

LaySeq: A New Representation for Non-Slicing Floorplans

S. SATHIAMOORTHY
Department of Computer Science and Engineering
Thiagarajar College of Engineering, Madurai 625015, INDIA
E-mail : sathi@DACafe.com

Submitted to **ACM/IEEE International Symposium on Physical Design 2002**, California.

Abstract

In this paper, we propose LaySeq a new representation for non-slicing floorplans and show its superior properties. Layseq uses only $n \lceil \lg n \rceil$ bits for a floorplan of n rectangular blocks. The solution space size of layseq is just $O(n!)$. This is very smaller than that of all recent representations. Given a layseq it takes only linear time to construct the floorplan. Layseq is very simple and easy to implement representation. Based on this new structure a hybrid genetic algorithm for floorplanning is given. We show that layseq is efficient in handling rectilinear blocks too. Experimental results show that layseq results in smaller silicon area than earlier approaches.

1. Introduction

Due to increase in design complexity, circuit sizes are getting larger. IP blocks are intensively used to reduce the design complexity. This trend makes floorplanning or building block placement much more critical process. A fundamental problem in floorplanning lies in the representation on geometric relation between the modules. The representation directly affects the quality of the floorplan/placement and the complexity of the process. Thus it is important to develop an efficient representation for floorplan/placement design.

1.1 Previous Work

VLSI floorplans are often grouped into two categories, the *slicing structure* [5] and the *non-slicing structure* [1, 2, 3, 4, 6, 7]. A binary tree whose leaves denote modules can represent a slicing structure, and internal nodes specify horizontal or vertical cut lines. Wong and Liu proposed an algorithm for slicing floorplan designs [5]. They presented a normalized Polish expression to represent a slicing structure, enabling the speed-up of its search procedure. However, this representation cannot handle non-slicing floorplans. There are $O(n! 2^{3n-3}/n^{1.5})$ combinations of the slicing tree structure. It takes only $O(n)$ time to derive a floorplan from a representation. Recently, researchers have proposed several representations for non-slicing floorplans, such as sequence pair [6], bounded slicing grid (BSG) [7], O-

tree [1], Transitive Closure Graph (TCG) [2], Corner Block List (CBL) [3] and B* Trees [4]. Onodera et al. in [8] used the branch-and-bound method to solve the module placement problem. This approach has very high time complexity and is thus feasible for only small problem sizes. The number of combinations of their representation model is $O(2^{n(n+2)})$. Murata et al. in [6] proposed the sequence-pair representation (SP) for rectangular module placement. The main idea is to use a sequence pair to represent the geometric relation of modules, place the modules on a grid structure, and construct corresponding constraint graphs to evaluate cost. They defined the *P-admissible solution space*, which satisfies the following four requirements: (1) the solution space is finite, (2) every solution is feasible, (3) packing and cost evaluation can be done in polynomial time, and (4) the best evaluated packing in the space corresponds to an optimal placement. The number of combinations of SP is $O((n!)^2)$. From SP to its corresponding placement, it takes $O(n^2)$ time. In BSG, a special $m \times n$ grid is applied for the placement of n blocks. BSG has $O(n^2)$ complexity and $n!C(n^2, n)$ combinations.

Guo et al. in [1] proposed the O-tree representation for a left and bottom compacted placement. In an O-tree, a node denotes a module and an edge denotes the horizontal adjacency relation of two modules. A pair of strings encodes each O-tree, denoting the DFS traversal order of the tree. The placement after O-tree packing might not be compacted. A sequence of compaction operations along the x and/or y directions may be needed to make all modules compacted to the left and bottom, possibly resulting in a mismatch between the original representation and its placement. O-tree has very small solution space $O(n! \cdot 2^{n-2}/n^{1.5})$ and $O(n)$ time complexity. Chang et al. recently in [4] presented a binary tree-based representation for a left and bottom compacted placement, called B*-tree, and showed its superior properties for operations. In a B*-tree, a node denotes a module, the left child of a node represents the lowest adjacent module on the right, and the right child represents the visible module above and with the same x coordinate. Similar to O-tree, the representation could be changed after a previously placed module may occupy packing since the space intended for placing a module. Therefore, given an O-tree or a B*-tree, it may not be feasible to find a placement corresponding to its original representation, and thus they are not P-admissible. B*-tree also has very small solution space $O(n! \cdot 2^{n-3}/n^{1.5})$ and $O(n)$ time complexity.

Lin et al. in [2] proposed transitive closure graph based representation for non-slicing floorplans. TCG describes the geometric relations among modules based on two graphs, namely a horizontal transitive closure graph and a vertical transitive closure graph. Like B* trees, TCG does not need any additional constraint graphs during cost evaluation. TCG has an $O(n^2)$ complexity for packing. The size of solution space for TCG is $O((n!)^2)$. Hong et al. in [3] proposed a corner block list (CBL) representation for non-slicing floorplans. Like sequence pair and BSG, but unlike the tree-based representations, CBL can represent general incompact non-slicing floorplans. CBL has smaller feasible solution space and a

faster packing scheme; however, it is not P-admissible. The time complexity to transform a corner block list to a corresponding placement is $O(n)$. The number of combinations in CBL is $O(n! 2^{3n-3}/n^{1.5})$.

1.2 Our Contributions

In this paper, we present a new representation LaySeq, which can represent general non-slicing floorplans. It defines a floorplan independent of the block sizes. Using layseq we propose a genetic algorithm for VLSI floorplan. The advantages of LaySeq are as follows:

1. The time complexity to transform a layseq to a placement configuration is $O(n)$.
2. The number of configurations of layseq is $O(n!)$. The solution space size of layseq is just square root of the solution space size of TCG and SP. Even the solution space size of O-tree, which is minimum among reported approaches, is very large when compared to solution space of layseq. When a floorplan consists of 50 blocks, the solution space size of O-tree is **9×10^{26}** times that of layseq.
3. Layseq takes only $n(\lceil \lg n \rceil)$ bits to describe, where $\lceil \lg n \rceil$ denotes the minimum integral number which is not less than $\lg n$.
4. Layseq represents the floorplan independent of the block sizes, so we can use layseq to optimize the blocks with multiple configurations of heights and widths. This is an advantage over the O-tree representation.
5. As Layseq representation does not contain information about block size, it is very flexible for handling the floorplanning problems with various types of modules (e.g., hard, soft, pre-placed, rectilinear) directly.
6. The area cost after perturbing can be recomputed incrementally as only modules next to those whose positions are changed in the sequence need to be considered.
7. Like B*-trees and TCG, but unlike SP, BSG, CBL and O-tree, layseq does not need to compute construct additional constraint graphs for the cost evaluation during packing, implying faster runtime.
8. As shown in results section rectilinear modules can be handled as easily as rectangular modules. In [10] rectilinear modules should be analyzed and decomposed into L-shaped modules. In [4] rectilinear modules should be partitioned into several rectangular sub-modules by slicing from left to right along each vertical boundary.
9. LaySeq is very simple, easy to understand and implement.

We summarize the properties of several recently published representations for non-slicing floorplans in Table 1. The fourth condition for P-admissibility is not satisfied by layseq, since the best evaluated packing in the solution space need not be an optimal one. But experimental results show that layseq is

able to find better solutions than those found by the P-admissible representations too. Solution space is $O(n!)$, which is very smaller than all existing approaches, and will result in quicker convergence towards the best solution.

Representation	SP	BSG	O-tree	B*-tree	CBL	TCG	Layseq
Is P-admissible?	Yes	Yes	No	No	No	Yes	No
Need sequence encoding?	Yes	No	Yes	No	Yes	No	No
Constraint graphs for packing?	Yes	Yes	Yes	No	Yes	No	No
Solution space size	$n!^2$	$2^{n(n+2)}$	$\frac{n!2^{2n-2}}{n^{1.5}}$	$\frac{n!2^{3n-3}}{n^{1.5}}$	$\frac{n!2^{5n-5}}{n^{1.5}}$	$n!^2$	$n!$
Runtime for packing	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n)$
Number of bits to describe floorplan	$2n[\lg n]$	-	$2n+n[\lg n]$	$6n+n[\lg n]$	$3n+n[\lg n]$	-	$n[\lg n]$

Table 1: Properties of representations. Here, n is the number of modules in the placement.

The paper is organized as following. Section 2 states the floorplanning problem. Section 3 presents a new representation for non-slicing floorplans - LaySeq. Section 4 gives the details about the genetic algorithm designed for searching the solution space of LaySeq. Experimental results are reported in section 5. Section 6 contains the concluding remarks.

2. Problem Definition

Given a set of n rectangular modules with their width and height the problem is to place the modules without overlapping. Due to increase in design complexity the modules need not be rectangular, they might be rectilinear or soft modules. The goal is to minimize area and wirelength induced by a placement. Area is typically the area of the final enclosing rectangle of all the modules in the floorplan. Wirelength is mostly approximated by half perimeter method. It will be clear that traditional approaches for wirelength optimization readily apply to the layseq representation. We shall thus focus on area optimization in this paper.

3. The LaySeq Representation

LaySeq consists of a list of length n representing n blocks in a random order. B_L and B_H are user specified values representing the minimum and maximum number of blocks, which form base for the floorplan. The value for B_L and B_H should lie in the range (one, n) and B_L should be less than or equal to B_H . The algorithm for obtaining placement from layseq is:

Step1: Assign BaseSize a random number in the range (B_L, B_H).

Step2: Remove first block in the layseq, randomly rotate it and place it at the (0,0) position.

Step3: Until BaseSize numbers of blocks are not placed

- a. Remove next block from the layseq, randomly rotate it.
- b. Place it on the bottom line, right to previously placed block such that the placement is LB compact.

Step4: Assign MaxX the width of the base formed by so far placed blocks.

Step5: MaxX is the width of the floorplan and all remaining blocks should be placed such that the MaxX width constraint of the floorplan is maintained.

Step6: Until all blocks are placed

- a. Remove next block from layseq.
- b. Place the block at a position over already placed blocks, which results in least increase in height of the floorplan. This results in B compact placement.
- c. Make the floorplan L compact one.

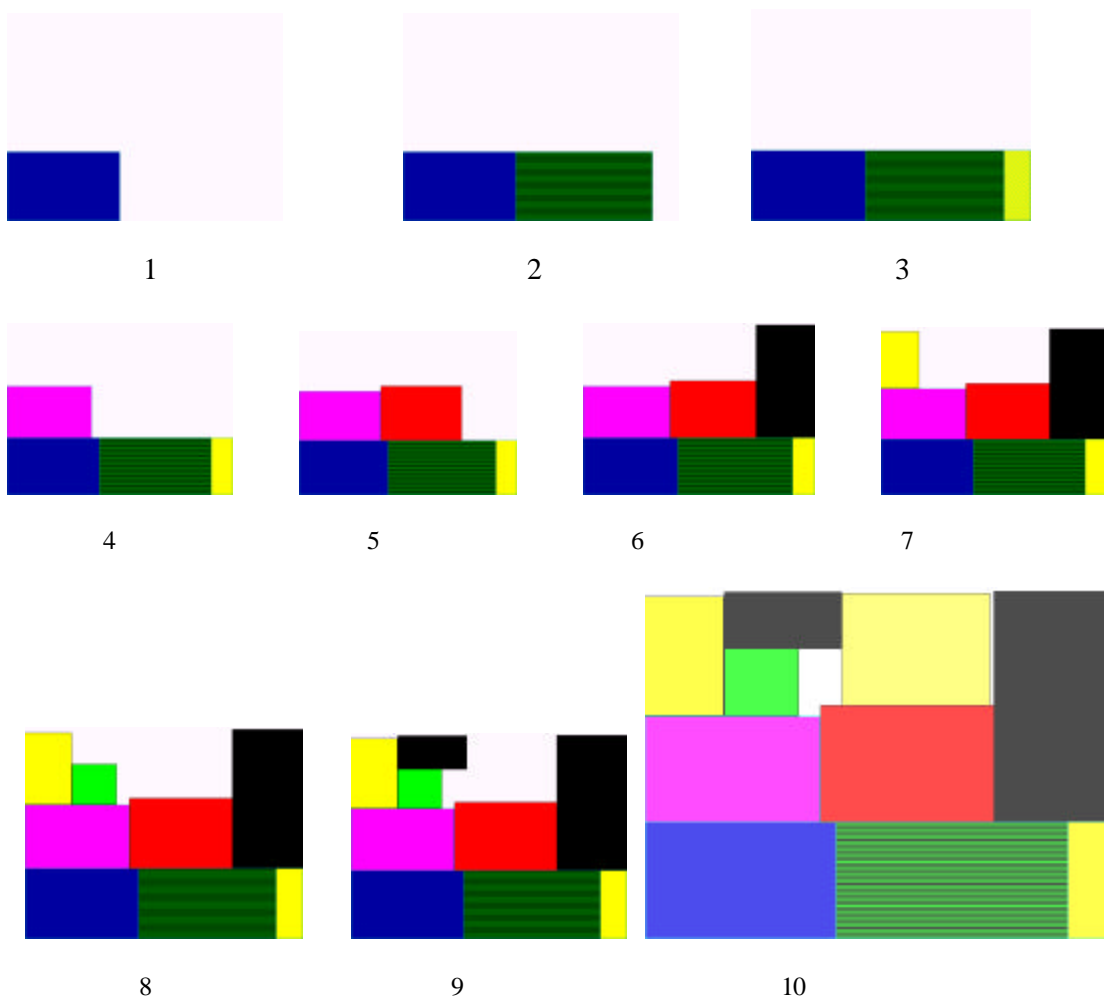


Figure 1: Example for LaySeq.

Example for layseq to placement transformation is shown in the Figure 1. The floorplan is obtained for a MCNC benchmark ‘Xerox’ containing 10 blocks. The BaseSize was three in this case. First three images show how base of the placement is formed. Placing remaining blocks according to the algorithm results in a placement of area 19.8.

4.The Hybrid Genetic Algorithm

A hybrid genetic algorithm (HGA) was designed to search the solution space of layseq for best floorplan. The HGA used is:

Step 1: Initialize population randomly

Step 2: Apply **Local Search** algorithm to the initial population

Step 3:

- ◆ Select two parents randomly
- ◆ Apply **Crossover** between parents and generate an offspring
- ◆ Apply **Local Search** algorithm to the offspring.
- ◆ If Fitness (offspring) > Fitness (any one of the parents) then replace the weaker parent by the offspring

Step 4: **Mutate** any one randomly selected layout from population with 1% probability.

Step 5: Repeat steps 3 and 4 until end of specified number of iterations.

The genotype representation is just the layseq itself. Crossover operator used is single point crossover. Mutation operator randomly changes the order of blocks in layseq.

4.1 The Local Search Algorithm

The Local Search algorithm will select q consecutive blocks $(b_p, b_{p+1}, \dots, b_{p+q-1})$ from the layseq and it arranges those blocks in such a way that the fitness of layseq is maximum by searching all possible ordering of those blocks. The value of p varies from 1 to $n-q$, where n is the number of cities.

5. Experimental Results

The experiments are carried out for the MCNC floorplanning benchmarks used in [1, 2, 3, 4]. The HGA using layseq representation was implemented in the C++ programming language on a Pentium III (800 MHz) machine with 128 MB memory. The population size was kept at 50. The BaseSize range is (1,8). During local search, searching is done with keeping the value of q random one among {2,3,4,5}. We performed two sets of experiments: one was based on MCNC benchmark circuits and other on some rectilinear modules.

Circuit	O-tree [1]		B*-tree [4]		Enhanced O-tree [9]		CBL [3]		TCG [2]		LaySeq	
	Area (mm ²)	Time (sec)	Area (mm ²)	Time (sec)	Area (mm ²)	Time (sec)	Area (mm ²)	Time (sec)	Area (mm ²)	Time (sec)	Area (mm ²)	Time (sec)
Apte	47.1	38	46.92	7	46.92	11	NA	NA	46.92	1	46.92	1
Xerox	20.1	118	19.83	25	20.21	38	20.96	30	19.83	18	19.80	10
Hp	9.21	57	8.947	55	9.16	19	-	-	8.947	20	8.947	3
Ami33	1.25	1430	1.27	3417	1.24	118	1.20	36	1.20	306	1.18	132
Ami49	37.6	7428	36.80	4752	37.73	406	38.58	65	36.77	434	36.37	483
Comp.	2.83%	-	1.79%	-	2.65%	-	4.52%	-	0.59%	-	0.00%	-

Table 2: Area and runtime comparisons between O-tree (on SUN Ultra60), B*-tree (on SUN Ultra-I), enhanced O-tree (on SUN Ultra60), CBL (on SUN Sparc20), TCG (on SUN Ultra60), and LaySeq (on Pentium III) for area optimization.

The area and runtime comparisons between O-tree, B*-tree, enhanced O-tree, CBL, TCG, and LaySeq for MCNC benchmarks are listed in Table 2. As shown in Table 2, LaySeq achieves average improvements of 2.83%, 1.79%, 2.65%, 4.52%, and 0.59% in area utilization compared to O-tree, B*-tree, enhanced O-tree, CBL, and TCG respectively. The runtimes are significantly smaller than O-tree and B*-tree, and comparable to the enhanced O-tree and TCG.

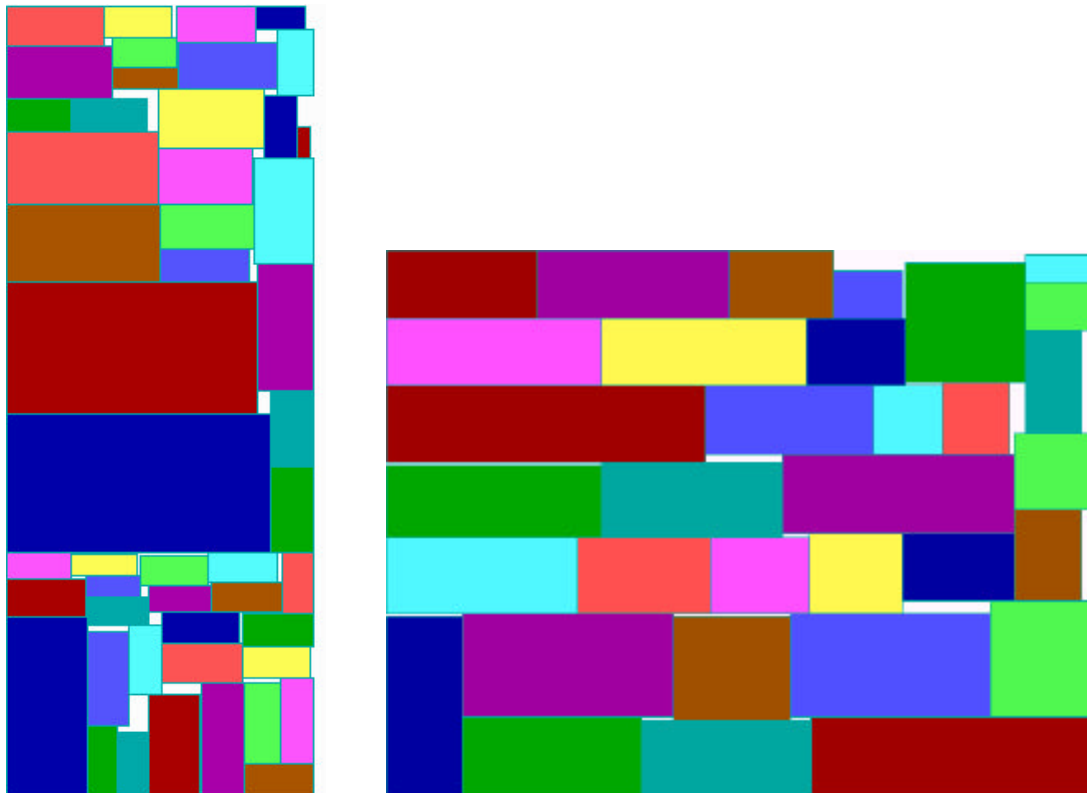


Figure 2: Best placement for ami49 (left) with area 36.37 and ami33 (right) with area 1.18

The experimental results for rectilinear blocks are summarized in Table 3. The test cases are based on MCNC benchmark ami49 by merging blocks to L and T shaped blocks randomly. Near optimum results are obtained for all the three cases. In case of ami49-r11, which contains 35 rectangular modules and 7 L-shaped modules the area obtained was equal to the area obtained for ami49 by TCG. Note that the test cases are more complicated than those used in [4], where the dead space of optimum result will be zero as the test cases are obtained by cutting a rectangle into a set of rectangular, L-shaped and T-shaped blocks. In our case optimum result will also have a significant dead space. The implementation could be easily changed to handle arbitrarily shaped rectilinear blocks.

Test Case	# Rectangular Modules	# L-Shaped Modules	# T-Shaped Modules	Area	Dead space (%)	Runtime (sec)
ami49-r11	35	7	0	36.77	3.68	1752
ami49-r12	28	0	7	37.03	4.35	2547
ami49-r13	14	7	7	37.85	6.43	8164

Table 3: Results for rectilinear modules using LaySeq based Hybrid Genetic Algorithm.

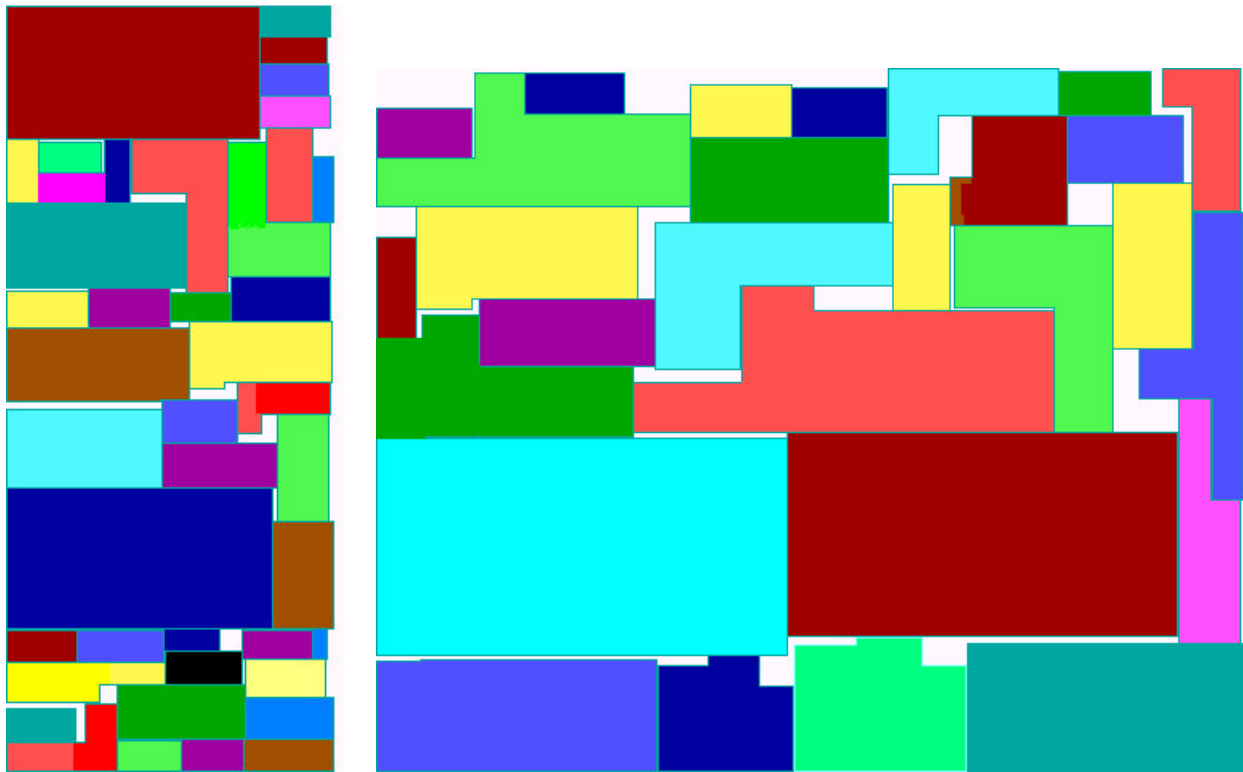


Figure 3: Best placement for ami49-r11 (left) with area 36.77 and ami49-r13 (right) with area 37.85

6. Concluding Remarks

We have presented LaySeq a new representation for non-slicing floorplans. We have shown its superiority over earlier reported approaches. We have also shown the flexibility of the representation for efficient handling of rectilinear blocks. By modifying Step6(b) of LaySeq algorithm alone we could easily incorporate other factors like interconnect cost. Research in interconnect driven floorplanning based on LaySeq is going.

References

- [1] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "Floorplanning Using a Tree Representation," IEEE TCAD February 2001, pp.281-289.
- [2] Jai-Ming Lin and Yao-Wen Chang "TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans," *Proc. DAC*, pp. 764–769, June2001.
- [3] X. Hong, G. Huang, Y. Cai, S. Dong, C.-K. Cheng, and J. Gu, "Corner Block List: An effective and efficient topological representation of non-slicing floorplan," *Proc. ICCAD*, pp. 8–12, Nov. 2000.
- [4] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans," *Proc. DAC*, pp. 458–463, June 2000.
- [5] D. F. Wong, and C.-L. Liu, "A new algorithm for floorplan design," *Proc. DAC*, pp. 101–107, June 1986.
- [6]H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing based module placement," *Proc. ICCAD*, pp. 472–479, Nov. 1995.
- [7] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," *Proc. ICCAD*, pp. 484–491, Nov. 1996.
- [8] H. Onodera, Y. Taniuchi, and K. Tamaru, "Branch-and-bound placement for building block layout," *Proc. DAC*, pp. 433–439, 1991.
- [9] Y.-Pang, C.-K. Cheng, and T. Yoshimura, "An enhanced perturbing algorithm for floorplan design using the O-tree representation," *Proc. ISPD*, pp. 168-173, April 2000.
- [10] J. Xu, P.N. Guo, C.K. Cheng, "Sequence Pair Approach for Rectilinear Module Placement," IEEE TCAD April 1999, pp.484-493