Large Scale VLSI Module Placement Using LFF Heuristics by Stages *

Wei Shaojun[1,†], Dong Sheqin[1], Hong Xianlong[1], and Wu Youliang[2]

(1 Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)
(2 Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, China)

Abstract: We present a deterministic algorithm for large-scale VLSI module placement. Following the less flexibility first (LFF) principle, we simulate a manual packing process in which the concept of placement by stages is introduced to reduce the overall evaluation complexity. The complexity of the proposed algorithm is $(N_1 + N_2) \times O(n^2) + N_3 \times O(n^4 \lg n)$, where N_1, N_2, and N_3 denote the number of modules in each stage, $N_1 + N_2 + N_3 = n$, and $N_3 \ll n$. This complexity is much less than the original time complexity of $O(n^5 \lg n)$. Experimental results indicate that this approach is quite promising.

Key words: floorplanning; placement; large scale; LFF principle; deterministic placement algorithm
EEACC: 2570
CLC number: TN405.97 **Document code:** A **Article ID:** 0253-4177(2006)05-0812-07

1 Introduction

Floorplanning is designing the layout of circuit blocks or IP blocks on a chip subject to various objectives. It is an early stage of physical design and determines the overall chip performance. A floorplan can be classified into one of two categories: slicing and non-slicing. A slicing floorplan[1,2] can be obtained by recursively cutting a rectangle into two parts by either a vertical line or a horizontal line, while a non-slicing floorplan[3~9] cannot.

Floorplan optimization is a kind of multi-objective optimization where an area and a wire length minimization present a simple but necessary part of practical floorplanning. Research on the floorplanning problem has mostly focused on topological representations[4~9] of floorplans that could be evaluated under the well-known simulated annealing (SA)[10] framework.

The largest benchmark circuit reported in the literature contains no more than 49 modules (MCNC benchmarks). Such a small scale is becoming impractical as the size and complexity of VLSI circuits are increasing. Benchmarks with over 100 modules have been used in some recently published works[11~14]. The need for faster floorplanning algorithms is also growing. Adya *et al.*[11] introduced PARQUET, a SA based floorplanner, in which new types of moves are applied to better guide the local search. Lee *et al.*[12] proposed a multilevel approach using B*-trees (MB*-tree) for large-scale modules. Others are Traffic[13] and BloBB[14], which are both non-SA based approaches.

The less flexibility first (LFF) principle[3] is derived from humanity's accumulated experience in handling rectangle packing problems in daily life. The LFF-based algorithm, which is a simulation of manual packing, is a deterministic and constructive algorithm that is proved to be both effective and efficient for small-scale benchmarks. However, it ignores an important characteristic of manual packing. During the manual packing process, packing resources such as unpacked modules and empty space decrease from sufficient to insufficient, and while packing the modules is easy in the beginning, it becomes difficult in the end. Therefore, based on the LFF principle, we make a simulation in this paper in which we introduce the concept of placement by stages to reduce the overall evaluation complexity. Experiments on GSRC benchmarks show that our approach is quite promising.

* Project supported by the Joint Project by NSFC and Hong Kong RGC (No. 60218004), the National Natural Science Foundation of China (No. 90307005), and the National High Technology Research and Development Program of China (No. 2004AA1Z1050)
† Corresponding author. Email: weisj03@mails.tsinghua.edu.cn
Received 5 December 2005, revised manuscript received 28 February 2006

Execution time can be saved by the new approach compared to the original algorithm proposed in Ref. [3]. Its solution quality compares favorably to that of the state-of-the-art floorplanner PARQUET-3, yet it is much quicker.

2 Preliminaries

2.1 Problem definition

Let $M = \{m_1, m_2, \cdots, m_u\}$ be a set of u rectangular modules and $N = \{n_1, n_2, \cdots, n_v\}$ be a set of v nets which specify the interconnections among the modules. If the width and height of a module is fixed, it is called a hard module; otherwise it is called a soft module. In this paper, we consider only hard modules, and with all modules hard floorplanning becomes placement. A placement $P = \{(x_i, y_i) \mid m_i \in M\}$ is an assignment of rectangular modules m_i with the coordinates of their bottom-left corners being assigned to (x_i, y_i)'s so that no two modules overlap. Placement is optimized by determining P such that the area of the minimum enclosing rectangle of the placement and/or the total length of the nets is minimized.

2.2 Less flexibility first principle

The LFF[3] principle is derived from humanity's accumulated experience in everyday life. For example, when masons plank a floor with rectangular wood blocks they fill first against the corners of the boundary, then along the boundary lines, and last inside the hollow spaces. Also, the larger and longer blocks are packed before the smaller and shorter ones. Such rules of thumb constitute the LFF principle.

Different flexibilities can be defined for various objectives. Figure 1 illustrates a definition of empty space flexibility. If the empty space is near a corner (Fig. 1(a)), then a module can move freely in 3 directions when it is packed there. If the empty space is near an edge (Fig. 1(b)), it can move freely in 5 directions. If the empty space is near nothing (Fig. 1(c)), it can move freely in 8 directions. Let $f(a)$, $f(b)$, and $f(c)$ denote the empty space flexibility in Figs. 1(a), (b), and (c), respectively. We define $f(a) < f(b) < f(c)$, which means that the priority should be: corner-packing > side-packing > hollow-space-packing.

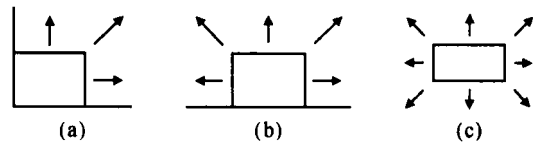

Fig. 1 Three kinds of empty spaces

Another example is module flexibility, which can be defined as

$$f_{\text{module}} = -\left\{\frac{w_m h_m}{WH} + \frac{\max(w_m, h_m)}{\min(W, H)}\right\} \quad (1)$$

where w_m and h_m denote the width and height of the module, and W and H denote the width and height of the pre-specified work space. Equation (1) indicates that the large or long modules should be considered first during the packing process.

2.3 LFF-based placement

The process of LFF-based placement can be briefly described as follows:

In the beginning, a fixed rectangular area is chosen as the work space. The modules are then put one by one into it, and the LFF heuristics are applied for the definition, evaluation, and selection of packing schemes (details will be given in the following sections). If all the modules can be packed without overlapping and exceeding the boundary, then a feasible solution will be achieved in the end.

3 Implementation

3.1 Data structures

According to the empty space flexibility, the best packing is corner-packing, which is the most area-saving kind of packing. A corner should be regarded as constrained by two perpendicular lines (Fig. 1(a)). By tracking the contour of the empty space, all the corners can be found. There are 4 corners in the beginning, and each time if a module is packed, at least 1 corner becomes occupied and 2 new corners are created. Thus the number of corners is $O(n)$. If the space near a corner is too small to hold any modules, we mark it as dead space and slightly clip the contour to create friendly corners (Fig. 2).

Two kinds of orientations are defined for each module: horizontal and vertical. Swapping the

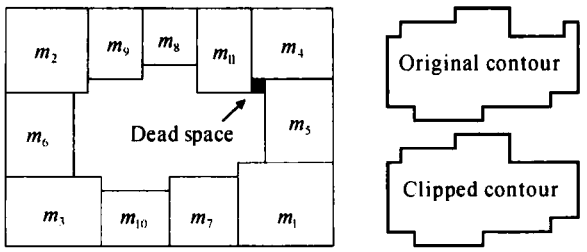

Fig. 2 Mark the dead space and clip the contour

width and height of a module changes its orientation. If a module can be validly packed against a corner in one of the orientations, we call such a scheme a "candidate corner packing scheme" (CCPS). In our implementation, a CCPS is represented by a four-tuple:

$$\langle \text{module id}, \text{orientation}, x_c, y_c \rangle$$

where (x_c, y_c) denotes the coordinates of the corner against which the module is placed. Each time, by testing all the remaining unpacked modules near the corners, we can get a CCPS list, and one of the best is chosen. The positions of packed modules are saved in a k-d tree data structure[16] by which module overlap detection can be done in $O(\log n)$ time.

3.2 CCPS evaluation

Corner-packing alone is insufficient for area optimization. The CCPS list must be carefully evaluated to choose the most area-saving and wire-saving one. In this subsection, we will discuss the heuristics that are used in CCPS evaluation.

Heuristic 1: Higher packing density first

After a module is packed, we would like to ensure that the remaining modules can also be packed easily. Therefore, we evaluate how the CCPS allows for the packing of the other modules.

Definition: The packing density of a CCPS is the area of all the modules that could be packed, if the CCPS is performed, into the area of the work space.

Therefore, a CCPS with a higher packing density should be considered first. To evaluate the packing density, we virtually perform the CCPS and then put the left-over modules into the work space one by one in ascending order of their respective module flexibility. Here the term pseudo means that such a packing process is just a test and it can be reverted after estimation. This strategy is greedy in the sense that it tries to pack as many modules as possible. Let A_{space} denote the area of the work space and A_{pseudo} denote the sum of the area of actually packed modules and virtually packed ones. The fitness value (FV) of a CCPS is calculated as:

$$FV_{packing} = A_{pseudo} / A_{space} \quad (2)$$

With this approach, the time complexity to evaluate a CCPS is $O(n^2 \log n)$[3].

Heuristic 2: Longer packing radius first

If the modules are put tightly along the boundary of the work space from outside to inside and are evenly distributed, then the shape of the empty space will always be kept regular, which is in favor of the packing process.

Definition: The packing radius of a CCPS is the distance between its corner and the center of the work space.

By preferring a CCPS with a longer packing radius, the heuristic mentioned above can be implemented easily. Therefore, the fitness value of a CCPS is

$$FV_{radius} = \sqrt{(x_c - x_o)^2 + (y_c - y_o)^2} \quad (3)$$

where (x_o, y_o) denotes the coordinates of the center of the work space. Obviously, the time complexity of evaluating a CCPS is only $O(1)$.

Heuristic 3: Less module flexibility first

In the stricter LFF-based algorithm[3], the module with the fewest CCPS will be packed first since such a module is regarded as the most difficult one to be packed. However, this hypothesis is deficient. In some cases, e. g. in the beginning of the packing, each unpacked module has the same number of CCPS, so that we cannot distinguish longer or larger modules from shorter or smaller ones.

The module flexibility mentioned in Section 2.2 implies that longer or larger modules should have more chances to be used first. Also, modules with less module flexibility are sure to have fewer CCPS when empty space becomes scarce. As a result, module flexibility is a good factor for CCPS evaluation:

$$FV_{module} = \frac{w_m h_m}{WH} + \frac{\max(w_m, h_m)}{\min(W, H)} \quad (4)$$

Note that FV_{module} is the negative of f_{module} since we use a greater fitness value to denote less flexibility. The time complexity to get FV_{module} is only $O(1)$.

Heuristic 4 : Higher connection density first

For two modules m_1 and m_2, if m_1 has more inner wires (nets between a module and the packed ones) and fewer outer wires (nets between a module and the unpacked ones) than m_2, then m_1 should be considered first.

Definition: The connection density (DC) of a module is the number of its inner wires divided by the sum of the number of its inner wires and outer wires:

$$CD_{module} = n_i / (n_i + n_o) \quad (5)$$

where n_i and n_o denote the number of inner wires and outer wires of a module, respectively. With this definition, the fitness value of a CCPS is CD_{module}, whose calculation complexity is $O(\bar{p})$, where $\bar{p}$ is an average number of nets connecting a module.

Heuristic 5 : Shorter local wire length first

For wire optimization, we tend to put a module in a position where the length of the wires connecting it to the packed modules (local wire length) is as short as possible. In Fig. 3, there are totally 2 nets between module m and the packed ones. If m is packed at position 1, the local wire length would be shorter than if packed at position 2.

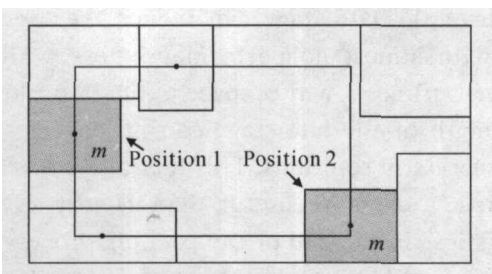

Fig. 3 An example of local connections

With this heuristic, the fitness value of a CCPS to be evaluated equals the negative of local wire length ($-WL_{local}$) of the module. In our implementation, the local wire length is evaluated using the half perimeter metric, as for the global wire length evaluation, and the evaluation complexity equals $O(\bar{p})$.

3.3 Placement by stages

Although the LFF-based algorithm is a simulation of manual packing, it ignores an important characteristic. During the manual packing process, packing resources such as the unpacked modules and the empty spaces are depleted, and it is easy to pack the modules in the beginning but difficult in the end. The concept of placement by stages, which involves dividing the placement process into stages and using different packing rules, is usually used by human-beings. With the definitions of the heuristics in Section 3.2, we can incorporate this concept into the LFF-based algorithm. We divide it into 3 stages in our implementation, namely the early stage, the middle stage, and the late stage, determined by the ratio of the number of packed modules to the total number of modules.

For Heuristics 1 ~ 5, their respective time complexity and effects are measured to determine which ones will be used in which stage. Heuristic 1 is good for area optimization, but its execution time increases dramatically when the problem scale increases, and thus it is only suitable for small-scale problems. The time complexities of Heuristics 2 and 3 are lower, but when the packing is to be finished, more consideration must be taken to better utilize the space. Thus, Heuristics 2 and 3 are applied in the early and middle stages while Heuristic 1 is applied in the late stage. For wire optimization, both Heuristics 4 and 5 should only be used after a certain number of modules have been packed.

The early stage

In this stage, the CCPS are evaluated with Heuristics 2 and 3. The fitness value of a CCPS is calculated as

$$FV_{CCPS} = w_1 \times FV_{radius} + w_2 \times FV_{module} \quad (6)$$

where $w_1 + w_2 = 1$. We find in the experiments that the second term of Eq. (6) affects the packing greatly. One way to simplify the normalization of w_1 and w_2 is to use a "2-step evaluation" in the implementation. First, we get the range of FV_{module}, and only the CCPS of the module whose FV_{module} are in the last 10 % of the range are considered in the second step. Second, the FV_{radius} of each remaining CCPS are calculated and the CCPS with the best FV_{radius} are chosen.

The middle stage

In this stage, since a certain number of modules have already been packed, wire length optimization should be taken into account. As a result, the CCPS are evaluated with Heuristics 2, 3, 4, and 5. Similarly, we use the "2-step evaluation" in our implementation. In the first step, the fitness value is

calculated as

$$FV'_{CCPS} = w_1 \times FV_{module} + w_2 \times CD_{module} \quad (7)$$

where $w_1 + w_2 = 1$. Only the CCPS of the module whose FV'_{CCPS} are in the last 10 % of the range are considered in the second step. In the second step, the fitness value is calculated as

$$FV''_{CCPS} = w_1 \times FV_{radius}[- w_2 \times WL_{local}] \quad (8)$$

where $w_1 + w_2 = 1$, and "[ ]" means the term only makes sense in comparing two CCPS that belong to the same module.

The late stage

In this stage, there are only a small number of unpacked modules remaining and more consideration must be taken to better utilize the space. Unlike the middle stage, Heuristic 1 is used instead of Heuristic 2 or Heuristic 3. There is no CCPS distillation in the first step, and in the second step the fitness value is calculated as

$$FV''_{CCPS} = w_1 \times FV_{packing}[- w_2 \times WL_{local}] \quad (9)$$

3.4 Overall time complexity

Since for large scale problems, $O(n) \gg O(\bar{e})$, we can ignore $O(\bar{e})$ in time complexity evaluation. The over all time complexity of the new LFF algorithm is

$$(N_1 + N_2) \times O(n^2) + N_3 \times O(n^4 \lg n) \quad (10)$$

where N_1, N_2, and N_3 denote the number of modules in the 3 stages, $N_1 + N_2 + N_3 = n$, and $N_3 \ll n$. This is much less than the $O(n^5 \lg n)$ complexity in Ref. [3].

4 Experiments

We implement our algorithm in ANSI C. In order to find the minimum bounding box sizes for successful solutions, we continue our experiments with the size of the bounding box increasing gradually. To avoid comparing pad placement algorithms, wiring results do not include nets going to pads. (Note that if only area optimization is needed, we merge the early stage with the middle stage and discard the heuristics for wire optimization in all the stages.) GSRC benchmark circuits are used in our experiments. For comparison, we choose the state-of-the-art floorplanner PARQUET-3 which uses either sequence pair (SP)[4] or B*-tree[8] as the topological representation of a floorplan. All experiments are conducted on a 2.3GHz Pentium4 workstation with 4GB RAM, running Linux.

In Table 1, we report the results of different stage division schemes on n100 when optimizing area only. We vary the aspect ratio of the outline from 1 to 2 with an increment of 0.02. We define "success rate" by the number of successful solutions divided by 50 (which is the total number of runs). To the left of the table are listed the 4 kinds of schemes we tested. A scheme, e.g. 0～10 %～95 %～100 %, means that 10 % and 95 % are set as the dividing points of the three stages. For looser outlines (white space ≥7 %), shortening the length of the late stage can reduce the execution time with almost no performance loss, while for tighter outlines (white space ≤6 %), prolonging the length of the late stage can make it easier to find successful solutions. This fact also proves that Heuristic 1 is more useful than Heuristic 2 and Heuristic 3 in the end of the packing process if we want to find better solutions.

Table 1 Comparison of stage division for n100 (Area optimization only)

Stage division scheme	White space	A		B		C		D	
		Success rate	Time/s	Success rate	Time/s	Success rate	Time/s	Success rate	Time/s
A:0～0～0～100 %	5 %	2 %	0.13	6 %	0.14	20 %	0.36	24 %	1.40
B:0～0～95 %～100 %	6 %	48 %	0.12	50 %	0.13	66 %	0.34	72 %	1.26
C:0～0～90 %～100 %	7 %	98 %	0.13	100 %	0.14	98 %	0.30	96 %	1.00
D:0～0～85 %～100 %	8 %	100 %	0.13	100 %	0.14	98 %	0.27	100 %	0.82

We also compare the results of different stage division schemes on n100 in Table 2 when simultaneously optimizing area and wire length. The white space is set to be 10 % in these experiments. Since wire optimization is not considered in the early stage, these results show that the final wire length is affected by the time when we start to consider wire optimization.

Table 2 Comparison of stage division for n100 (Area + wire optimization)

Stage division scheme	Wire (avg)	Wire (min)	Wire (max)	Time/s
0～0～90 %～100 %	138	126	147	0.36
0～10 %～90 %～100 %	138	132	143	0.40
0～20 %～90 %～100 %	143	130	150	0.43
0～30 %～90 %～100 %	143	142	161	0.44

In Table 3(Table 4) we list the results of LFF and PARQUET-3 on area (area + wire) optimization. We use the default parameters of PARQUET-3,and its results are the best of 50 runs for each benchmark circuit. LFF outperforms PARQUE-3 on area optimization with white space < 4 % for all the three benchmarks. Also,LFF tends to achieve smaller white space when wire optimization is considered,and its wiring results are comparable to those of PARQUET-3.

Table 3 Comparison of LFF with PARQUET-3 (Area optimization only)

Circuit	LFF	SP	B*-tree
(# modules)	WS/ time	WS/ time	WS/ time
n100(# 100)	3.85 %/ 0.29s	7.20 %/ 5.10s	4.52 %/ 2.69s
n200(# 200)	3.61 %/ 1.07s	8.74 %/ 26.6s	5.07 %/ 11.3s
n300(# 300)	3.59 %/ 3.38s	9.60 %/ 60.8s	5.34 %/ 24.0s

Table 4 Comparison of LFF with PARQUET-3 (Area + wire optimization)

Circuit	LFF	SP	B*-tree
	WS/ wire/ time	WS/ wire/ time	WS/ wire/ time
n100	8.2 %/ 132/ 0.4s	7.5 %/ 121/ 18s	11 %/ 123/ 12s
n200	7.8 %/ 271/ 2.7s	12 %/ 268/ 101s	10 %/ 259/ 63s
n300	8.9 %/ 415/ 6.5s	13 %/ 422/ 256s	12 %/ 397/ 128s

5 Conclusion

We have presented a deterministic and constructive algorithm for the large-scale VLSI module placement problem. We base the algorithm on LFF heuristics and introduce the concept of placement by stages to reduce the evaluation complexity.

Our approach is good for generating optimized results in a short time,and we believe that even if time were less critical,it could also be used to provide initial solutions quickly for floorplan algorithms such as simulated annealing to reduce the overall time.

References

[1] Otten R H J M. Automatic floorplan design. DAC,1982:261

[2] Wong D F,Liu C L. A new algorithm for floorplan design. DAC,1986:101

[3] Dong Sheqin, Hong Xianlong, Wu Youliang, et al. VLSI block placement using less flexibility first principles. Proc IEEE ASPDAC, Yokohama,2001:601

[4] Murata H, Fujiyoshi K, Nakatake S, et al. VLSI module placement based on rectangle-packing by the sequence-pair. IEEE Trans CAD,1996,15(12):1518

[5] Hong Xianlong, Dong Sheqin, Huang Gang, et al. Corner block list representation and its application to floorplan optimization. IEEE Trans Circuits Syst Ⅱ: Express Briefs,2004, 51(5):228

[6] Lin J M,Chang Y W. TCG:a transitive closure graph-based representation for non-slicing floorplans. DAC,2001:764

[7] Nakatake S,Murata H,Fujiyoshi K,et al. Module placement on BSG-structure and IC layout applications. ICCAD,1996: 484

[8] Chang Y C,Chang Y W,Wu G M,et al. B*-trees:a new representation for nonslicing floorplans. Proc Design Automation Conf,2000:458

[9] Guo P N,Cheng C K,Yoshimura T. An O-tree representation of nonslicing floorplan and its applications. Proc Design Automation Conf,1999:268

[10] Kirkpatrick S,Gelatt C D,Vecchi M P. Optimization by simulated annealing. Science,1983,220(4598):671

[11] Adya S N,Markov I L. Fixed-outline floorplanning:enabling hierarchical design. IEEE Trans VLSI,2003,11(6):1120

[12] Lee H C,Chang Y W,Hsu J M,et al. Multilevel floorplanning/ placement for large-scale modules using B*-trees. DAC, 2003:812

[13] Sassone P,Lim S K. A novel geometric algorithm for fast wire-optimized floorplanning. Proc International Conference on Computer-Aided Design,2003

[14] Chan H H, Markov I L. Practical slicing and non-slicing block packing without simulated annealing. Proceedings of the ACM Great Lakes Symposium on VLSI,2004:282

[15] Wu Y L,Huang W,Lau S C,et al. An effective quasi-human based heuristic for solving rectangle packing problem. European Journal of Operational Research,2002,141(2):341

[16] Bentley J L. Multidimensional binary search trees used for associative searching. Communication of ACM,1975,18(9):509

基于分阶段 LFF 策略的大规模模块布局方法*

魏少俊[1,†] 董社勤[1] 洪先龙[1] 吴有亮[2]

(1 清华大学计算机科学与技术系，北京 100084)
(2 香港中文大学计算机科学与工程系，香港)

摘要：提出了一种用于求解大规模 VLSI 模块布局问题的确定性方法. 该方法在“最小自由度优先”原则的基础上，模拟人工布局过程提出了“分阶段布局”的思想. 分阶段布局就是将布局过程按照布局完成的比例划分成若干个阶段，再将各种启发式策略适当地应用到各个阶段中，从而改善算法的整体性能. 理论上，算法的时间复杂为 $(N_1 + N_2)O(n^2) + N_3O(n^4\lg n)$，其中 N_1, N_2, N_3 为各个阶段的模块数目，$N_1 + N_2 + N_3 = n$，且 $N_3 \ll n$，比原有的最小自由度优先算法复杂度 $O(n^5\lg n)$ 小很多. 实验结果也表明该方法很有潜力.

关键词：布图规划；布局；大规模；LFF 原则；确定性布局算法
EEACC：2570
中图分类号：TN405.97 **文献标识码**：A **文章编号**：0253-4177(2006)05-0812-07

*国家自然科学基金委与香港 RGC 联合资助项目(批准号:60218004)，国家自然科学基金(批准号:90307005)及国家高技术研究发展计划(批准号:2004AA1Z1050)资助项目

† 通信作者. Email:weisj03@mails.tsinghua.edu.cn

2005-12-05 收到，2006-02-28 定稿