An Enhanced Perturbing Algorithm for Floorplan Design Using the O-tree Representation*

Yingxin Pang
Dept. of CSE
Univ. of California, San Diego
La Jolla, CA 92093
ypang@cs.ucsd.edu

Chung-Kuan Cheng
Dept. of CSE
Univ. of California, San Diego
La Jolla, CA 92093
kuan@cs.ucsd.edu

Takeshi Yoshimura
NEC Corp.
4-1-1 Miyazaki, Miyanae-Ku
Kawasaki 216, Japan
yoshi@ccm.cl.nec.co.jp

ABSTRACT

Recently, a deterministic algorithm based on the O-tree representation has been proposed. This method generates excellent layout results on MCNC test cases with $O(n^3)$ complexity, where n is the number of blocks. In this paper, we reduce the complexity of the deterministic algorithm to $O(n^2)$. Experimental results indicate our algorithm maintains the high quality of the deterministic algorithm at a fraction of the CPU time.

1. INTRODUCTION

One of the most important steps of VLSI circuit layout is the floorplan design. The goal is to arrange a set of non-overlapping rectangular modules so that a certain objective function is minimized. The objective function could be the total area of the floorplan, the total wire length, or a weighted sum of these two quantities.

Topological representations including both slicing structure, [5,6], and nonslicing structures [3,4] have been very popular in recent years. More recently, a nonslicing representation called the O-tree[2] has been proposed. Compared with previous representations, the O-tree has a smaller upper bound on the number of possible configurations, needs only linear computation effort to generate a corresponding placement, and decreases the drawback of redundancies in the previous representations.

We consider only the O-tree representation here as we have pointed out such a structure has important advantages. The deterministic algorithm [2] based on the O-tree representation can be used to improve an existing floorplan. For an initial O-tree with n blocks, the deterministic algorithm iteratively perturbs each block, and, for each block, tries all possible insertion positions in the tree. For each insertion position, it constructs a corresponding placement. Because the evaluation of an O-tree takes $O(n)$ operations and the number of possible insertion positions is $O(n)$, it takes $O(n^2)$ for one block. Therefore, the placement of n blocks takes $O(n^3)$.

*This work was supported in part by grants from NSF Project MIP-9529077 and the California MICRO program

ISPD 2000, San Diego, CA.

In this paper, we describe an enhanced perturbing algorithm (ENPA) which achieves the same goal of the deterministic algorithm with less complexity. Like the deterministic algorithm, it iteratively perturbs each block. For each block, however it follows a depth first search sequence, reducing the checking of each position to constant time by amortizing the checking of other positions in that sequence. ENPA finds the relatively best insertion position without constructing the O-trees. Therefore ENPA takes only $O(n)$ for each block and runs in $O(n^2)$ for n blocks. ENPA can be used to improve an existing floorplan, or carry out a complete design of a floorplan. To demonstrate the efficiency of ENPA, we apply it to five MCNC benchmarks and compare it with the deterministic algorithm. The results show that we produce comparable high quality results with significantly less CPU time.

The rest of this paper is structured as follows: in section 2, we give a brief review of the O-tree representation and the deterministic algorithm. In section 3, we present our enhanced perturbing algorithm (ENPA) with the objective function restricted to be the total area. In section 4, we extend the objective function to consider the wire length. In section 5, we illustrate how to carry out the complete floorplan design by ENPA. In section 6, the experimental results are shown. Concluding remarks are in section 7.

2. BRIEF REVIEW OF THE O-TREE STRUCTURE AND THE DETERMINISTIC ALGORITHM

2.1 The O-tree Representation

A n-node O-tree is a tree with $n+1$ nodes and is encoded by (T, π), where T is a $2n$-bit string that identifies the branching structure of the tree, and π is a permutation of the n node labels (excluding the root). When traversing the tree, we write a '*0*' for descending an edge and a '*1*' for subsequently ascending that edge. Given the *6*- node O-tree in Fig. 1, we can represent it as (T= *001100011101*, π=*bcdafe*). Let B_1, B_2, ... , B_n be the blocks to be placed on a chip, where each B_i is a rectangle having associated with it a width w_i and a height h_i, and having (x_i, y_i) as coordinates of its left-bottom corner. To construct a minimum area placement for an O-tree, we place the blocks in depth first search order, and the positions of the blocks are determined by the following conventions:

(1) if B_i is the parent of B_j, then $x_j=x_i+w_i$. The root may be viewed as the left chip boundary, $x_{root}=0$ and $w_{root}=0$.

(2) for each block B_i, let $\psi(i)$ be the set of blocks B_k which have

been placed on the chip and whose spanning interval $(x_k, x_k + w_k)$ overlaps $(x_i, x_i + w_i)$. Then

$y_i = max_{k \text{ in } \psi(i)} y_k + h_k$ if $\psi(i)$ is non-empty

$=0$ otherwise

Fig. 1 shows an O-tree and its corresponding placement.

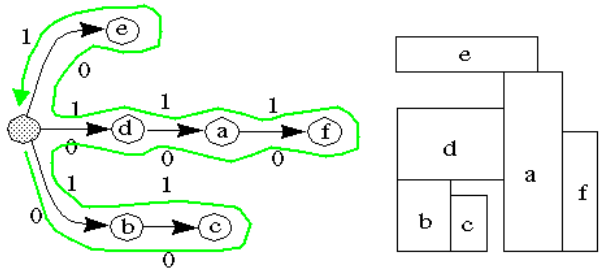

Fig. 1 An O-tree and its placement

2.2 The Deterministic Algorithm

Given an O-tree, a perturbation to that configuration can be made by deleting a block from the O-tree and inserting it in other positions of the O-tree. For simplicity, we constrain our inserting positions to be the external nodes of the tree. If an initial O-tree has *n* blocks, then after deleting the perturbed block, the number of the possible insertion positions as external nodes is *2n-1* (it may include the previous position of the block). Fig. 2 shows the resulting O-tree obtained by deleting block *a* from the original O-tree shown in Fig. 1 and all the possible insertion positions.

The deterministic algorithm attempts to reconstruct the O-tree by placing the deleted block in the position so that the corresponding placement has the minimal cost. Therefore an original O-tree is iteratively improved by perturbation.

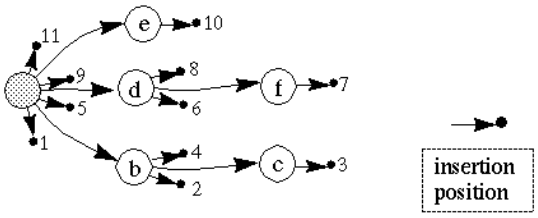

Fig. 2 Possible insertion positions

The basic procedure of the **Deterministic Algorithm** is as follows:

given an initial O-Tree OT
for *each block b in OT*
  set *min_cost = infinite*
  remove b from OT, a resulting O-tree OT_1 is obtained
  for *each possible position p in OT_1 for b*
    set *OT_2 = new O-Tree inserting block b in position p*
    build the placement of OT_2
    set *c = cost (OT_2)*
    if c < min_cost then
      set *min_cost = c*
      set *min_OT = OT_2*
    end if
  end for
  set *OT = min(OT, min_OT)*
end for
Output placement for OT

There are two "**for**" loops in the algorithm: the first iterates over all blocks, the second iterates over all insertion positions. For each inserting position, it builds an O-tree and its corresponding placement. Since O-tree evaluation can be performed in linear time, if there are *n* blocks to be placed on a chip, the complexity of the deterministic algorithm is $O(n^3)$.

3. THE ENHANCED PERTURBING ALGORITHM

In the deterministic algorithm, there are *2n-1* insertion positions for each perturbed block. Among them there exists one position corresponding to a placement minimizing the value of the cost function. It is desirable to determine the best one without constructing all possible new placements corresponding to the inserting positions. Based on this observation, in this section we present an enhanced perturbing algorithm (ENPA) which follows a depth first search order sequence, reducing the time of checking each inserting position to constant time by amortizing the checking of other positions in that sequence. It can achieve the goal of the deterministic algorithm in much less time.

If we visit the nodes of the O-tree shown in Fig. 2 in depth first search order, the numeric labels next to the insertion position indicate the order in which we visit the inserting positions. At each insertion position, we virtually place that deleted block, and attempt to obtain the value of the cost function without reconstructing the whole placement. In order to make things clear, we restrict the objective function to be the total area of the floorplan in this section. Our algorithm could be extended to minimize a different objective function, such as the weighted sum of the area and the wire length. This extension is shown in next section.

3.1 Outline of the Enhanced Perturbing Algorithm

Suppose all blocks are to be placed within a bounding box. The box size is *(H,W)*. *W* is the width of the box, *H* is the height of the box. We refer to the top part of the box as the *ceiling*, and the bottom part of the box as the *floor*. We define *Gap* as the minimum distance of the empty space between the ceiling and the floor. *Width* is the right boundary of all blocks.

Given an original O-tree, we remove the perturbed block from the O-tree and perform the depth first search for the resulting O-tree. We evaluate all insertion positions during this search. At each possible insertion position, we virtually place the perturbed block and estimate the value of the objective function. In order to prevent the virtually placed block from overlapping with other blocks, we arrange some blocks on the ceiling and some blocks on the floor to make sure that there is enough space for the perturbed block to be virtually inserted. The best result among all possible insertion positions is chosen as the initial for the next iteration.

The outline of the enhanced perturbed algorithm is as follows:

Procedure *The enhanced perturbing algorithm*

Input: An initial O-tree OT

Output: A reconstructed O-tree OT

for each block u in OT do the following 5 steps:

Step 1: Delete block u from OT and a resulting O-tree $OT_1(T_1, \pi_1)$ *is obtained*

Step 2: Place all blocks in OT_1 *on the floor.*

Step 3: Slide all blocks up toward to the ceiling.

Step 4: current_block=root

bestcost=infinite

// insert a at the first insertion position

$x_{root}=y_{root}=0$, $x_u=y_u=0$

do step 4.3

index=1

for $T_1[i]$, *i=1,2,...,2n, repeat Step 4.1 through Step 4.3*

Step 4.1: if $T_1[i]$ *is 0*

peel block π_1 *[index] down to the floor*

update the contour structure of the ceiling and floor

current_block=π_1 *[index]*

index=index+1

else current_block=parent of current_block

Step 4.2: // virtually place u at position (x_u,y_u)

$x_u=x_{current\_block}+w_{current\_block}$

let ψ *be the set of blocks which are on the floor and their horizontal spanning intervals overlap with* (x_u, x_u+w_u),

$y_u = max_{k\ in}\psi\ (y_k + h_k)$

Step 4.3: // evaluate this insertion position:

let ψ *be the set of blocks which are on the ceiling and horizontal spanning intervals overlap with* (x_u, x_u+w_u)

$G_u = min_{k\ in}\psi\ (y_k - (y_u+h_u))$

newGap=min(Gap, G_u*)*

newWidth=min(Gap, x_u+w_u*)*

newArea=newWidth(H-newGap)*

cost=newArea

if (cost < bestcost)

bestcost=cost

best_insertion_position=this insertion position

Step 5: Insert the deleted block in the best_insertion_position, construct the corresponding O-tree OT_{new}

$OT=min(OT, OT_{new})$

3.2 An Illustrative Example

A behavioral example is presented in this section to illustrate the enhanced perturbing algorithm.

Suppose the O-tree *OT (001100011101, bcdafe*) shown in Fig. 1 is given as the input O-tree and block *a* is selected to be perturbed. In step 1, block *a* has to be deleted from its original position in *OT*. The resultant O-tree OT_1 *(0011001101, bcdfe)* is shown in Fig. 2. We will traverse OT_1 in depth first search order, and visit the inserting positions in the order listed in Fig. 2.

In step 2 and step 3, the placement of OT_1 is pushed up to the ceiling of the bounding box. Fig. 3 shows the result of the two steps.

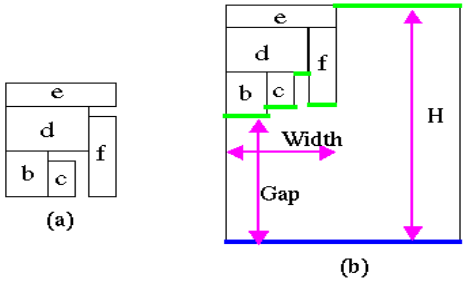

Fig. 3 Step 2 and Step 3

In step 4, we first virtually insert *a* at the insertion position 1 as shown in Fig. 4. After *a* is inserted, *newGap* is y_b-h_a , *newWidth* is the same as *Width*, the total area of the placement is *(H-*(y_b-h_a*))*Width*. We update the *best cost* to be *(H-*(y_b-h_a*))*Width*, and *best insertion position* to be *1*.

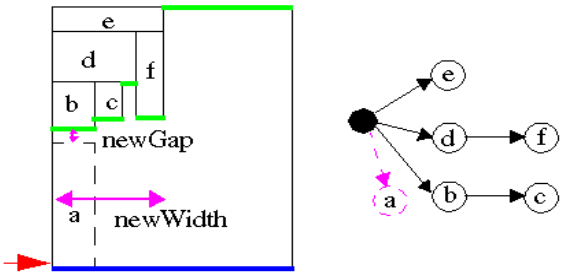

Fig. 4 Insert a in insertion position 1

Then we traverse OT_1 in depth first search order. In iteration *i=1*: $T_1[1]=0$, $\pi_1[1]=b$, we peel block *b* down to the floor and virtually place *a* right next to block *b* in the insertion position 2 shown in Fig. 5. Therefore the value of the cost function which is also the total area is *(H-*(y_c-h_a*))*Width*. Since *best cost* is larger than current *cost*, we update the *best cost* and the *best insertion position*.

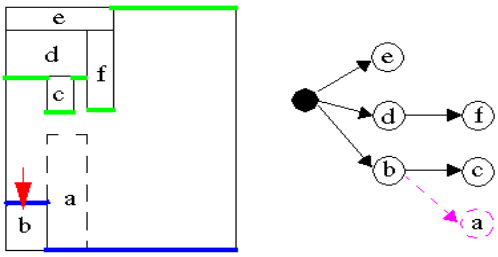

Fig. 5 Insert a in insertion position 2

In iteration *i=2*: $T_1[1]=0$, $\pi_1[2]=c$, we peel *c* down to the floor. We virtually place *a* right next to block *c* in the insertion position 3 shown in Fig. 6. Now the *cost is (H-*(y_f-h_a*))*width*. Since this *cost* is smaller than the *best cost*, we update the best cost, and the current best insertion position is position 3.

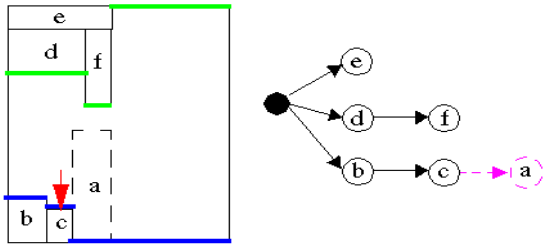

Fig. 6 Insert a in insertion position 3

In iteration *i=3*: $T_1[3]=1$, we update the current block to be block *b* (parent of block *c*) and insert *a* at the insertion position 4 shown in Fig. 7. Since *best cost* is smaller than this current *cost*, no update is needed.

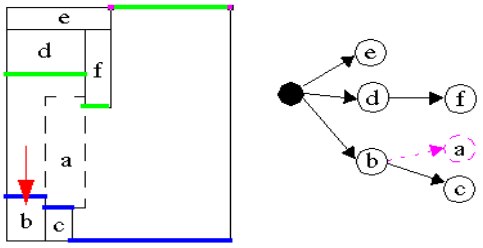

Fig. 7 Insert a in insertion position 4

In iteration *i=4*: $T_1[4]=1$, we update the current block to be the root which represents the left boundary of the bounding box. We insert block *a* in the fifty insertion position. Since the current cost is larger than best cost, no update is needed.

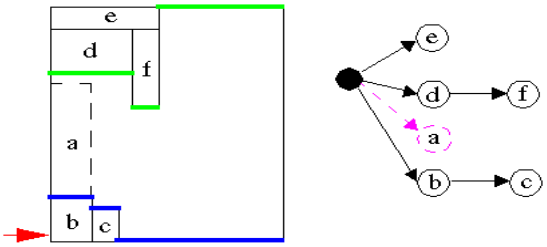

Fig. 8 Insert a in insertion position 5

We continue this process. When the code is '0', we peel down the corresponding block, update the contour structure of both the ceiling and the floor, and update the *current block* to be the block just peeled down. Then we insert block *a* next to the *current block*. When the code is '1', we update the *current block* to be the parent of the previous *current block*, and then insert block *a* next to the *current block*. In both cases, we calculate the new area and update the best cost and the best insertion position when it is necessary. After we traverse all the blocks, we visit all insertion positions, and the best insertion position is the one corresponding to the minimum area. We note that more than one position may lead to the same minimum area. In our example, position 7 is one of them. We insert block *a* in position 7. The resulting O-tree and its placement are shown in Fig. 9.

After block *a* is perturbed, the tree then is reconstructed as in Fig. 9. Repeat the procedure until all blocks are perturbed.

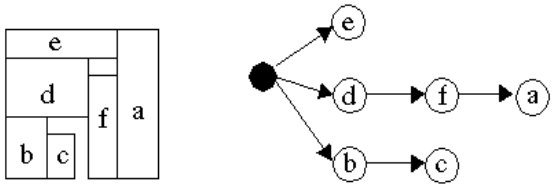

Fig. 9 The reconstructed O-tree and its placement

3.3 The Complexity of the Enhanced Perturbing Algorithm

Theorem 1: The enhanced perturbing algorithm runs in $O(n^2)$, where *n* is the number of the blocks of the O-tree.

Proof: The enhanced perturbing algorithm iterates over all blocks of the given O-tree, so we need to prove that for each iteration the complexity of the algorithm is *O(n)*.

Step 1 takes at most *n* operations. Steps 2 and 3 can be achieved by the following procedure -- *PushToCeiling*. We will explain that it is *O(n)* later in this subsection.

In step 4, we visit the nodes in depth first search order, so each node is visited exactly twice: once at the nodes encoded '*0*' and once at the nodes encoded '*1*'. This loop executes exactly *2n* times. Inside the loop, we perform four major operations: (1) peel down a block from the ceiling (2) find the local maximum of y-coordinate of the floor (3) find the local minimum of y-coordinate of the ceiling (4) update the contours of the ceiling and the floor. We only need to pass a limited set of blocks on the above four operations. The number of the blocks accessed is equal to the number of edges inserted in the vertical constraint graph. The constraint graph is planar, and the number of edges in the vertical constraint graph is *O(n)*. In the whole loop, we go through every edge in the vertical constraint graph exactly twice. The overall complexity for step 4 is then linear. Therefore the complexity for the enhanced perturbing algorithm is $O(n^2)$.

We apply the following algorithm *PushToCeiling* to step 3. The basic idea of this algorithm is to transform the given O-tree to another O-tree structure whose siblings are in the reverse order of the original O-tree, and then place the new structured O-tree on the ceiling.

Procedure *PushToCeiling*

Input: O-tree OTd

Output: the vertical constraint graph and the placement of blocks in OTd on the ceiling.

```
perm=1
current=node_head=NULL
for i=1 to 2n
    new_T[2n-i]=1-T[i]   // put the siblings in reverse order
    if (T[i]==0)
        put the π [perm] after the current block
        perm=perm+1
    else  current=prev(current)
```

//write new permutation in the correct corresponding order

current=node_head

for i=1 to n

new_π [i]=current

current=next(current)

place O-tree(new_T, new_π) on the ceiling

Observation: Algorithm *PushToCeiling* runs in linear time since transforming an O-tree to another O-tree with reverse order sibling structure takes linear time, and evaluating an O-tree to the placement is in linear time.

4. EXTENDED COST FUNCTION

In physical design, wires are usually considered in addition to the area in the objective function. We extend our objective function to be the weighted sum of the area and wire length in this section. Wire length is defined by the perimeter of the minimum bounding box.

At each insertion position, the area of the corresponding placement can be obtained by the method described in the above section. In order to save the computational cost, we only consider the nets related to the inserted block for the wire length. We use the same notation here as in the procedure for the enhanced perturbing algorithm. OT_1 is the resulting O-tree obtained by removing block *u* from the O-tree *OT*. At insertion position *i*, the block *u* is virtually placed in (x_u, y_u). The wire length of the corresponding placement can be approximated by the following steps:

for each pin_i of block u

$x=x_u$+relative x-coordinate of pin_i

$y=y_u$+relative y-coordinate of pin_i

net_i is the net that pin_i need to be connected

(x_1,x_2, y_1, y_2) is the boundary of net_i without block u in the placement of OT_1

if $x<x_1$ $x_1=x$ if $x>x_2$ $x_2=x$ if $y<y_1$ $y_1=y$ if $y>y_2$ $y_2=y$

change_wire=the wire length changed by the above loop

newWirelength=change_wire+wirelength(OT_1)

*cost=w_1*newArea+w_2*newWirelength*

5. COMPLETE DESIGN OF A FLOORPLAN

In addition to improving an existing floorplan, the idea of the enhanced perturbing algorithm can be used to carry out the complete design of a floorplan. Given a random permutation of the labels of the blocks, we can heuristically construct a floorplan in the following way:

Procedure *Complete floorplan design*

Input: A random sequence π from 1 to n

Output: A floorplan of block 1 to block n

construct an O-tree OT with empty nodes

for π [1] to π [n]

evaluate all possible inserting position in OT

construct a new O-tree new_OT by inserting π [i] in the best insertion position

OT=new_OT

run the enhanced perturbing algorithm to OT

output the placement of OT

By using the order of the given random sequence, we insert the blocks to the constructed tree in the best position each time. We then improve the already constructed O-tree by running the enhanced perturbing algorithm. At each step we perform a greedy strategy. This process continues until that a complete pass through all the blocks produces no improvement. If the orientation of the blocks is flexible, we can also evaluate the different orientations in each insertion position and choose the best orientation each time.

6. EXPERIMENTAL RESULTS

We have implemented our algorithm in C on a Sun Ultra60 workstation. The test cases are the five MCNC benchmarks. We compared our algorithm with the deterministic algorithm. The experimental results are reported in Tables 1, 2 and 3. All the results have been run on the same Sun workstation. The cost function we use here is w_1*area+w_2*wirelength. Tables 1, 2 and 3 correspond to the following sets of $\{w_1, w_2\}$ values: {0,1}, {1.0} and {0.5, 0.5}. DA denotes the deterministic algorithm and ENPA denotes our algorithm.

The initial sequences of the problems in the tables are randomly generated. In each test case, we use 100 randomized sequences and present the best results among the 100 runs. The running time of our algorithm on each of the test examples is substantially less than that of the deterministic algorithm. For circuit *apte* (which has only 9 cells and is the smallest circuit among the MCNC benchmarks), our algorithm is about 3 times faster than the deterministic algorithm under all three different cost functions. But for the largest circuit *ami49* with 49 cells, our algorithm is about 20 times faster on average. The larger the circuit, the greater the performance gain with our algorithm.

Table 1 is a summary of results for the case where the cost function is the total area. Among the five circuits, our algorithm has slightly better solutions than the deterministic algorithm in three of them. In the other two test examples, the areas of the placements by our algorithm are no more 0.55% larger than those by the deterministic algorithm. Therefore the two algorithms produce comparable results. In Tables 2 and 3, wire length has been considered in the cost function. Comparing with the deterministic algorithm, we have -2% to 2% improvement in area and in wire length. On average, the two algorithms generate the same high quality placements. Therefore our approximation strategy in section 4 works well.

For the cost function with weights 0.5 for both the area and wire length, we display two placements. The placement of *ami33* is presented in Fig. 10. This circuit has 33 cells, and the placement takes only 3.42 minutes. Fig. 11 shows the largest circuit *ami49* of the five examples. This placement was completed in about 12 minutes.

The above experimental results show clearly that the results obtained by the ENPA are as good as those of the deterministic algorithm, while taking much less time. Therefore, the ENPA is more effective.

7. CONCLUSIONS

We have proposed a new method utilizing the properties of an O-tree to improve an existing floorplan or carry out a complete design of a floorplan in faster time. Since the speed up is proportional to the size of the problem, it is especially useful for handling large circuits.

Table 1 cost function=area

Circuit	w1=1 w2=0			
	Time(min)		Area(mm*mm)	
	ENPA	DA	ENPA	DA
apte	0.19	0.63	46.92	47.1
xerox	0.63	1.97	20.21	20.1
hp	0.32	0.95	9.159	9.21
ami33	1.98	23.83	1.242	1.25
ami49	6.76	123.5	37.73	37.6

Table 2 cost function=wirelength

Circuit	w1=0 w2=1			
	Time(min)		Wire(mm)	
	ENPA	DA	ENPA	DA
apte	0.25	0.79	316.8	317
xerox	0.65	2.66	372.2	368
hp	0.32	1.50	150.4	153
ami33	2.95	37.52	51.58	51.5
ami49	11.46	235.2	629.3	636

Table 3: cost function=0.5*area+0.5*wirelength

Circuit	w1=0.5 w2=0.5					
	Time(min)		Area(mm*mm)		Wire(mm)	
	ENPA	DA	ENPA	DA	ENPA	DA
apte	0.24	0.77	51.95	51.92	320.7	320.7
xerox	0.68	2.37	20.42	20.42	380.6	380.6
hp	0.35	1.40	9.384	9.490	151.9	152.6
ami33	3.42	39.15	1.299	1.283	52.13	51.31
ami49	11.67	255.3	39.92	39.55	702.8	688.7

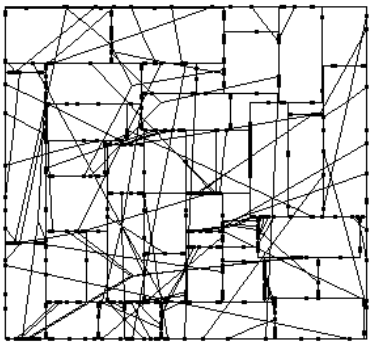

Fig. 10 Circuit ami33

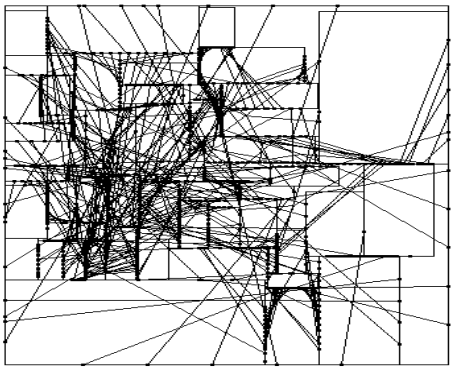

Fig. 11 Circuit ami49

8. REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, "Introduction to Algorithm," *The MIT press*, 1990.

[2] P.-N. Guo, C.-K. Cheng, T. Yoshimura, ''An O-tree representation of non-slicing floorplan and its applications,'' *Proc. 36th ACM/IEEE Design Automation Conf.* , pp. 268-273, June 1999.

[3] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair,'' *IEEE Trans. on Comp. Aided Design of IC's and Systems*, Vol. 15, No. 12, pp. 1518-1524, Dec. 1996.

[4] S. Nakatake, K. Fujiyoshi, H. Murata, Y. Kajitani, ''Module packing based on the BSG-structure and IC layout applications,'' *IEEE Trans. on Comp. Aided Design of IC's and Systems*, Vol.17, No.6, pp. 519-530, June 1998.

[5] R. Otten, ''Complexity and diversity in IC layout design" *Proc. IEEE Intn'l Symp. Circuits and Computers*, 1980.

[6] D. F. Wong, C. L. Liu, "A new algorithm for floorplan design," *Proc. 23th ACM/IEEE Design Automation Conf. pp. 101-107, June 1986*