VLSI/PCB PLACEMENT WITH OBSTACLES BASED ON SEQUENCE-PAIR

H. Murata[1] *K. Fujiyoshi*[2] *M. Kaneko*[3]

[1]Japan Advanced Institute of Science and Technology, Japan
murata@jaist.ac.jp
[2]Tokyo University of Agriculture and Technology, Japan
fujiyosi@cc.tuat.ac.jp
[3]Japan Advanced Institute of Science and Technology, Japan
mkaneko@jaist.ac.jp

ABSTRACT

In a typical VLSI/PCB design, some modules are pre-placed in advance, and the other modules are requested to be placed without overlap with these pre-placed modules. The presence of such obstacles introduces inconsistency to a coding scheme, called sequence-pair, which has been proposed for an obstacle free placement problem. We solve this difficulty by proposing a procedure, called "adaptation", which transforms inconsistent sequence-pair to consistent one, with utmost consideration for minimizing the modification. It is shown that a simulated annealing is well organized to test only feasible placements with the adaptation procedure. Using the adaptation, an MCNC benchmark data, ami49, is packed with 20% of the modules being pre-placed. Further, a PCB example which includes 32 free modules and 4 pre-placed modules (connectors) is laid out successfully by our method with a conventional wiring estimation followed by a commercial router.

1. INTRODUCTION

In VLSI design, it often happens that the locations of some macro cells, such as RAM, ROM, and CPU core, are fixed a priori and the other components are subject to be placed in the rest of the chip area. Also in PCB design, it is common that the exact coordinates of connectors are determined before designing the placement of the other components. We formulate such situations as a problem called "rectangle packing with pre-placed rectangles (RPP)". Not only the circuit components but also rectangular obstacles in any type are candidates to be modeled as pre-placed modules. For example, pre-placed modules can be used for representing a rectilinear substrate and holes of the substrate. The other "free" modules are requested to be placed onto the substrate without any overlap with the pre-placed modules.

Chi [2] studied a similar but restricted problem, where all the free modules have a regular height, assuming they are standard cells. Force-Directed-Relaxation method (FDR) [3, 4] can be easily tailored to handle the obstacles, but the method has inherent defects in the sensitiveness to the initial placement and in the incompleteness of the overlap elimination. The most practical way would be to use a stochastic algorithm, such as simulated annealing or genetic algorithm, if a proper coding scheme is available.

A coding technique for the slicing structure [5] is not useful for **RPP**, since the pre-placed modules might be given non-slicibly. For general (including both slicible and non-slicible) placements, two coding schemes are recently proposed, namely, sequence-pair [1] and Bounded-Sliceline-Grid [6], and we follow the former. Using the sequence-pair, it is easy to generate non-overlapping placements of all the modules by encoding all the (free and pre-placed) modules, but the code could be inconsistent to recover the locations of the pre-placed modules.

In this paper, we present a procedure called adaptation, which changes a sequence-pair so that it becomes consistent to the pre-placed modules. The procedure only changes the positions of pre-placed modules in a sequence-pair, and it runs in $O(n^2)$ time, where n is the total number of pre-placed modules and free modules. The alternative ways to incorporate the adaptation procedure in a simulated annealing are demonstrated through experiments on an MCNC building block benchmark example, named *ami49*. The simulated annealing is applied to a PCB example, with a standard wiring length estimation. The resultant placement has turned out to be successfully routed by a commercial router.

The organization of this paper is as follows. Section 2 gives a formal definition of **RPP**, and addresses that a sequence-pair can be inconsistent to **RPP**. In Section 3, the adaptation procedure is presented. Section 4 is devoted for the experiments. Section 5 is for conclusion.

2. PRELIMINARY

2.1. Rectangle Packing with Pre-Placed Rectangles (RPP)

A *module* is a rectangle. A *packing* of a set of modules is a non-overlapping placement of the modules. The coordinates of a module is the coordinates of the lower left corner of the module. A *pre-placed* module is a module whose coordinates as well as width and height are specified. A *free* module is a module whose width and height are specified but the coordinates is not specified. Set P of pre-placed modules is given such that no two pre-placed modules overlap each other and all of them lie in the first quadrant of the plane. Set F of free modules is also given. A *feasible* packing of $P \cup F$ is a packing of $P \cup F$ on the first quadrant of the plane such that all the modules in P

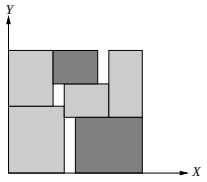


Fig. 1. Feasible packing example. The dark rectangles are the pre-placed modules.

are placed at their specified locations. The evaluation of a feasible packing is the area of the minimum bounding rectangle whose lower left corner is at the origin of the plane. Find the best feasible packing of $P \cup F$.

Fig. 1 shows a feasible packing of an instance of **RPP**.

When all the modules are free, i.e. $P = \emptyset$, then **RPP** coincides with a known packing problem, denoted by **RP**, which is proved to be $\mathcal{NP}$-hard [1]. Thus, **RPP** is also in $\mathcal{NP}$-hard class.

2.2. Sequence-Pair

For the free packing problem (**RP**), an elegant coding scheme is proposed in [1] as follows.

A *sequence-pair* for a set of n modules is a pair of sequences of the n module names. For example, (abc, bac) is a sequence-pair for module set $\{a, b, c\}$. It is easily understood that the variety of the sequence-pair for n modules is $(n!)^2$.

A sequence-pair imposes a horizontal/vertical (H/V) constraint for every pair of modules, as follows.

$(\cdots a \cdots b \cdots, \cdots a \cdots b \cdots) \quad \rightarrow \quad a$ should be placed to the left of b

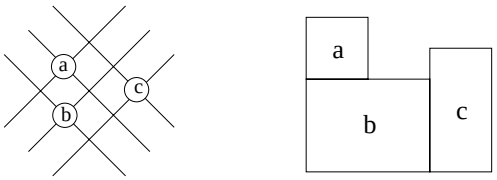
$(\cdots b \cdots a \cdots, \cdots a \cdots b \cdots) \quad \rightarrow \quad a$ should be placed below b

For example, sequence-pair (abc, bac) imposes a set of H/V constraints: { a should be placed to the left of c, b should be placed to the left of c, b should be placed below a}.

The H/V constraints of a sequence-pair can be intuitively grasped using the *oblique-grid* notation. For example, Fig. 2(a) shows the oblique-grid of sequence-pair (abc, bac). It is an $n \times n$ grid obliquely drawn on the plane, so constructed that the first sequence is observed when one reads the module names on the positive slope lines from left to right, and the second sequence is observed similarly with respect to the negative slope lines. It shows the H/V constraints in such a way that: Modules c is in the right quarter view range (between -45 degree and $+45$ degree) of module a, then c should be placed to the right of a.

It is proved in [1] that: The set of H/V constraints imposed by every sequence-pair is satisfiable, and an area minimum packing under the constraint can be obtained in polynomial time, and further, there is a sequence-pair which leads an (globally) area minimum packing. Then, the sequence-pair is easily utilized as a coding scheme of a stochastic algorithm.

To construct an area minimum placement for a sequence-pair, one dimensional compaction is carried out under the H/V constraints of the sequence-pair. The modules are greedily pushed to the left, and to the bottom, as shown in Fig. 2(b). The resultant placement is called the *free-realization* of the sequence-pair.



(a) sequence-pair (abc, bac) (b) free-realization

Fig. 2. Oblique grid notation of a sequence-pair and the free-realization of the sequence-pair

The free-realization can be obtained in $O(n^2)$ time [1] by using the "H/V constraint graph" which is constructed faithfully to the H/V constraints. More in detail: (Step 1) Construct a vertex weighted directed acyclic graph whose vertex set corresponds to the modules, and whose edge set corresponds to the horizontal constraints in the direction from left to right. The weight of each vertex is the widths of the corresponding module. (Step 2) Determine X coordinate of each module by the longest path length from the source nodes to the node of the module. (Step 3 and Step 4) Determine Y coordinate of each module in a similar way using the vertical constraints in the direction from bottom to top.

2.3. Feasibility of Sequence-Pair

Now we assume $P \neq \emptyset$, i.e., one or more modules are pre-placed. If we only encode the free modules into a sequence-pair, a free module and a pre-placed module can easily overlap each other in the free-realization of the sequence-pair, since the pre-placed module is totally ignored. Fig. 3(a) illustrates such an example for this situation. In the figure, free modules a, b, c, d are placed without considering the pre-placed modules x and y. As a result, d and b overlap with y.

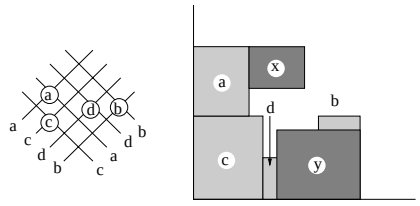
We employ an alternative approach, that is, to encode both the free modules and the pre-placed modules into a sequence-pair. One difficulty in the free-realization of such a sequence-pair is, the X (as well as Y) coordinate of a pre-placed module may be set too small, because modules are compacted to left (bottom) without considering the pre-assigned coordinates. Fig. 3(b) shows such an example, where the X and Y coordinates of pre-placed module x are set too small, and the X coordinate of pre-placed module y is set too small. This difficulty is easily solved by introducing an additional constraint for each pre-placed module:

> The X (Y) coordinate of each pre-placed module should not be less than the specified value.

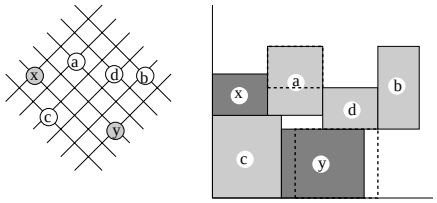
The free-realization procedure is easily modified to handle the additional constraints without increasing the asymptotic complexity, by adding additional edges from source to the pre-placed modules, with the pre-assigned coordinates as their weights, in the H/V constraint graph (Fig. 3(c)). The resultant placement is now called the *propped-realization*, named from an intuitive image of the additional constraints.

Fig. 3(c) shows the propped-realization for the same example in Fig. 3(b). The arcs in the oblique-grid denote the

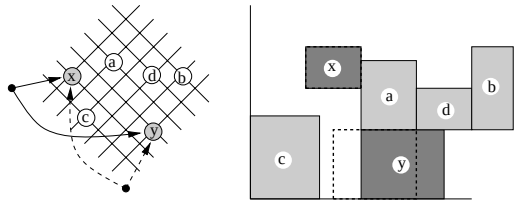
[1] The time complexity is reduced to $O(n \log n)$ by [7].



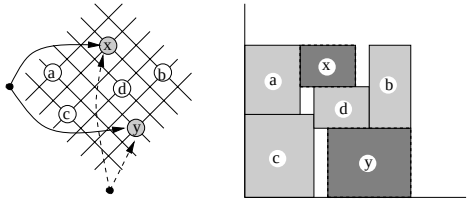
(a) sequence-pair of free modules



(b) sequence-pair of free and pre-placed modules



(c) additional constraints for pre-placed modules



(d) feasible sequence-pair

Fig. 3. Feasibility of Sequence-Pair

additional constraints. In the figure, pre-placed module x is placed at the specified location. However, pre-placed module y is not placed at the specified location (X coordinate is set too large). This is because the additional constraints introduced above can not prevent the coordinates being set too large. It is impossible to reform the X coordinate of module y by adding another constraint, since the sequence-pair inherently imposes y should be placed right of x. In the following, a sequence-pair is said *feasible* when its propped-realization is feasible, otherwise *infeasible*.

It is concluded that a sequence-pair is not necessarily feasible for **RPP**. However, it is still useful from the following fact.

Lemma 1 : For any instance of **RPP**, there is a feasible sequence-pair. Furthermore, there is a sequence-pair which leads an optimal solution of the problem.

□

A proof is omitted here.

Fig. 3(d) shows an example of a feasible (and optimal) sequence-pair and its propped-realization.

3. ADAPTATION

Among $(n!)^2$ sequence-pairs, there are feasible sequence-pairs, including one for optimal solution of **RPP**, and infeasible sequence-pairs. We should consider how to treat the infeasible sequence-pairs, when exploring the space by a stochastic algorithm. The easiest way would be to evaluate them infinitely negative, but the smoothness of the search would be strongly weakened. To keep the smoothness as much as we can, it is desirable that each infeasible sequence-pair is evaluated equally to a feasible sequence-pair which resembles to the infeasible one. For this purpose, we present a procedure, called *adaptation*, which transforms a given sequence-pair to a feasible sequence-pair, by changing only the positions of pre-placed modules.

3.1. Necessary Condition

Suppose unit square modules a and b are pre-placed at $(3,3)$ and at $(5,5)$, respectively. Then, both of "a is left of b" and "a is below b" are true, but "a is right of b" and "a is above b" are both false. It turns out that a is necessary to appear before b in the second sequence of a feasible sequence-pair. More in general, we describe a necessary condition for the second sequence of a feasible sequence-pair.

For any two pre-placed modules a and b, we say that a *dominates* b if

$$x(a) < x(b) + w(b) \quad \text{and} \quad y(a) < y(b) + h(b)$$

where

- $x(a), y(a)$: X and Y coordinates of module a,
- $w(a), h(a)$: width and height of module a.

The domination relation introduces a partial order into the module set. Then, the second sequence of a sequence-pair is said to be *topologically sorted* if it satisfies the condition: If a dominates b, then a appears prior to b in the second sequence. It is clear that the following lemma holds.

Lemma 2 : It is necessary for a sequence-pair being feasible that the second sequence is topologically sorted. □

3.2. Algorithm

In the propped-realization of a sequence-pair, the X and Y coordinates of a module is determined only by the preceding modules in the second sequence. It turns out that the coordinates of a module can be determined one by one, traversing the second sequence. Then, we design our adaptation procedure so that it iteratively "test and virtually place" a module according to the second sequence. Thus, after some iterations, free modules would be virtually placed, as well as pre-placed modules. The relation "dominate" was already defined for the pre-placed modules, but in the following, we use the relation also for a free module when it is virtually placed.

Procedure Adaptation

Input: A set of pre-placed modules, a set of free modules, a sequence-pair.

Output: A feasible sequence-pair.

(Step 1) Topologically sort the second sequence, only when it is not already topologically sorted.

(Step 2) For $k = 1, 2, \ldots, n$, repeat **(Step 2.1)** through **(Step 2.4)**.

(Step 2.1) Let the k'th module in the second sequence be a. If module a is a pre-placed module, go to **(Step 2.4)**. Otherwise, temporary set the coordinates of a, according to the propped-realization of a.

(Step 2.2) Determine whether there exists a pre-placed module which dominates a, in the last $n-k$ modules of the second sequence. When it exists, determine a pre-placed module q such that q directly or transitively dominates a, and there is no pre-placed module which dominates q. Otherwise, go to **(Step 2)** for the next iteration.

(Step 2.3) In the second sequence, move q just before a. After that, a is used for referring q. (since q is now the k'th module in the second sequence.)

(Step 2.4) If a is a free module, go to **(Step 2)** for the next iteration. Let (x, y) be the coordinates of a in the propped-realization. Let $(x(a), y(a))$ be the specified coordinates of pre-placed module a. If $x > x(a)$, then in the first sequence, move a minimally toward the top so that $x \leq x(a)$ holds. Otherwise, if $y > y(a)$, then in the first sequence, move a minimally toward the end so that $y \leq y(a)$ holds.

3.3. Illustrative Example

A behavioral example is presented for the **RPP** instance shown in Fig. 4(a), in which the dark two modules x and y are pre-placed, and the rest of the modules a, b, c, d are free.

Suppose sequence-pair $(acdbyx, cadxyb)$ is given as the input sequence-pair. Fig. 4(b) shows this initial sequence-pair. In **(Step 1)**, x and y are exchanged in the second sequence, and the sequence-pair becomes $(acdbyx, cadyxb)$, as shown in Fig. 4(c).

Now, **(Step 2)** begins. In iteration $k = 1$ and in $k = 2$, free modules c and a are processed because they are the first and the second modules in the second sequence, respectively. They are virtually placed at the positions shown in Fig. 4(a).

In iteration $k = 3$, module d is once tried to be placed at the right of c in **(Step 2.1)**, as indicated by the rectangle with dotted lines, which is marked "d" in Fig. 4(a). Then, module y is selected in **(Step 2.2)**, and it is brought before d in the second sequence in **(Step 2.3)**. As a result, the sequence-pair becomes $(acdbyx, caydxb)$ as shown in Fig. 4(d). Module y is virtually placed at its specified position, as it is marked "y" in Fig. 4(a).

In iteration $k = 4$, module d is placed at the position marked "d'"in Fig. 4(a).

In iteration $k = 5$, pre-placed module x is processed. In **(Step 2.4)**, it is once tried to placed at the position indicated by the rectangle with dotted lines, which is marked "x" in Fig. 4(a), since it is the position imposed by the sequence-pair shown in Fig. 4(d). Then, the X coordinate is found too large, and module x is moved toward the top of the first sequence, and put between a and d. The sequence-pair becomes $(axcdby, caydxb)$, which is illustrated in Fig. 4(e). Module x is virtually placed at its specified position, as it is marked "x'" in Fig. 4(a).

This sequence-pair is not changed in iteration $k = 6$, and becomes the output of the procedure.

One can examine on this example that the resultant sequence-pair is feasible. It is important to note that the H/V constraints among the free modules are preserved.

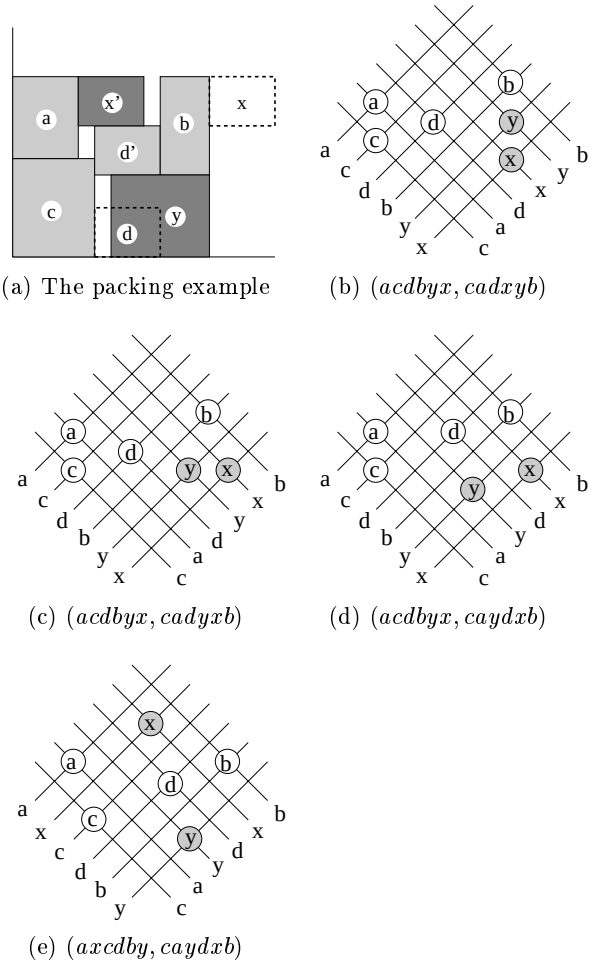


Fig. 4. Snapshots of the adaptation procedure. (a) shows the packing corresponding to the sequence-pair shown in (e). (b) shows the input sequence-pair. In (c), y is brought before x in the second sequence. In (d), y is brought before d in the second sequence. In (e), x is brought before c in the first sequence.

3.4. Proof of Adaptation

Theorem 1 : The adaptation procedure changes the given sequence-pair such that

(1) the output sequence-pair coincides with the input sequence-pair if the given sequence-pair is feasible,

(2) the output sequence-pair is always feasible,

(3) the H/V constraints with respect to the free modules are preserved,

(4) the procedure runs in $O(n^2)$ time, where n is the total number of the pre-placed modules and the free modules.

Proof :

(1) and (3) are easily understood. A proof for (2) is omitted because of the space limitation. Outline of a proof for (4) is presented in the following.

(Step 1) can be done in $O(n^2)$ time as follows.

(a) Construct an $n \times n$ matrix D such that

$$D(a,b) = \begin{cases} 1 & \text{if module } a \text{ directly or transitively} \\ & \text{dominates } b \\ 0 & \text{otherwise} \end{cases}$$

D can be constructed in $O(n^2)$ time by sorting the modules by their coordinates in X and in Y, essentially because of the fact: If a transitively dominates b and $x(a) < x(b)$, then there is a series of modules from a to b, dominating one after another, and their X coordinates are monotonically increasing.

(b) Topologically sort the second sequence using D. This can be done also in $O(n^2)$ time by a selection sort algorithm.

(The claim is easily proved also by the ordinary depth first algorithm, but we use the above algorithm with utmost consideration for minimizing the modification.)

(Step 2.1) is easily done in $O(n)$ time. **(Step 2.2)** can be done in $O(n)$ time by traversing the modules reversely in the second sequence. **(Step 2.3)** can be easily done in $O(n)$ time. **(Step 2.4)** can be done in $O(n)$ time by introducing two arrays $X[1 \ldots n]$ and $Y[1 \ldots n]$ which holds a "negative locus" [1] of the k'th module in the second sequence after the module is virtually placed.

(Step 2.1) through **(Step 2.4)** is repeated n times in **(Step 2)**. Therefore, the adaptation can be done in $O(n^2)$ time. □

4. EXPERIMENTS

4.1. Packing Experiments

The adaptation procedure is implemented in a standard simulated annealing to solve **RPP**. The outline of the simulated annealing is as follows.

Procedure SA

Input: Set P of pre-placed modules, set F of free modules, number of iterations, initial temperature, and final temperature.

Output: A feasible packing of $P \cup F$.

(Step 1) Generate a random sequence-pair, initialize the loop counter, set temperature to the initial value, and set the decreasing ratio of the temperature such that it reaches to the final temperature when the loop counter reaches to the limit.

(Step 2) If the loop exceeds the given limit, output the best packing obtained so far and then stop.

(Step 3) Apply one of the following three move operations to alter the sequence-pair.

- exchange two module names in the first sequence,
- exchange two module names both in the first sequence and the second sequence, and
- exchange the width and the height of a module.

(Step 4) Evaluate the sequence-pair by the area of the corresponding placement.

(Step 5) If the evaluation is improved or not changed, then accept the change. Otherwise, accept the change stochastically depending on the temperature and on the difference of the evaluations.

(Step 6) Decrease the temperature exponentially by the ratio obtained in **(Step 1)**, and go to **(Step 2)**.

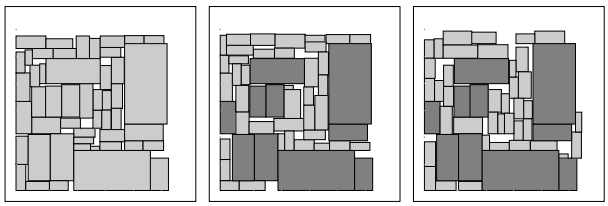


Fig. 5. Three packings of ami49. The left figure is obtained by RP. The middle and the right figures are obtained by Adapt-in-Eval method and Adapt-in-Move method, respectively, setting the dark modules being pre-placed.

Table 1. An experimental result of ami49. (average of 10 runs)

	RP	Adapt-in-Eval	Adapt-in-Move
Area (mm^2)	37.978808	38.353762	40.062617
Time (sec)	331.78	1004.15	949.68

The adaptation can be used in one of the following two manners:

- Adapt-in-Eval : Use the adaptation procedure internally in the evaluation step **(Step 4)**, intending to evaluate an infeasible sequence-pair as an equivalent of a feasible sequence-pair.
- Adapt-in-Move : Use the adaptation procedure internally in the initialization step **(Step 1)** and in the move step **(Step 3)** so that an infeasible sequence-pair is never generated.

The major difference of the two methods would be as follows. In both methods, the SA explores the space of $(n!)^2$ sequence-pairs. Suppose there is a cluster of infeasible sequence-pairs in the solution space. The search may go into the cluster in the Adapt-in-Eval method, while in the Adapt-in-Move method, it is flicked out from the cluster right after an infeasible sequence-pair is generated eventually. Then, the reachability to an optimal solution is guaranteed in the Adapt-in-Eval method, and not in the Adapt-in-Move method.

The biggest MCNC building block benchmark example *ami49* is used as the input data. As a preliminary run, the 49 modules are all treated as free modules, and they are packed by the simulated annealing. The first 10 biggest modules, with respect to the areas, are specified as pre-placed modules whose coordinates are fixed to those obtained by the preliminary run. The reason why these modules are selected is because of our experience that the obstacles are usually not many, but big. The same input, except for the presence of the pre-placed modules, is applied to Adapt-in-Eval method and to Adapt-in-Move method.

Fig. 5 shows the resultant packing of these methods, including the preliminary run. Table 1 shows the average performance of 10 runs of the Adapt-in-Eval method, Adapt-in-Move method, and also the preliminary run in the column **RP** for reference. Every trial is performed on a Sun SS-5 (75 MHz).

The run time is shortest in **RP**, as in Table 1. This would be natural because the adaptation is not executed, and also because **(Step 4)** runs in $O(n \log n)$ time when

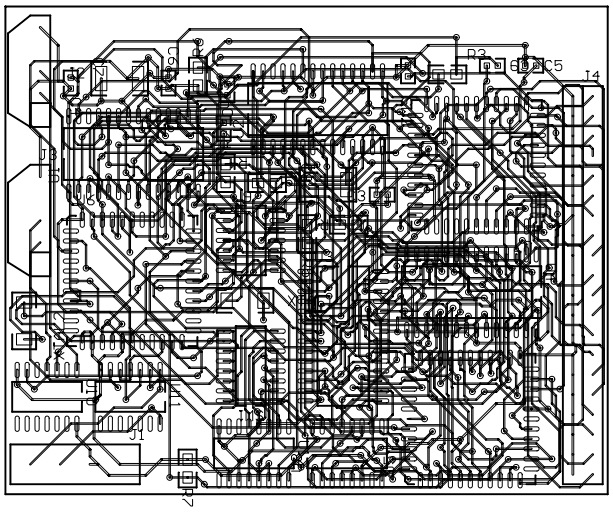


Fig. 6. A place-and-route result of DEMOSMD. Connectors J1,J2,J3 and J4 are pre-placed.

there is no pre-placed modules, with an algorithm similar to [7]. The reason why Adapt-in-Move runs faster than Adapt-in-Eval method would be because the input sequence-pair of the adaptation in Adapt-in-Move method is feasible or almost feasible, while that of Adapt-in-Eval method can be far from feasible.

It is interesting that **RP** achieved the minimum area, although the number of the tested sequence-pairs is the same as those for Adapt-in-Move and Adapt-in-Eval. This implies that the adaptation can not compensate completely the inconsistency introduced by the pre-placed modules. It is also observed from Table 1 that Adapt-in-Eval method achieves better than Adapt-in-Move method. This result seems to reflect the reachability difference which is mentioned earlier in this section. From this reachability difference together with the above experimental result, we can conclude that Adapt-in-Eval method is preferable for practical applications.

4.2. Place and Route Experiments

Based on Adapt-in-Eval method, a PCB example is placed with setting the connectors pre-placed.

Wires are considered additionally to the area in the evaluating function. To keep the wiring space, the widths and the heights of the modules are uniformly enlarged. After the packing is obtained from a sequence-pair, the minimum spanning tree is calculated for each net, and the sum of the length of each edge in the tree is used as an estimation of the wiring length of the net. The evaluation of a sequence-pair is the area of the packing plus the total sum of the estimated length of every net, with weighting coefficients which are assigned so that the term of the area and the term of the estimated wiring length are approximately balanced.

A PCB example which includes 36 modules and 88 multi-terminal nets was used in the experiment. The data is called DEMOSMD, which is provided with a commercial layout editor (Protel Advanced PCB 2.8). The simulated annealing was scheduled on this data to try 100,000 moves, on a note-type personal computer (Sharp Mebius PC-A355, Pentium 100MHz). The resultant placement was given to a commercial router (Protel AdvancedRoute3) to finish the design using 4 layers.

Table 2. An experimental result of DEMOSMD

	MST	MBOX
Total wiring length (mm)	7,914.64	8,188.96
Number of vias	368	382
Placement time (sec.)	630	150
Routing time (sec.)	568	590
Total time (sec.)	1,198	740

All the nets were successfully routed, as shown in Fig. 6. The performance is summarized in Table 2, in the column labeled with "MST".

By replacing the MST estimation with the half perimeter of the minimum bounding box, which is a popular way to speed up the estimation, the same trial was carried out and also completely routed by the router. Results are in Table 2, in the "MBOX" column.

5. CONCLUSION

We showed that the presence of the pre-placed modules introduces an inconsistency to the sequence-pair coding scheme. An adaptation procedure is proposed to transform an inconsistent sequence-pair to a consistent one. It is shown that a simulated annealing is well organized to test only feasible placements with this adaptation procedure. A PCB example is placed with a standard wiring estimation, and is routed successfully by a commercial router.

ACKNOWLEDGMENT

Authors would like to thank Professor Yoji Kajitani and Research Associate Shigetoshi Nakatake of Tokyo Institute of Technology for their helpful discussions. This work was supported in part by Research Body CAD21.

REFERENCES

[1] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-Packing-Based Module Placement," in *IEEE International Conf. on Computer Aided Design*, pp. 472–479, 1995.

[2] M. C. Chi, "An Automatic Rectilinear Partitioning Procedure for Standard Cells," in *24th ACM/IEEE Design Automation Conference*, pp. 50–55, 1987.

[3] L. Sha and R. W. Dutton, "An Analytical Algorithm for Placement of Arbitrarily Sized Rectangular Blocks, " in *Proc. 22th ACM/IEEE Design Automation Conf.*, pp. 602–608, 1985.

[4] A. Alon and U. Ascher, "Model and Solution Strategy for Placement of Rectangular Blocks in the Euclidean Plane," *IEEE Trans. on CAD*, vol. 7, no. 3, pp. 378–386, 1988.

[5] D. F. Wong and C. L. Liu, "A New Algorithm for Floorplan Designs," in *Proc. 23rd ACM/IEEE Design Automation Conf.*, pp. 101–107, 1986.

[6] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module Placement on BSG-Structure and IC Layout Applications," in *IEEE International Conference on Computer-Aided Design*, pp. 484–491, 1996.

[7] T. Takahashi, "An Algorithm for Finding a Maximum-Weight Decreasing Sequence in a Permutation, Motivated by Rectangle Packing Problem," *Technical Report of IEICE*, vol. VLD96, no. 201, pp. 31–35, 1996.