primary. Total wire length is just a rough estimation of the interconnect performance. A short total wire length does not mean a reasonable critical net wire length. If we could ensure the performance of critical net, we may put less effort on the total wire length minimization. One of our next studies on LFF algorithm is performance driven placement.

Our experimental results, using the algorithm which is not implemented in the stricter manner (block with the fewest CCPs will be packed first), are presented with a comparison with J. Xu's Cluster Refinement (CR) algorithm[5,6] in Table 2. Obviously, our algorithm LFF achieves results with better chip area for all cases while their wire lengths are comparable. Note that the CPU time is much less than that of CR algorithm.

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Area usage (%)</th>
<th>Wire length (mm)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apte</td>
<td>LFF 97.60</td>
<td>233.84</td>
<td>0.22</td>
</tr>
<tr>
<td>Xerox</td>
<td>LFF 96.20</td>
<td>259.64</td>
<td>0.63</td>
</tr>
<tr>
<td>Hp</td>
<td>LFF 97.00</td>
<td>251.38</td>
<td>0.73</td>
</tr>
<tr>
<td>Ani33</td>
<td>LFF 96.00</td>
<td>63.06</td>
<td>59.18</td>
</tr>
<tr>
<td>Ani49</td>
<td>LFF 97.00</td>
<td>1,150.59</td>
<td>487.63</td>
</tr>
</tbody>
</table>

Table 3. Comparison of S-LFF with O-Tree (Ultra-60) after One Run of Both Algorithms

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Usage (%)</th>
<th>Wire length</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ani33</td>
<td>S-LFF 95.0/1.217</td>
<td>63.5</td>
<td>2.95</td>
</tr>
<tr>
<td>O-tree</td>
<td>86.3/1.34</td>
<td>50.9</td>
<td>21.3</td>
</tr>
<tr>
<td>S-LFF</td>
<td>96.0/36.92</td>
<td>1,166.84</td>
<td>11.83</td>
</tr>
<tr>
<td>Ami49</td>
<td>O-tree 77.9/45.50</td>
<td>673</td>
<td>117</td>
</tr>
</tbody>
</table>

Table 4. Minimum/Average Distribution with Different Weights

<table>
<thead>
<tr>
<th>Circuits</th>
<th>area</th>
<th>wire</th>
<th>area</th>
<th>wire</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFF</td>
<td>1.20/1.12</td>
<td>63.1/64.6</td>
<td>1.20/1.12</td>
<td>64.5/67.0</td>
</tr>
<tr>
<td>Ami33</td>
<td>O-tree 1.26/1.34</td>
<td>51.6/59.8</td>
<td>1.25/1.32</td>
<td>61.1/67.4</td>
</tr>
<tr>
<td>C-LFF</td>
<td>1.18/1.205</td>
<td>51.27/55.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFF</td>
<td>36.5/37.3</td>
<td>894/1,034</td>
<td>36.5/37.3</td>
<td>1,319/1,396</td>
</tr>
<tr>
<td>Ami49</td>
<td>O-tree 39.1/42.0</td>
<td>671/777</td>
<td>37.6/39.9</td>
<td>819/1,375</td>
</tr>
<tr>
<td>C-LFF</td>
<td>37.8/37.9</td>
<td>940.9/989.96</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
sults are presented in Table 3 with a comparison with the algorithm based on $O$-tree\cite{6} after one run of the two algorithms under the original block sequence in MCNC benchmarks. Note that the algorithm based on $O$-tree is also a deterministic algorithm.

In Table 3, we achieve reasonable results in a much less time than $O$-tree and the experiment proves that our algorithm is not sensitive to the initial block sequence.

In Table 4, we compare the results of the normal LFF algorithm (not the stricter one) with $O$-tree-based algorithm\cite{6} in a minimum/average mode for both Ami33 and Ami49. We achieve results that the average chip area is better than the minimum chip area reached by $O$-tree-based algorithm while their wire lengths are comparable.

Fig.5 is a placement example of Ami33. Its area usage is 96\% (its area is 1.20mm$^2$) and wire length is 63.1.  Fig.6 is a placement example of Ami49. Its area usage is 94\% (its area is 37.7mm$^2$) and wire length is 894.3.

In some cases, the total wire length of the present implementation of LFF is not as good as CR or $O$-tree. There are 3 reasons. 1) We use the local net number as one of the optimization objects to optimize the interconnection of the placement, while CR and $O$-tree use the total wire length measured by half perimeter of each net to directly optimize the total wire length of a placement. Actually, the local interconnection in a circuit is dominant. The more nets between two modules, the closer they should be packed. Because VLSI placement is a multiple-object optimization problem (here we only optimize area and total wire length), emphasizing one object will affect the other. It is obvious that the local interconnection optimization is consistent with the total wire length optimization only in a reasonable range. It has been proved by experiment that a placement which has a longer total wire length may have much less congestion\cite{11}, so a placement of slightly longer total wire length does not mean a worse placement, and the key point is that all the optimization objects of a placement is within a reasonable range. Besides, our method can reduce the number of long nets in a placement effectively, which is very important to the performance of a circuit. 2) We use an indirect method to optimize the total wire length and the interconnection estimation model is a sketchy one. When we use the total wire length of the local nets measured by half perimeter model to estimate the local interconnection, we have another implementation of the LFF. We name it as Cluster LFF (C-LFF). The experimental results are listed in Table 4. We can see that the total wire length of C-LFF is improved a lot while still with the better area usage than $O$-tree. Obviously, the difference of the total wire length is partly because LFF uses a different interconnection estimation model. 3) Placement based on LFF is the automatization of the actual manual placement process. It does placement within a fixed die, while CR and $O$-tree do placement with no area limitation, but just to find the placement with the smallest area or shortest total wire length. So the direct and simple comparison in the tables is unfair to LFF. When LFF tries with both area and aspect ratio varying in a wide range, the total wire length will be decreased dramatically. For example, if we set the area usage of Ami33 as 91\%, we can get the smallest total wire length 43.9044\cite{12}. This placement is much better than the results of CR and $O$-tree in area or in total wire length. In practice, the area and aspect ratio of a placement both have limitations. When both area and aspect ratio

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig5.png}
\caption{Ami33 placement example.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig6.png}
\caption{Ami49 placement example.}
\end{figure}
vary in a wide range in an algorithm, it may be hard for it to find a practical useful placement, although the results seem “good”. For example, in Table 3, when O-tree gets a placement of Ami49 that has total wire length of 673, it is a very small one compared with LFF, but the placement area usage is only 77.9%. Obviously, it is not a good placement for practical use. In actual placement, when we optimize the area and total wire length simultaneously, a good placement must have a good area usage first, then a shorter total wire length. For a multiple-object optimization problem, emphasizing one object will affect the other. In the tables, LFF always has better area usage than others, while the differences of their total wire lengths are in a reasonable range. Experiments prove that the interconnection estimation models of LFF and other algorithms are more approximate, the difference of the total wire lengths of the placement got by LFF and other algorithms will become smaller. Experiments also prove that a placement with shorter total wire length does not mean a good placement to some extent. So we will integrate the interconnect planning into LFF to optimize the congestion and the performance of a placement. Some effective optimization strategies have been proposed and the theoretical analysis shows the promising of the ongoing researches.

5 Conclusion

Human beings have ever done a good job in solving packing problems. Our work has proved that imitating human behavior in solving complex problems, such as NP-hard problems, is an effective way to find efficient algorithms.

For more complicated packing problems, such as boundary constraint floorplanning, abutment constraint floorplanning, floorplanning with some blocks pre-located in the working area, etc., human also has accumulated experience. To find this kind of effective heuristics and implement them in our algorithm is our future work.

References


DONG SheQin received the B.S. degree in computer science in 1985, M.S. degree in semiconductor physics and device in 1988, and Ph.D. degree in mechanical control and automation in 1996, all from Harbin Institute of Technology. From 1997 to 1999, he worked as a postdoctoral fellow in the State Key Lab. of CAD&CG at Zhejiang University. He is currently an associate professor of the Department of Computer Science and Technology, Tsinghua University. His current research interests include CAD for VLSI, parallel algorithms, multi-media ASIC and hardware design.

WU YuLiang finished his B.S. and M.S. degrees in computer science at Florida International University and University of Miami in 1983 and 1984 respectively. He received Ph.D. degree in electrical and computer engineering from University of California at Santa Barbara in 1994. Before he joined the Chinese University of Hong Kong in January 1996, he had worked at Cadence Design Systems Incorporation as a senior MTS since December 1994, where he worked in the R&D of the silicon synthesis product (PBS) targeting at binding the gap between logic and physical level optimizations for deep-submicron chip designs. His current research interests are mainly related to hard optimization problems in object packings, optimal FPGA architecture designs, and logical/physical design automation (CAD) of VLSI designs.

For the biography of HONG XianLong/GU Jun please refer to P.639, No.5, Vol.18 of this journal.