Slicing Floorplan with Clustering Constraints

W.S. Yuen and F.Y. Young
Department of Computer Science and Engineering
The Chinese University of Hong Kong
wsyuen@cse.cuhk.edu.hk fyyoung@cse.cuhk.edu.hk

Abstract—In floorplan design it is useful to allow users to specify placement constraints in the final packing. Clustering constraint is one kind of placement constraint in which a given set of modules are restricted to be geometrically adjacent to one another. The wiring cost can be reduced by putting modules with a lot of connections closely together. Designers may also need this type of placement constraint to suit different functionality of the modules. In this paper, a method addressing clustering constraint in slicing floorplan is presented. A linear time algorithm is devised to locate neighboring modules in a normalized Polish expression and re-arrange the module in order to satisfy the constraints. Experiments were performed on some benchmarks and the results are promising.

I. Introduction

Floorplan design is the problem of planning the position and shapes of the modules at a very early designing stage in order to optimize the circuit performance after circuit partitioning. During this floorplanning phase, the circuit performance like layout area, interconnect cost, heat dissipation and power consumption, etc, should be minimized.

There are two types of floorplans: slicing and non-slicing. A slicing floorplan is one that can be obtained by recursively partitioning a rectangle into two parts by either a vertical line or a horizontal line. The advantage of using slicing floorplan is that it has simple representation such as slicing tree and Polish expression [1, 2]. A non-slicing floorplan is a floorplan which is not necessarily slicing. Several representations, sequence-pair [3], bound-sliceline-grid(BSG) [4], O-tree [5], B*-tree [6] and Corner Block List [7], have been proposed for solving the placement problem in non-slicing floorplan. There are many other floorplanners for fixed blocks or flexible blocks [8, 9] and many of them are based on a slicing floorplanner developed in [2].

There are several aspects that a floorplanning algorithm has to deal with: the shapes of the modules, routability, area and delays. All these are essential for optimization of the circuit performance. With the scaling down of the IC technology, the number of transistor that can be built into a standard size chip has increased rapidly. It is important to optimize the circuit performance in the early designing stage. Timing-driven floorplanner aims at handling this problem [10, 11]. [12] proposed a hybrid floorplanning method using partial clustering and module restructuring. This method allows clusters to be placed as rectangles only subtrees in a slicing tree. The moves in the annealing process are limited and the deadplace in the final packing is usually large.

Placement constraints in floorplan design are useful for specifying the placement relationship between the modules according to their functionality in order to improve the circuit performance like interconnect cost and delay, etc. Some previous work on placement constraints in slicing floorplan [13, 14] have been done. In this paper, clustering constraint is considered in which some modules are required to be placed next to each other. The cost of routing can be reduced by imposing clustering constraints to modules which are heavily connected. The method we used can determine the neighboring positions of a target module in a Polish expression and swap the constrained modules to these positions in order to satisfy the constraints. Our method can also be extended to handle more than one cluster in the floorplan.

This paper is organized as follows. Problem definition is given in section 2. In section 3, a slicing floorplanner which our method based on will be described. In section 4, detailed descriptions of the clustering method will be given. The results will be shown in section 5.

II. Problem Definition

A floorplan with n modules (1, 2, ..., n) is an enveloping rectangle R subdivided by horizontal and vertical line segments into n or more non-overlapping rectilinear regions such that each region R_i must be large enough to accommodate the corresponding module i.

In most iterative methods, a floorplan is evaluated by a function $A + \lambda W$, where A is the area and W is the wirelength. The overall aspect ratio of the floorplan is also required to be within a given range. The aspect ratio of each module will be limited so that the delay inside each module will not be too long. For each rectangular module i, there are three input values A_i, r_i and s_i. where A_i is the area of module, r_i and s_i are the minimum and maximum aspect ratio of the module respectively. Let

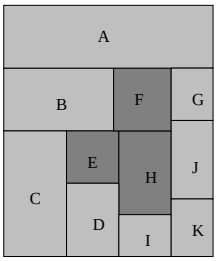

Fig. 1. An example of the clustering constraint

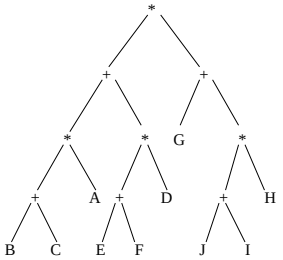

Fig. 2. Slicing Tree

w_i and h_i be the width and height of the module, then $A_i = w_i h_i$ and $r_i \leq \frac{h_i}{w_i} \leq s_i$.

In this paper, clustering constraint is considered in floorplan design. Given a set of modules Φ and a subset of modules $\Delta \subseteq \Phi$ we want to pack the modules in Φ such that the modules in Δ will be geometrically adjacent to each other. Figure 1 shows an example of the clustering constraint. Modules E, F and H are the subset of modules to be clustered and they have to be placed adjacent to each other in the final packing. The floorplanning problem with clustering constraints is defined as follows:

Problem FP/CC *Given a set of n modules $\Phi = \{m_1, m_2, \ldots, m_n\}$ and $m_i = (A_i, r_i, s_i)$ for $i = 1, \ldots n$ where A_i is the area of modules, r_i and s_i are the minimum and maximum aspect ratio of modules i respectively. Let Δ be a subset of modules in Φ, pack the modules in Φ to minimize the total area and interconnect cost such that the following three conditions are satisfied.*

1. *Every M_i in Δ should be geometrically adjacent to at least one $M_k \in \Delta$ where $k \neq i$.*
2. *Each module satisfies its area and aspect ratio constraint.*
3. *The aspect ratio of the whole packing is within a give range [r,s].*

III. Basic Slicing Floorplanning Algorithm

Our work is based on a well-known slicing floorplanner [2]. A slicing floorplan can be represented by a tree structure. Leaf nodes of the tree are the basic modules and the internal nodes are either labelled with + or *, known as operators, to represent a horizontal cut or a vertical cut. An example is shown in Figure 2. the tree in postorder, a Polish expression is produced which is used to represent the floorplan structure in the algorithm. A normalized Polish expression is a Polish expression with no identical operator between an internal node and its right child.

Simulated annealing is used to optimize the cost of the floorplan. The neighbors of each solution have to be defined such that the optimal solution is reachable. Three moves M1, M2 and M3 are used. M1 swaps two adjacent operands, M2 interchanges the operators in a chain (a chain is a substring in the expression with consecutive operators) and M3 swaps two adjacent operand and operator.

IV. Floorplanning with Clustering Constraint

In this paper, we consider clustering constraint in slicing floorplan. One method to solve this problem is by adding a term to the cost function of the annealing process as a penalty for violating the constraints. This method was tested but the result is poor and the constraints will usually be violated in the final packing. A better approach will be introduced in which clusters are maintained throughout the annealing process.

A method is devised to locate all neighbors of a target module. A target module M_t is picked from the subset Δ. A set of M_t's neighboring modules Π_t is obtained by the algorithm. For each $M_i \in \Pi_t$, if $M_i \notin \Delta$, we will swap M_i with M_j where $M_j \in \Delta \backslash \Pi_t$. This algorithm maintains the clustering constraint throughout the annealing process.

An overview of the algorithm is given as follows:

```
Main Program
begin
    While T ≥ threshold do
    begin
        Move by either M1, M2 or M3
        Call procedure Clustering
        Compute Cost
        If Cost is reduced
            Accept the move
        Else
            Prob = min(1, e^{-Δc/T}) where Δc = change of cost.
            If random(0,1) ≤ Prob then
                Accept the move
            Else
                Reject the move
    end
end
```

A. Locating Neighboring Modules

An algorithm is devised to locate all neighboring modules of a target module in the normalized Polish expression. Note that we can locate the neighbors in linear time by just looking at the Polish expression and no real packing is needed.

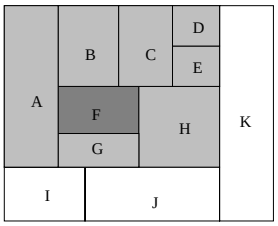

Fig. 3. An example of the neighboring structure

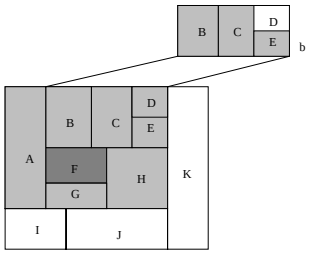

Fig. 5. D does not belong to Π_t where M_t is F

A target module $M_t \in \Delta$ is selected. A set Π_t is found such that M_t is surrounded by the modules in Π_t in the packing. An example is shown in figure 3. In this example $M_t = F$ and $\Pi_t = \{A, B, C, D, E, G, H\}$. Note that the modules found (e.g. D and E) may not be next to M_t.

For each module M_i in the slicing floorplan, M_i is surrounded by four cuts which correspond to four operators in the normalized Polish expression. If those four operators are found in the Polish expression, the neighboring structure can be located and Π_t can be found.

For a Polish expression $\alpha = \alpha_1\alpha_2\ldots\alpha_n$, we define a *valid sub-expression* $\beta = \alpha_k\alpha_{k+1}\ldots\alpha_{k+m}$ where $k \geq 1$ and $n \geq k+m$ as a sub-expression in α such that α_k must be an operand and the number of operands is equal to the number of operators plus one in β. A valid sub-expression indeed represents a sub-tree in the whole slicing tree.

The two operators correspond to cuts of different orientations. Let ζ, δ and γ be valid sub-expressions in the Polish expression. Some terms are defined as follows:

Below : If $\gamma = \zeta\delta+$, $Below(\delta, \zeta)$
Above : If $\gamma = \delta\zeta+$, $Above(\delta, \zeta)$
Left : If $\gamma = \zeta\delta*$, $Left(\delta, \zeta)$
Right : If $\gamma = \delta\zeta*$, $Right(\delta, \zeta)$

Given a target module M_t, the algorithm *Find_Surrounding* finds four valid sub-expressions a, b, c and d such that $Below(\delta_1, a)$, $Above(\delta_2, b)$, $Left(\delta_3, c)$ and $Right(\delta_4, d)$ where $\delta_i's$ are valid sub-expressions containing M_t.

```
Algorithm: Find_Surrounding(M_t, α)
Input:α = α_1 α_2 ... α_n is a Polish
      expression of the original packing.
      t is the index of the target module,
      i.e., α_t = M_t
Output: a is a valid sub-expression such that Below(δ_1, a)
        b is a valid sub-expression such that Above(δ_2, b)
        c is a valid sub-expression such that Left(δ_3, c)
        d is a valid sub-expression such that Right(δ_4, d)
        where δ_i for i = 1...4 is the shortest valid
        sub-expression containing M_t
        such that the a, b, c and d above can be found.
1 first = end = t
2 While a, b, c, d are not found and first≥1 and end≤ 2n − 1
3  begin
4     If α_{end+1} is an operator
5     begin
6        Find k such that
7          e = α_{first−k} α_{first−k+1} ... α_{first−1}
8        is the shortest valid sub-expression
9        If α_{end+1} is + and a is not found yet
10          a = e
11       Else if α_{end+1} is *  and c is not found yet
12          c = e
13       first = first − k; end = end + 1
14    end
15    Else
16    begin
17       Find k such that
18          e = α_{end+1} α_{end+2} ... α_{end+k}
19       is the shortest valid sub-expression
20       If α_{end+k+1} is + and b is not found yet
21          b = e
22       Else if α_{end+k+1} is * and d is not found yet
23          d = e
24       end = end + k + 1
25    end
26 end
```

The complexity of this algorithm is O(n). Figure 4 illustrates the steps of the algorithm. a, b, c and d are valid sub-expressions representing a sub-tree in the slicing tree. For the example in Figure 3, $a = G, b = BC*ED+*, c = A, d = H$ and $M_t = F$. The shortest sub-expression can be obtained by counting the number of operators and operands. Note that not all the basic modules in a, b, c and d belong to the neighboring module set Π_t of M_t. If a sub-expression b is above M_t, then only the modules at the bottom of the supermodule corresponding to b belong to Π_t. Figure 5 shows an example that $b = BC*ED+*$ but D does not belong to Π_t in this case. A recursive procedure shown below is used to find Π_t efficiently give a, b, c and d. This procedure is only for sub-expression below the target module, .e., a. Procedures for sub-expressions above, to the left and to the right of the target module can be done similarly.

```
Procedure: Marking_Neighbor_Below(first, end)
Input: first is the first index of is valid
          sub-expression a
       end is the last index of a
       where first ≤ end
Output: Π is the set of module at the supermodule
          represented by a
1  If first = end
2     Π = Π ∪ α_first
3  Else
4  begin
5     Find k such that
5       e = α_{end−k} α_{end−2} ... α_{end−1}
6     where e is the shortest sub-expression
```

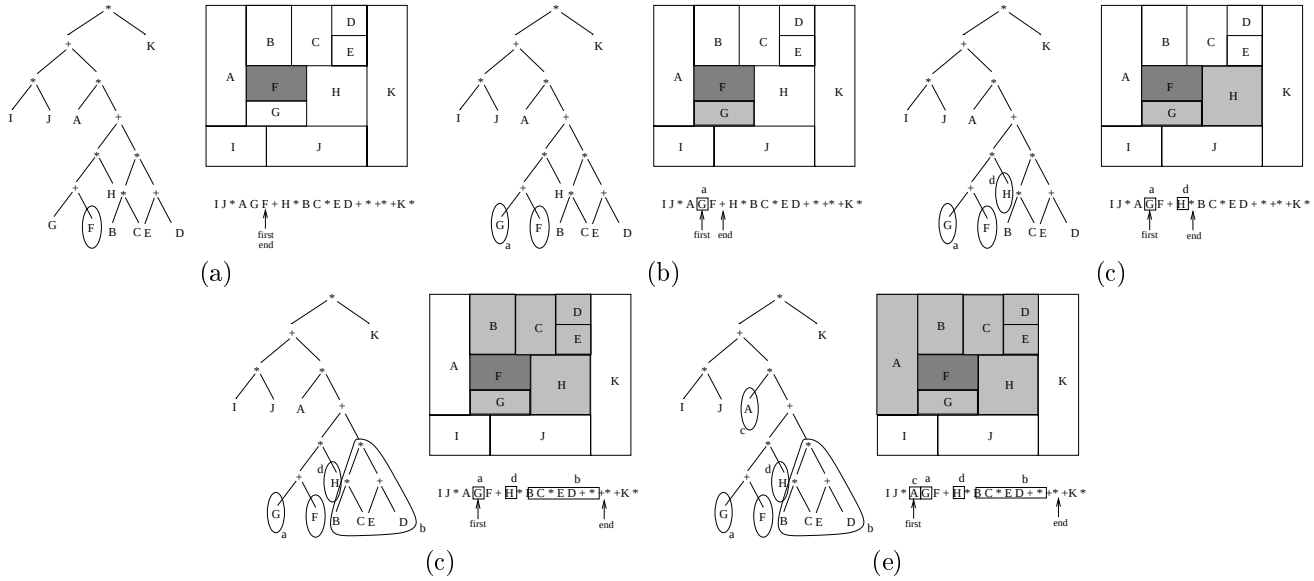

Fig. 4. An example to illustrate algorithm 1 Find_Surrounding

```
7     If α_end is *
8         marking_neighbor(first, end − k − 1))
9         marking_neighbor(end − k, end − 1)
10    Else if α_end is +
11        marking_neighbor(end − k, end − 1)
12 end
```

B. Constraint Satisfaction

In the annealing process, all the constraints have to be satisfied to make the floorplan feasible. Modules in the constraint set Δ will be swapped with the neighbor set Π_t until the conditions $\Delta \subseteq \{M_t\} \cup \Pi_t$ is satisfied.

In the first iteration, M_t is randomly selected from Δ. An intersect set Υ is defined to be $\Delta \cap \Pi_t$. If $|\Delta| > |\Upsilon| + 1$, swapping is needed to satisfy the clustering constraints. If $|\Delta| > |\Pi_t| + 1$, there is not enough space for swapping, the whole process will be repeated recursively by selecting another module which is already in the cluster as the new target module until all the constraints are satisfied. All three moves in the simulated annealing can affect the neighboring structure and give an infeasible packing. However we will swap operands in the Polish expression to maintain a feasible one.

There is only one case in move M1 that does not affect the neighboring structure of the packing, i.e., if two adjacent operands to be swapped are both in Δ or both in $\Phi - \Delta$. Modification is not required in this case and the clustering constraint will not be violated after the move.

The following algorithm describes the swapping strategy such that the clustering constraints are satisfy throughout the annealing process.

```
Algorithm : Clustering(α, Δ)
Input:α = α1α2...αn is a Polish expression of the problem.
      Δ is the set of modules having clustering constraint.
1  For each M_i ∈ Δ
2  begin
3      Call Find_Surrounding(M_i, α)
4      Call Marking_Neighbor_Below(first(a), end(a))
5      Call Marking_Neighbor_Above(first(b), end(b))
6      Call Marking_Neighbor_Left(first(c), end(c))
7      Call Marking_Neighbor_Right(first(d), end(d))
8      Υ_i = Δ ∩ Π_i
9      If |Υ| + 1 ≤ |Δ|
10         clustering constraints satisfied
11         Return
12 end
13 count = 0
14 While count < |Δ| − 1
15 begin
16     Take i where |Υ_i| is maximum and M_i is not marked
17     If |Π_i| ≥ |Δ| − 1
18     begin
19         For each M_k ∈ Δ ∩ ¬Π_i find M_j ∈ Π_i ∩ ¬Δ_i
20             swap(M_j, M_k)
21          count = |Δ| − 1
22     end
23     Else
24     begin
25         For each M_j ∈ Δ ∩ ¬Π_i find M_k ∈ Π_i ∩ ¬Δ_i
26             swap(M_j, M_k)
27         count = count + |Π_i|
28     end
29     Mark M_i
30 end
```

If $|\Pi_i| < |\Delta| - 1$ (lines 26-31), the number of positions in Π_i are not enough for swapping all the constrained modules into the neighboring positions. The other target module will be selected from Π_i and the process is repeated until all the constraints are satisfied. The algorithm can handle even large cluster size.

C. Multi-clustering Extension

Multi-clustering constraint allows us to have more than one cluster in the final packing. The algorithm described

above can only handle one cluster. Multi-clustering constraints can be resolved similarly. The major problem of addressing multi-clustering constraint is that the neighboring sets, can overlap. Infeasible packing will be resulted if modules are swapped randomly.

For example, given two clustering sets Δ_1 and Δ_2. A target module is found in each set M_{t_1} and M_{t_2}. Let Π_{t_1} and Π_{t_2} be the corresponding neighboring sets. If a module M_k, where $M_k \in \Pi_{t_1}$ and $M_k \in \Pi_{t_2}$, exists the neighboring sets M_k should be removed from either Π_{t_1} or Π_{t_2}.

While locating the neighboring modules, the module found earlier is probably nearer to the target module. Hence, it is better to swap into those positions first. This property make sure that the modules under clustering constraints will be placed as close to each other as possible.

D. Cost Function

The cost function is defined as $A + \lambda W + \beta C$ where A is the total area of the packing, W is the half perimeter estimation of the wirelength, and C is a penalty for the clustering constraint. The penalty term C is the sum of center to center distances between the modules within the same cluster. The penalty term helps to produce a packing in which the modules under clustering constraints will be packed closely together. λ and β are constants that control the weight of importance between the three terms.

V. Experimental Results

Our method is tested with three MCNC building blocks examples (ami33, ami49 and playout). Ami33 has 33 modules and 123 nets. Ami49 has 49 modules and 408 nets. Playout has 62 modules and 1161 nets. In the first set of experiment, we pick 20% of the modules randomly in each benchmark to have clustering constraint, i.e., ami33, ami49 and playout have 7, 10 and 12 modules respectively. For each benchmark, we repeat the experiment three times by selecting different modules into the constrained set. The results are given in Table I.

In the second set of experiment, we tested our method with multi-clustering constraints. In each benchmark problem, we picked 3, 4 and 5 clusters and each cluster has 2 to 7 modules. The results are given in Table II. A control experiment is performed without clustering constraint for each data set and the results are shown in Table III. The temperature decreases with a constant rate (0.9), and the number of iterations at one temperature step is one hundred times the number of modules. All experiments were done using UltraSPARC-II 400MHz processor.

Figure 6 and 7 shows a result packing of three clusters in ami33 and a result packing of four clusters in ami49 respectively. Figure 8 and 9 shows the improvement in interconnection by imposing clustering constraints. In Figure 8, we observed from the data set that modules 15, 18, 19, 20, 21, 24, 25 are heavily connected with each other, so we impose clustering constraint between them. Figure 9 shows the result packing without imposing any clustering constraints One can see that the interconnect cost is much smaller in Figure 8.

TABLE I
Results of testing with one cluster for the MCNC examples

Data Set	n	Cluster Size	% Dead-space	Wirelength (nm)$\times 10^6$	Time (sec)
ami33-cc1	33	7	1.62	0.1598	19.1
ami33-cc2	33	7	2.80	0.1600	18.4
ami33-cc3	33	7	2.74	0.1661	22.6
ami49-cc1	49	10	4.65	3.3379	53.4
ami49-cc2	49	10	3.53	3.2226	53.2
ami49-cc3	49	10	4.04	3.0970	51.9
playout-cc1	62	12	8.44	0.1770	146.5
playout-cc2	62	12	7.43	0.1862	147.8
playout-cc3	62	12	6.57	0.1868	146.4

TABLE II
Results of testing with multi-cluster for the MCNC examples

Data Set	n	# of Clusters (Cluster Size)	% Dead-space	Wirelength (nm)$\times 10^6$	Time (sec)
ami33-mc1	33	3(4,4,3)	2.35	0.1632	21.0
ami33-mc2	33	4(3,3,3,2)	3.16	0.1597	21.4
ami33-mc3	33	5(3,2,2,2,2)	2.17	0.1602	21.9
ami49-mc1	49	3(6,5,5)	3.83	3.3092	57.8
ami49-mc2	49	4(4,4,4,4)	2.77	3.3412	56.7
ami49-mc3	49	5(4,3,3,3,3)	3.99	3.4328	57.1
playout-mc1	62	3(7,7,6)	8.28	0.1800	156.9
playout-mc2	62	4(5,5,5,5)	7.18	0.1773	154.6
playout-mc3	62	5(4,4,4,4,4)	5.87	0.1728	151.8

TABLE III
Results of the control experiments

Data Set	n	% Deadspace	Wirelength(nm)$\times 10^6$	Time (sec)
ami33	33	2.45	0.1615	12.7
ami49	49	3.00	3.2894	37.5
playout	62	4.35	0.1729	124.4

References

[1] R. H. J. M. Otten, "Automatic floorplan design," in *Proceedings of the 19th IEEE/ACM International Conference on Computer-Aided Design*, pp. 261 – 267, 1982.

[2] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *Proceedings of the 23rd ACM/IEEE Design Automa-*

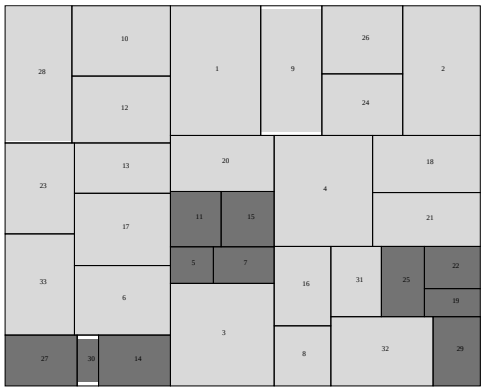

Fig. 6. A result packing of ami33 with three clusters (C_1:5,7,11,13; C_2:14,27,30; C_3:19,22,25,29)

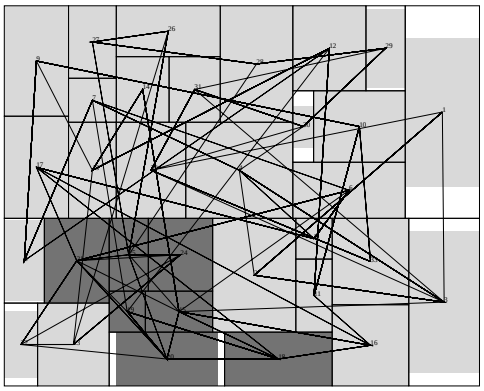

Fig. 8. A result packing showing the improvement in interconnection by imposing clustering constraints(wirelength = 0.1472×10^6nm).

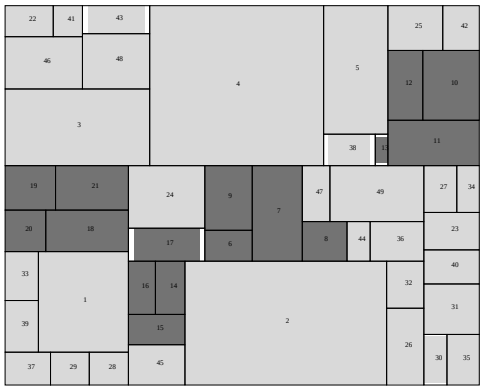

Fig. 7. A result packing of ami49 with four clusters (C_1:6,7,8,9; C_2:10,11,12,13; C_3:15,16,17,18; C_4:18,19,20,21)

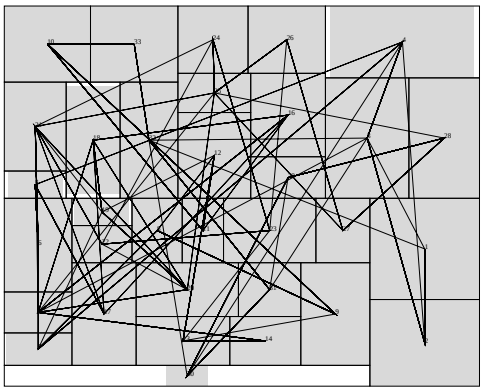

Fig. 9. A result packing of the same problem in Figure 8 without clustering constraints(wirelength = 0.1596×10^6 nm).

tion Conference (ACM/IEEE, ed.), (Las Vegas, NV), pp. 101–107, IEEE Computer Society Press, June 1986.

[3] H. Murata, K. Fujiyoshi, and M. Kaneko, "VLSI/PCB placement with obstacles based on sequence-pair," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 60–68, Jan. 1998.

[4] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, (Washington), pp. 484–493, IEEE Computer Society Press, Nov. 10–14 1996.

[5] P. N. Guo, C. K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," in *Proceedings of the 36th ACM/IEEE Design Automation Conference*, pp. 268–273, 1999.

[6] Y. C. Chang, Y. W. Chang, G. M. Wu, and S. W. Wu, "B*-trees: A new representation for non-slicing floorplans," in *Proceedings of the 37th ACM/IEEE Design Automation Conference*, pp. 458–463, 2000.

[7] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu, "Corner block list: An effective and efficient topological representation of non-slicing floorplan," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2000.

[8] M. Z. Kang and W. W. M. Dai, "Arbitrary rectilnear block packing based on sequence pair," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 259–266, IEEE Computer Society Press, 1998.

[9] H. Murata and E. S. Kuh, "Sequence-pair based placement method for hard/soft/pre-placed modules," in *Proceedings of International Symposium on Physical Design*, pp. 162–172, 1998.

[10] H. Youssef, S. M. Sqit, and K. J. Al-Farra, "Timing influenced force directed floorplanning," in *Proceeding of the EURO-VHDL, Proceedings EURO-DAC '95*, pp. 156–161, 1995.

[11] G. Vijayan, V. Narayananan, D. LaPotin, and R. Gupta, "PEPPER: A timing driven early floorplanner," in *International Conference on Computer Design*, (Los Alamitos, Ca., USA), pp. 230–235, IEEE Computer Society Press, Oct. 1995.

[12] T. Yamanouchi, K. Tamakashi, and T. Kambe, "Hybrid floorplanning based on partial clustering and module restructuring," in *Processing of the International Conference on Computer-Aided Design*, pp. 478 – 483, 1996.

[13] F. Y. Young and D. F. Wong, "Slicing floorplans with range constraints," in *Proceeding of the International Symposium on Physical Design*, pp. 97–102, 1999.

[14] F. Y. Young and D. F. Wong, "Slicing floorplans with boundary constraints," in *Proceeding of IEEE Asia South Pacific Design Automation Conference*, pp. 17–20, 1999.