Efficient Floorplanning by O-Tree with Genetic Algorithm

Tetsutarou Hara Katsumi Harashima
Osaka Institute of Technology,
5-16-1, Ohmiya, Asahi-ku, Osaka, 535-8585 Japan
Tel.: +81-6-6954-4305, Fax: +81-6-6957-2136
E-mail : harashima@elc.oit.ac.jp

Abstract: Module placement is an important phase for VLSI layout design. However, huge time is necessary to obtain the optimal layout. This paper proposes a block packing method using O-Tree with Genetic Algorithm. O-Tree can transform a code into a placement in linear time to the number of modules. Moreover, Genetic Algorithm is effective for searching a good layout because it can search two or more layouts concurrently. In experiment, we have confirmed to obtain nearly optimal packing results efficiently.

1. Introduction

The development of advanced VLSIs is indispensable to realize the high function of electronics. In the design of the LIS, many methods have been developed to minimize area and wiring-length for various module forms for the packing problem of functional block. Sequence-Pair(SP)[2] and Bounded-Sliceline-Grid(BSG) are the most major methods for the block packing ploblem. They can obtain excellent packing results using the module name codes for all solutions. However, as for them, long time is necessary for conversion to a packing with a code. Furthermore, the solution search by the SA that they adopt causes the increase of processing time. In this paper, we propose a block packing method using O-Tree with Genetic Algorithm(GA) for to search for the efficiently most suitable solution.

O-Tree expresses a module placement with a code like SP. It can transform a code into a placement in linear time to the number of modules. Therefore, reduction of computation time is expected. In using O-Tree, the one of most conventional searches of the optimal solution is an exhaustive one[1]. Becouse it also needs long processing time, an advantage which O-Tree can convert a code into a placement efficiently is not made use of. In contrast, the proposed method searches the optimal solution efficiently by GA. GA can search for two or more placement simultaneously. Therefore, the proposed method can obtain a better placement in short time.

2. Packing Problem

Given rectangular modules with any height and width, packing problem is determed to place them not to overlap and to become minimum area. Figure 1 shows the block packing. Figure 1(b) is the packing result for the modules of Fig. 1(a). They are delivered to the inside of the square of 12 every direction.

3. O-Tree

O-Tree is method to express packing by multi-way tree structure with a root. In O-Tree, all nodes except the root correspond to modules. The root is defined as module whose width is zero and height is large enough. Figure 2 (a) shows a multi-way tree structure, and (b) shows packing corresponds to this tree.

Generally, a multi-way tree of O-Tree describes the root to left side, and a child-node to right side of its parent-node. The positional relationship of the nodes is important in O-Tree. By searching order of nodes, O-Tree determines the positional relationship of each modules. Searching nodes uses Depth-First Search(DFS). DFS searches each nodes in deep order.

A packing of O-Tree is given two coordinate restrictions.

x: Module p's x coordinate of the left side is equal to its parent's x coordinate of the right side.

y: Module p's y coordinate of the lower side is equal to q's y coordinate of upper side. q is module of shallow than p in DFS order and having range which overlaps with p.

O-Tree has two characteristics [1].

1. Necessary time to obtain a layout from a O-Tree code is linear time to the number of modules.
2. O-Tree can express any placement on left-bottom compaction.

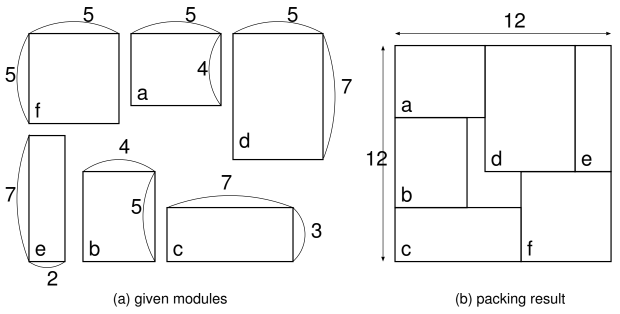


Figure 1. Packing Problem.

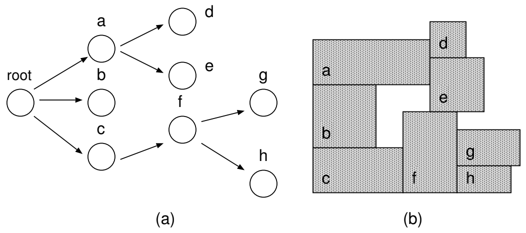


Figure 2. An O-Tree and its layout.

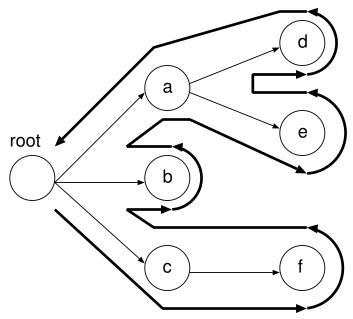


Figure 3. Generate of O-Tree Code.

3.1 Coding of O-Tree

An O-Tree code consists of T and π. T is a bit string for expressing a position information. π is a permutation for expressing a place order. Figure 3 shows converting a tree structure into the corresponding O-Tree code. The coding process uses DFS. In DFS, "visiting node a" means "passing below node a." "Backing to the root from node a" means "passing above node a." Starting from the root, when the coding process visits node c first, the coding outputs "0" to T and "c" to π. Then, it visits node f and outputs "0" to T and "f" to π. On the way back to the root from nodes f and c, it outputs "11" to T. Finally, we can obtain code $(T, \pi) = (001101001011, cfbaed)$. This code can express an only tree structure.

An O-Tree code must satisfy the following constraints.

- The number of "0" is equal to "1".
- When the number of "0" and "1" is counted from left side, "1" doesn't exceed the number of "0" in any point.
- A module name only appears once in π.

3.2 Admissible O-Tree

When the layout of obtaining from an O-Tree code is left-bottom compaction, it is called Admissible O-Tree(AOT). Otherwise, it is called Not Admissible O-Tree(NAOT). Figure 4 shows them. In Figure 4(b), module "c" and "d" are not packed state. By packing left-bottom compaction, the layout becomes AOT like Figure 4(a).

When we generate an O-Tree code randomly, it may become NAOT. The NAOT code is not assessed accurately. Therefore, whenever an O-Tree code updates, it is necessary to pack left-bottom compaction.

4. Proposal Technique

Like [1], our method converts from a O-Tree code into a packing. Because of this, this section describes about an application approach of GA to the proposed method.

4.1 Using Genetic Algorithm

In our approach, an O-Tree code is defined as one individual. They are sorted into the small order of cost. Our cost function is shown in the equation (1).

$$COST = \frac{S}{S_{min}} * \alpha + \frac{W}{W_{best}} * \beta + \frac{l_{long}}{l_{short}} * \gamma \quad (1)$$

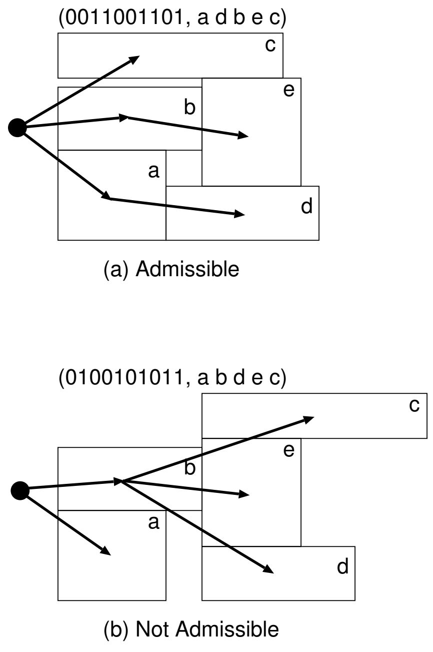


Figure 4. AOT and NAOT.

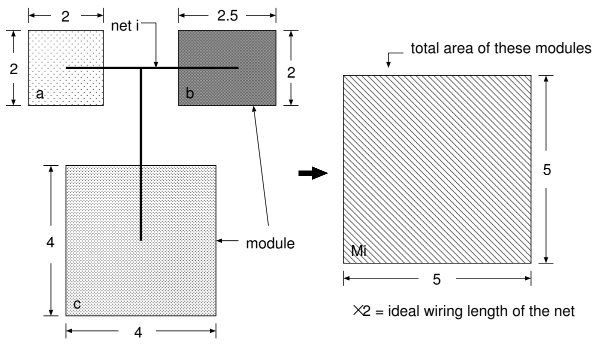

2
net i
2.5
2
a
b
2
4
module
c
4
total area of these modules
5
Mi
5
×2 = ideal wiring length of the net

Figure 5. W_{best}.

where S is the minimum rectangle area, S_{min} is the sum of area of all modules, W is total wire length, W_{best} is best wire length, l_{long} is the long side of the minimum rectangle area, l_{short} is the short side of it, and α, β, and γ is respectively the weight to the area, the wiring, and the aspect ratio. M_i is a square with the area that is equal tp the sum area of modules connecting with net i. W_i is the double length of one side of length of square M_i. With variable W_i, W_{best} is expressed by the equation (2).

$$W_{best} = \sum_{i=1}^{n} W_i \quad (2)$$

In Fig. 5, the area of "a", "b" and "c" are 4, 5 and 16 respectively. Consequently, the area of M_i and W_i become 25 and 10. l_{long}/l_{short} is called aspect ratio. Using l_{long}/l_{short}, solutions that are too long vertically or horizontally are hard to be selected to parents.

The selection of the parent uses an Elite Choice Method and a Roulette Choice Method based on a sorting result. A new individual is produced from the lowest-cost parents.

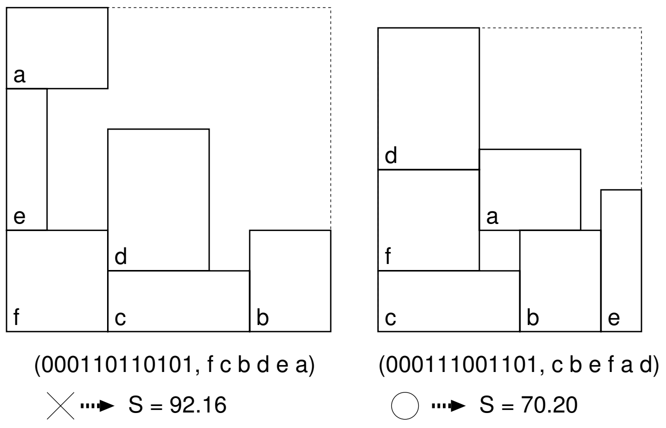


Figure 6. Choice Parent.

Therefore, there is a high possibility that the new individual is generated having excellent area and wiring. Figure 6 shows an example of parent selection. When these are same cost of wiring, the probability that the right side layout is chosen is high.

4.2 Crossover

Crossover uses respectively different method for T and π. Making a next generation needs to satisfy restrictions for crossover. Therefore, our method uses Subtree Crossover Method for T, and Order Crossover for π. The Subtree Crossover Method selects a subtree S_1 randomly from the parent P_1. Similarly, it selects a subtree S_2 also from the parent P_2. When the number of nodes of S_2 is less than S_1, it selects a subtree S_{22} from P_2, and updates S_2 in sum of S_2 and S_{22}. It repeats the selecting operation till the number of nodes of S_1 accords with S_2. Then S_1 and S_2 are replaced and inherited to children. Remainder nodes are inherited to children directly. By replacing subtrees, crossover can satisfy the restrictions.

Figure 7 shows example of Subtree Crossover. S_1 is 000110100111, S_2 is 001100010111. The generated children by replacing their genes are C_1 and C_2 respectively.

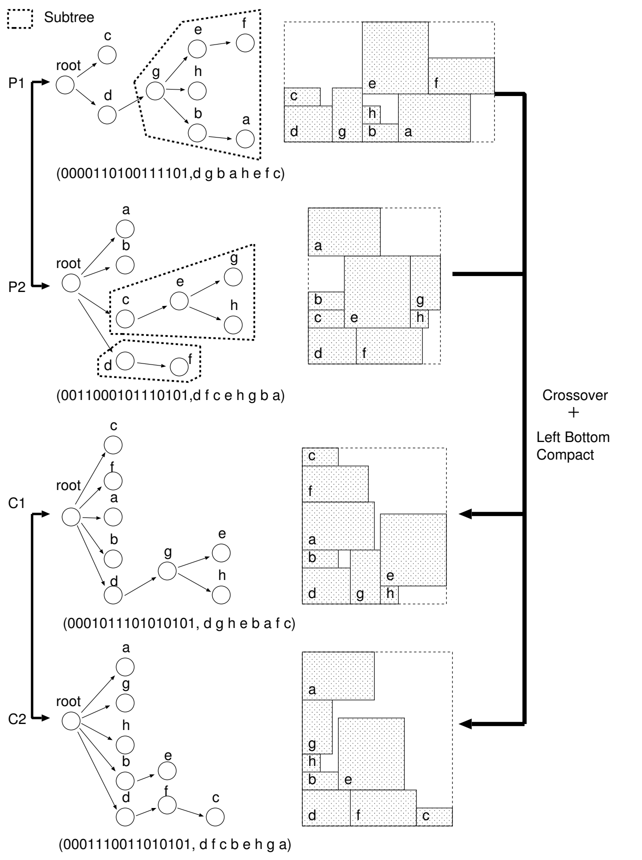
4.3 Mutation

Mutation replaces some modules in π. Figure 8 shows example of mutation. Our mutation selects "a", "b", and "e", then reolaces them in the direction of dotted arrows. As the result, "$fcbdea$" becomes "$fcedab$". Thus like this, the solution may be improved by the mutation. The mutation may prevent lapsing into the partial optimal solution.

4.4 Processing procedure

This subsection describes our packing process with GA.

1. Make plual O-Tree codes arbitrarily for a given netlist.
2. Be the initial individuals that every module is moved to left-bottom of packing space as much as possible.
3. Sort the individuals in order to low cost.
4. Select some pairs of individuals among the low cost as parents group sequentially.



Subtree
P1
root
(0000110100111101,d g b a h e f c)
P2
root
(0011000101110101,d f c e h g b a)
Crossover + Left Bottom Compact
C1
root
(0001011101010101, d g h e b a f c)
C2
root
(0001110011010101, d f c b e h g a)

Figure 7. Subtree Crossover.

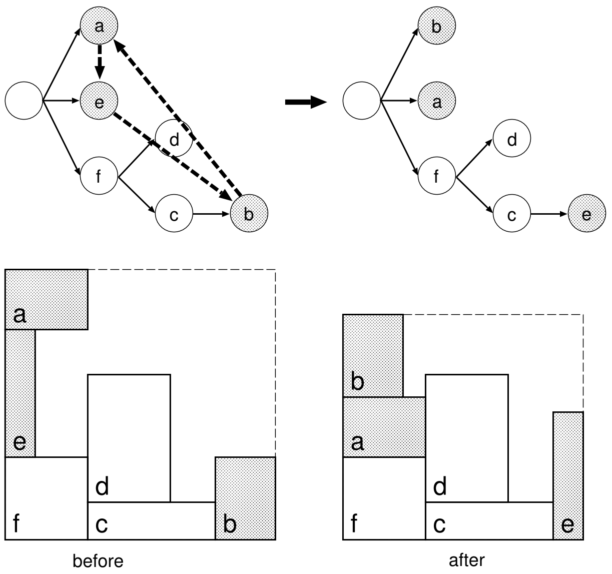


Figure 8. Mutation.

5. Generate new generation by applying crossover and mutation to every parents.
6. Pack new individuals to left-bottom, and evaluate cost.
7. Repeat from 3 to 6 till the number of generations reaches the limitation.

Table 1. Experiment datas and GA parameters.

number of modules	50
sum area of all modules	553
number of nets	500
mutation rate	0.03

Table 2. Experimental results(min/avg).

	DA[1]	**Ours**	**initial generate**
S_{min}	600	624	950
W_{sum}	11139	11217	15777
cost	137.70	140.66	203.70
t_{avg}	429s	181s	-

If the best solution are not improved until 500 generations successively, the GA finishes. We process this 10 times. Our packing method can search for the optimal packing result efficiently by utilizing the abilitu of the many points search of GA.

5. Experiment

We experimented in order to confirm availability of a floorplanning by O-Tree with GA.

In our experiment, initial solutions are generated randomly. Table 1 shows GA parameter. In this experiment, α, β, and γ are respectively 45, 45, and 10. Using this parameter, we searched an optimal solution 10 times.

6. Result

Table 2 shows the averages of conventional method(DA[1]) and the GA method and the minimum value of them. In Table 2, S_{min} is minimum rectangle area, W_{sum} is total wire length, t_{avg} is processing time average. In cost evaluation the conventional method is better 2.1% than our proposal method. However, processing time has improved 57.7%. We have obtained nearly optimal solution in short time.

7. Conclusion

We have confirmed to obtain nearly optimal packing results efficiently by using O-Tree with GA. Our method is effective for obtaining an optimal solution of large-scale layout.

References

[1] Pei-Ning Guo, Chung-Kuan Cheng, Takeshi Yoshimura, Toshihiko Takahashi, “Floorplanning Using a Tree Representation,” *IEEE Trans. CAD*, Vol.20, No.2, pp.281–289, 2001.

[2] Toshihiko Takahashi, “An Algorithm for Finding a Maximum-Weight Descreasing Sequence in a Permutaion, Motivated by Rectangle Packing Problem,” *IEICE*, pp.31–35, 1996.