Branch-and-Bound Placement for Building Block Layout

Hidetoshi Onodera[†‡], Yo Taniguchi[‡], and Keikichi Tamaru[‡]

[†] Department of Electrical Engeneering and Computer Sciences
University of California, Berkeley, CA 94720
[‡] Department of Electronics, Kyoto University
Sakyo-ku, Kyoto 606 Japan

Abstract

We present a branch-and-bound placement technique for building block layout that effectively searches for an optimal placement in the whole solution space. We first describe a block placement problem and its solution space. Then we explain branching and bounding operations designed for the placement problem. Constraints on critical nets and/or the shape of a resulting chip can be taken into account in the search process. Experiments reveals that the number of blocks the method can manage is around six if the whole solution space is explored. For a problem which contains more blocks than the limit, we decompose the problem hierarchically and apply the method to each sub-problem. The results for standard benchmark examples and a comparison with those of other systems are given to demonstrate the performance of the method.

1 Introduction

The problem of placing building blocks on a two-dimensional surface is a very complex combinatorial optimization problem. Even subsets of the problem are NP-complete or NP-hard. In order to reduce the complexity of the problem, many approaches proposed so far use heuristics to restrict and abstract the solution space to be examined, and obtain an approximate solution [1]. For example, many of them restrict their placements to ones with the slicing structure [2]. A variety of heuristics have been deviced to obtain a good local optimal solution, which includes the min-cut algorithm [3], force-directed methods [4, 5], simulated annealing [6, 7, 8], etc.

In contrast to this, very few algorithms have been proposed that will find an optimal solution within the constraints of the model used. There are however a few exceptions including Refs. [9, 10] of which Ref. [9] by Preas and vanCleenput is the earliest work. They described a recursive procedure which constructs an optimal placement by adding blocks one by one to a partial placement. They, however, imposed a limit on the depth of recursion when implementing the algorithm, thus the solution remains sub-optimal. Dai and Kuh proposed a method which enumerates possible placements using pre-defined floorplan templates [10]. However, the number of blocks treated at a time was limited to five or less, and not all the possible templates were used.

In this paper, we present a method which belongs to the latter approach, that is, a method which searches for an optimal solution in the whole solution space. Since the number of possible placements increases explosively with the number of blocks, we employ a branch-and-bound strategy for effective exploration. Constraints on critical nets and the shape of a resulting placement are taken into account in the search process. For a large problem which cannot be handled at a time, we decompose it hierarchically and construct a placement in a bottom-up manner.

This method was first introduced at the 1990 International Workshop on Layout Synthesis [11], and demonstrated its performance successfully at the benchmark session. This paper describes the details of the method and shows experimental results as follows. Section 2 describes the formulation of the block placement problem and its solution space. Section 3 gives a search procedure using a branch-and-bound technique. Section 4 shows experimental results for several examples including standard benchmark circuits distributed by MCNC, results of which are favorably compared with those of other systems. Finally, Section 5 presents concluding remarks.

2 Building Block Placement — Problem Description —

In this paper, we discuss the problem of placing building blocks. Inputs of the problem consist of

- a set of building blocks with fixed geometries and fixed pins,
- a net list specifying which pins should be interconnected,
- constraints on the locations of external pins (pads), if any,
- constraints on the shape of the chip (eg. aspect ratio, width, height, etc.), if any,
- constraints on critical paths (nets), if any.

Given the inputs, the problem to be solved is to:

> determine the locations and orientations of all the blocks so that all the constraints are satisfied and so that the area of the chip is minimized while ensuring routability.

The constraints on critical paths may be imposed in several different forms. The maximum and/or minimum delay time may be specified. The range of delay time associated with a path may be specified as a function of that or some other path. These constraints on delay time, however, can be converted to constraints on the lengths of nets associated with the path (hereafter, we call these nets as critical nets) using such a method as described in Ref [12]. We thus assume that all the constraints on critical paths are transformed to the constraints on the length of critical nets.

In the original problem stated above, routability should be ensured in a resulting placement. The routability, however, is very difficult to evaluate at this stage where detailed routing is not yet carried out. Hence we approximately take it into account by appending a routing region to each block periphery beforehand. The

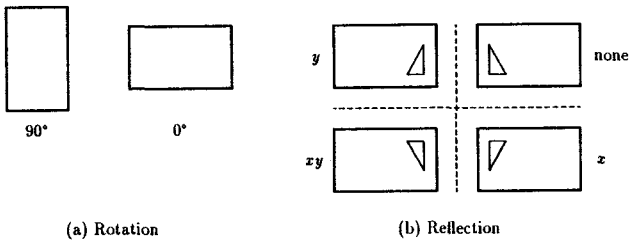

Figure 1: Orientation of a block.

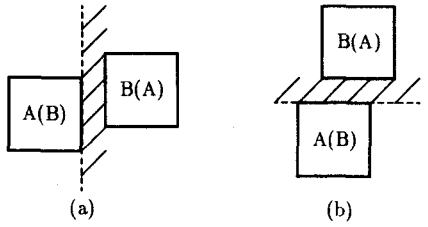

Figure 2: Topological relationship between blocks.

amount of the routing region is estimated according to the number of pins on each side of the block. Besides the pre-determined routing space, we also estimate additional routing space from the calculation of net length based on actual pin locations, and include it in the estimation of the chip area.

Thus, we adopt the following equation as an objective function for evaluating a placement.

$$(W_x + T_x \times \frac{\sum L_{iy}}{W_y}) \times (W_y + T_y \times \frac{\sum L_{ix}}{W_x}) \qquad (1)$$

where W_x and W_y represent the width and height of the chip (the smallest rectangle that encloses all the blocks), T_x and T_y represent the spacings between routing tracks in the x and y directions, and L_{ix} and L_{iy} are the width and height of the smallest rectangle enclosing all the pins of net i. The second term in each pair of parentheses represents the amount of increase associated with routing space which is estimated assuming all the nets are uniformly distributed in the whole chip area. The problem we want to solve is to find a placement which minimizes the objective function while satisfying all the given constraints.

Next we define the space where the search for an optimal solution is carried out. There are several ways to describe a placement configuration. Wong and Liu have proposed a Polish postfix representation for describing a placement with slicing structure [8]. Dai and Kuh use pre-defined floor-plan templates for describing placement configurations [10]. Here we take a more general approach which is a direct extension to the one used in the formulation of a two-dimensional compaction problem [13, 14]. We describe a placement by specifying the orientation of each block and the topological relationship (constraint) between each pair of blocks (see Figs. 1 and 2). The orientation is described by specifying the rotation (0° or 90°) and reflection (none, x, y, or xy) of a block as shown in Fig. 1. Thus we are to specify one orientation out of eight for each block. The topological relationship between two blocks is classified into four constraints (Fig. 2). Assume there are Blocks A and B, and (x_a, y_a) represents the center position of Block A, and h_a and w_a represent the height and width of Block A respectively. The four constraints are:

- Block A is to the right of Block B: $x_a - x_b \geq \frac{1}{2}(w_a + w_b)$
- Block A is to the left of Block B: $x_b - x_a \geq \frac{1}{2}(w_a + w_b)$
- Block A is above Block B: $y_a - y_b \geq \frac{1}{2}(h_a + h_b)$
- Block A is below Block B: $y_b - y_a \geq \frac{1}{2}(h_a + h_b)$

They are all expressed in a form of simple linear inequalities. The four constraints are not exclusive, and at least one constraints should be satisfied to prevent overlapping of blocks. If we assume the number of blocks is N_b, there are ${}_{N_b}C_2 = N_b(N_b-1)/2$ block pairs. Thus we are to specify one topological constraint out of four for each of the $N_b(N_b-1)/2$ pairs.

Not all the combinations of topological constraints correspond to a "legal" placement configuration. A combination of constraints which introduces inconsistency in block locations (such as, Block A is to the right of Block B, Block B is to the right of Block C, and Block C is to the right of Block A) does not lead to a feasible placement. Except for the above case, given a set of topological constraints on all the block pairs, we can derive a placement of minimum height and width subject to the constraints.

Now the problem to be solved can be stated that:

> Find an optimum combination of orientations for all the blocks and topological relationships for every pair of blocks, which combination leads to a placement that minimizes the objective function (1) subject to all the given constrains such as critical net length and aspect ratio of the chip.

3 Branch-and-Bound Placement

We may find a solution by examining all the possible combinations exhaustively. However, the number of combinations becomes explosively large as the number of blocks grows. Assuming the number of block is N_b, the total number of combinations including illegal placements is $2^{N_b(N_b+2)}$. Thus a simple enumerative method is only applicable to a very small problem. In order to cope with the vast solution space, we have employed a branch-and-bound technique for effective exploration.

3.1 Branch-and-Bound Method

In a branch-and-bound method, search is carried out according to a decision tree (ex. Fig. 3). The decision tree, which consists of nodes and branches, represents a search schedule, and the value of each decision variable is specified one by one at each level of the tree according to the schedule. In our problem, a decision variable corresponds to a selection of the orientation of a block or a selection of the topological constraint on a pair of blocks. Except for a leaf node which leads to a placement of the original problem, a node in the decision tree represents a partial problem in which some decision variables are fixed and others are not. A branch represents a possible assignment of a value to a decision variable.

For each node in the tree, a lower bound of the objective function is computed using a corresponding partial problem. If the lower bound exceed the currently known lowest value of some solution, the search process along the branch is terminated (bounding operation). Thus, by pruning unnecessary branches in the decision tree, the branch-and-bound method explores the solution space effectively to find a solution.

3.2 Branching Operation

The design of a branching schedule (a decision tree) is very important since it affects the efficiency of the method a great deal. A good

Table 1: Branching schedule

	A	B	C	D	...
A	1	2	4	7	.
B		3	5	8	.
C			6	9	.
D				10	.
⋮					.

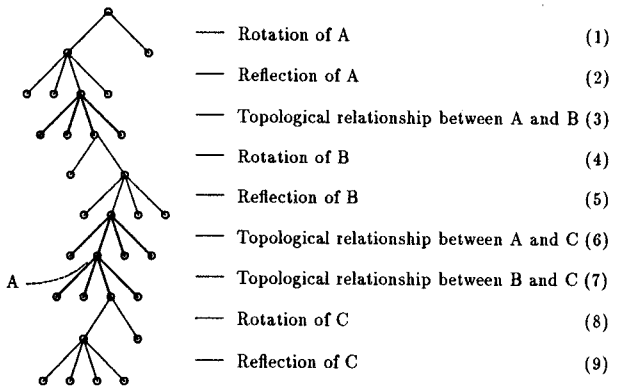

Figure 3: A part of the decision tree for a three-block problem.

schedule leads to a good partial problem from which we can obtain a good lower bound. The better the lower bound, the sharper the pruning, which results in reduced computation.

Table 1 shows the branching schedule we adopt for the block placement problem. The order of decision is listed in the table. A, B, C, ... represent blocks to be placed. A diagonal element represents the decision on the orientation of a corresponding block. An off-diagonal element represents the decision on the topological relationship of a corresponding pair of blocks. Blocks A, B, C, ... are sorted in decreasing order according to the magnitude of their influence on a placement. In usual cases, size of a block is a good measure for assessing the influence. If constraints on critical nets are imposed, a block related to the critical nets should be ranked higher for exploiting the possibility of pruning associated with the critical net constraints. In our branching schedule, at the time the orientation of a block is determined, the topological relationships between the block and those higher in the order are already established, which means that we can obtain a partial placement of these blocks from which a lower bound of the original problem is calculated.

We choose the orientation of each block in two steps. The first step select its rotation (Fig. 1(a)) and the second step select its reflection (Fig. 1(b)). This is because the magnitude of influence on a placement is different between rotation and reflection. Rotation affects the dimensions of the placement directly, whereas reflection does only through the change in routing area. Fig. 3 shows a part of the decision tree for a problem with three blocks.

There are many search schedules in traversing a decision tree. In this study, we use a simple depth-first search because of its moderate requirement on storage.

3.3 Bounding Operation

The efficiency of the branch-and-bound method also depends on a bounding technique. Obviously, without any bounding operations, it becomes just a simple enumerative search. For effective bounding, we calculate a lower bound at each node in a decision tree. Be-

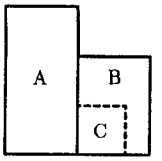

Figure 4: A placement of a partial problem.

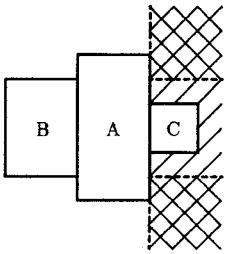

Figure 5: A case in which the constraint between B and C is redundant.

sides the pruning by a lower bound, there exists the possibility of pruning associated with topological constraints among the blocks, shape constraints of the chip, and critical net constraints. These bounding operations are described in the following.

Pruning by Lower Bound

Each node in a decision tree corresponds to a partial problem in which orientations of some blocks are not yet determined (here we call them as "non-oriented" blocks) and topological constraints on some block pairs are not yet established. Accordingly, we treat a non-oriented block as a square block whose width and height are equal to the length of the smaller side of the block when we calculate the placement area. In the calculation of net length, we take into account nets between oriented blocks only, since complete topological relationships are established only among those blocks.

For example, suppose we are at node A in the decision tree shown in Fig. 3. At this point, the orientations of Blocks A and B are already fixed, but that of Block C is not. The topological relationships between Blocks A and B and between Blocks A and C are fixed, but not between Blocks B and C. A possible solution of a corresponding partial problem is shown in Fig. 4. In the calculation of W_x and W_y, Blocks A and B remain unchanged, while Block C is treated as a square. Also, in the calculation of L_{ix} and L_{iy}, we don't consider all the nets but those which interconnect pins in Blocks A and B only. The value of the objective function calculated in this way gives a lower bound of the original problem. The lower bound is examined whether we can prune the fan-in branch to the node or not.

Pruning by Topological Constraint

If a branch corresponds to the selection of a topological constraint between a pair of blocks, there exists a situation in which we can prune three branches out of four at the same decision level. This happens when one of four topological constraints introduced at the decision level is redundant, that is, the topological constraint to be imposed is already satisfied by the topological constraints established previously. In this case, except for the redundant branch, we can prune three other branches at the decision level.

Fig. 5 explains this situation which possibly happens at level (7) in Fig. 3. We have already established the topological constraints between Blocks A and B and between Blocks A and C, that is, "

Block A is to the right of Block B", and "Block C is to the right of Block A". Feasible region for Block C is indicated in Fig. 5 by shading. In this situation, the constraint that "Block C is to the right of Block B" is redundant and the feasible region for Block C remains the same after imposing the constraint. On the other hand, the reverse constraint that "Block C is to the left of Block B" introduces a cycle in the topological order of Blocks A, B, and C, and thus the corresponding branch should be pruned. Also, the rest two branches, which correspond to the constraints that "Block C is above (below) Block B", can be pruned because the two constraints only limit the feasible region for Block C, which limited region is indicated by cross hatching in Fig. 5. In other words, the solution spaces defined by the two branches are included in the solution space defined by the redundant branch, and thereby need not be explored. Thus we can prune three branches out of four at a time.

Pruning by Shape Constraint

We consider width/height constraints and aspect ratio constraints for shape constraints on a placement.

In the case that a maximum width/height is given, we can prune a branch in the decision tree when the width/height of a partial placement, which is derived from a corresponding partial problem, exceeds the limit.

In the case that an aspect ratio is given, we set up and update maximum width and height of a placement and use them for pruning, as described in the following. Assume a represents the given aspect ratio, and $H_{\max}$ and $W_{\max}$ represent the maximum height and width of a placement. First, we set $H_{\max}$ and $W_{\max}$ as

$$H_{\max} = \sqrt{bS/a}$$
$$W_{\max} = \sqrt{bS\,a},$$

where $b(>1)$ is a constant which represents the maximum allowable area of a placement in proportion to the total area S of all blocks. The constant b is only used for setting initial values to $H_{\max}$ and $W_{\max}$, so that it does not affect final solutions. When a temporary solution of the original problem is obtained, they are updated as

$$H_{\max} = \max(W/a, H)$$
$$W_{\max} = a\,H_{\max},$$

where H and W stand for the height and width of the temporary solution. $H_{\max}$ and $W_{\max}$ thus give the height and width of the bounding rectangle of aspect ratio a for the temporary solution. During the search process, if the width and/or height of a partial problem exceed $W_{\max}$ and/or $H_{\max}$, we can prune a corresponding branch. Every time a new temporary solution is obtained, $H_{\max}$ and $W_{\max}$ are reduced, which results in more effective pruning.

Pruning by Critical Net Constraint

In the case that the upper bounds of critical nets are given, we can prune a branch if the possible length of a critical net exceeds its limit in a corresponding partial problem. On the other hand, we don't utilize the lower bounds of critical nets, if any, for pruning since it is not easy to confirm their violations until the search reaches to a leaf of the decision tree, ie. a solution of the original problem is obtained.

3.4 Solution Method

The calculation of a lower bound at each node requires to obtain the minimum value of the objective function Eq. (1) subject to a set of linear inequalities which represent topological constraints among blocks. The calculation of the minimum in an exact way, however, is very difficult. The difficulty stems from the necessity of simultaneous consideration for two terms in Eq. (1) which are related to the area and wire length of a placement. In usual cases, expansion of a placement area leads to an increase in wire length. We thus obtain the minimum of Eq. (1) approximately in two steps. First we minimize the placement area $W_x W_y$ and then calculate the wire length of the placement, from which the value of Eq. (1) is calculated.

The minimization of the area $W_x W_y$ is decomposed into two linear programming problems of minimizing W_x and W_y respectively, because a set of topological constraints is separated into two independent groups consisting of constraints on x or y locations only. Each problem is equivalent to a one-dimensional compaction problem which is solved effectively using a longest path method [1] by representing the constraints in the form of a constraint graph [1]. As for the details of the method, please refer to Ref. [1].

Table 2: CPU times in second for placing circuits with 5, 6, and 7 blocks. AR; Aspect Ratio constraint CN; Critical Net constraint

Circuit	EX1	EX2	EX3	AR	CN
#Blocks	5	6	7		
#Nets	73	92	102		
#Critical Nets	5	7	8		
#I/O	2	2	2		
Condition 1	213	1066	—	No	No
Condition 2	28	47	10512	Yes	No
Condition 3	5	64	6202	No	Yes
Condition 4	1	28	1522	Yes	Yes

4 Experimental Results

We have examined the efficiency of the branch-and-bound process using several examples including the standard benchmark circuits supplied from MCNC. An experimental program is written in C and runs on a Sun4/330 and a DEC3100. In this section, we show the results on small examples and the benchmark examples, and compare them with those of other systems.

In order to examine the computational cost of the branch-and-bound process, we have made experiments using small examples with 5, 6, and 7 blocks which are subsets of *xerox* (BBL1) example. Table 2 lists the CPU second of a Sun4/330 for placing these examples under four different conditions. In Condition 1, we don't impose aspect ratio constraint nor critical net constraint, whereas we do both in Condition 4. Since the number of all the possible placements grows exponentially with the increase in the number of blocks, the computational cost of the branch-and-bound process also grows rapidly. In the case of EX3 example (7 blocks) under Condition 1, the search process does not terminate after 4 hours of computation. The table shows that constraints on a placement decrease the amount of CPU time to a great extent. The reason is that the constraints help to prune branches, which pruning results in the reduction of the computational cost of the branch-and-bound process.

The cost depends also on the uniformity of block shapes. When we place square blocks of the same sizes, even a five-block problem requires 15 min of CPU time under Condition 2. This is because the possibility of pruning associated with shape constraints(eg. aspect ratio, width, height, etc.) becomes low.

From the experiments described above, the maximum number of blocks which can be placed at a time in a reasonable amount of CPU time is around six. When the number of blocks in a problem exceeds the manageable limit, we need some method to reduce the

number of objects below the limit. A possible solution is to construct a placement in a bottom-up manner by clustering the blocks hierarchically. At each level of the hierarchy, the number of objects is kept below the limit. This method does not guarantee an overall optimal solution. We, however, expect to reach a good sub-optimal placement. We adopt this approach for problems with large number of blocks including the benchmark examples. For simplicity, we partition blocks and make clusters only on the basis of their connectivity. The details of the method for hierarchical placement is described in Appendix.

We have placed the standard benchmark circuits (*xerox, ami33, apte, ami49, hp*) provided by MCNC for the 1990 International Workshop for Layout Synthesis. In order to evaluate the quality of the results, we have routed them in detail by a commercial router. Power and ground nets are treated as signal nets. Although the router has capability of compacting/decompacting routing channels to complete routing, no significant modification occurred in the placements, which can be seen from their plots before and after detailed routing (Figs. 6–10). Table 3 summarizes the results for all the benchmark examples.

Xerox (BBL1) example contains 10 blocks and 203 nets, and the maximum length is specified for each of 16 nets. Also, the width of the chip is requested to be 4500 μm. We have placed the example in three different ways (Cases 1–3). In Case 1, all blocks were placed at a time subject to the critical net and width constraints. In Case 2, a placement was obtained hierarchically under both constraints. Case 3 also adopted hierarchical placement but no constraints were imposed. Figure 6 shows the results of Case 1. The result of Case 2 is essentially the same as that of Case 1 except for orientation(reflection) of two blocks. All the wire lengths of critical nets satisfy their requirements in both Cases 1 and 2. As a matter of course, Case 1 gives better results than Case 2, but the difference is not much. However, a big difference appears in computational costs. Case 1 required 2312 min of CPU time on a DEC3100 (although the final placement itself was obtained after 294 min), whereas Case 2 consumed only 2 sec. Without imposing any constraints on placement (Case 3), a placement with the smallest area was obtained after 4001 sec of computation.

Except for Case 1 of *xerox* example, all other examples were placed hierarchically. In the case of *ami33* example, which has 33 blocks and 123 nets, the placement consists of 5 clusters with 6 blocks and one cluster with 3 blocks. Figure 7 shows the results. The CPU time for each sub-problem, ie. placing blocks within a cluster and placing clusters, ranged 0.4 sec to 2520 sec. The total amount of CPU time was 5344 sec on a DEC3100.

The biggest circuit among the benchmarks is *ami49* example with 49 blocks and 408 nets. Since the example contains more than 36 blocks, it requires two levels of decomposition. Owing to the simple algorithm used for circuit decomposition that only considers connectivity, size difference among clusters becomes large in this example, which results in insufficient area usage. More sophisticated method of decomposition, which takes into account not only connectivity but also the size of each block, has to be considered.

The examples *xerox* and *ami33* have been widely used for benchmark purpose. In Table 4 we compare our results (listed under system "BB") with those of other systems. Although direct comparison is difficult since most of them were placed under different conditions, our results are quite satisfactory. They are the smallest for *ami33* example and very close to the smallest for *xerox* example.

Table 4: Comparison with other systems.

System	Circuit	Area [mm^2]	WL [mm]	Constraints considered
BB	xerox	26.60	566	Critical nets, Width
	xerox	26.17	628	No
	ami33	2.24	109	Aspect ratio
Delft†	xerox	26.21	576	No
	ami33	2.47	143	No
Bear [15]	xerox	28.47	633	P/G nets
	ami33	2.83	131	P/G nets
Seattle Silicon [16]	xerox	25.79	601	N/A
	ami33	2.42	91	N/A

†Delft results are taken from the 1990 International Workshop on Layout Synthesis. Placements are done manually.

5 Conclusion

We have presented a branch-and-bound method for building block placement. The method searches for an optimum solution subject to the constraints on critical nets and/or the shape of a resulting placement specified by a designer. The performance of the method has been examined experimentally. The maximum number of blocks that the method can handle in a reasonable CPU time is around six if no constraints are imposed on a placement. The manageable limit increases when tight constraints such as critical net constraints are imposed. This is because the constraints help to prune a decision tree, which results in the reduction of computational cost. For a problem which contains more blocks than the limit, we decompose the problem hierarchically and construct a placement in a bottom-up manner. At each level of the hierarchy, the proposed method is applied to each sub-problem. The method has been successfully applied to several examples including the benchmark circuits distributed by MCNC. The results for *xerox* and *ami33* examples are quite satisfactory. The result for *ami49* suggests that there exists some room for improving placement by refining the method of hierarchical decomposition, which is a subject for future work.

Acknowledgement - The authors would like to thank Prof. E. S. Kuh and Prof. C. K. Cheng for useful discussions.

References

[1] B. Preas and M. Lorenzetti, ed., *Physical Design Automation of VLSI Systems*, The Benjamin/Cummings Publishing Company, California, 1988.

[2] R. H. J. M. Otten, "Automatic Floor-plan Design," *Proc. 19th Design Automation Conf.*, pp. 261–267, June 1982.

[3] U. Lauther, "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," in *Proc. 16th Design Automation Conf.*, pp. 1–10, June 1979.

[4] G. J. Wipfler, M. Wiesel, and D. A. Mlynski, "A Combined Force and Cut Algorithm for Hierarchical VLSI Layout," *Proc. 19th Design Automation Conf.*, pp. 671–676, June 1982.

[5] H. Onodera, M. Sakamoto, T. Kurihara, and K. Tamaru, "Step by Step Placement Strategies for Building Block Layout," *Proc. 1989 Int. Symp. on Circuits and Systems*, pp. 921–926, 1989.

[6] D. Jepsen and C. Gelatt, "Macro Placement by Monte Carlo Annealing," in *Proc. 1983 IEEE Int. Conf. in Comp. Design*, pp. 495–498, Nov. 1983.

[7] C. Sechen, "Chip-Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing," in *Proc. 25th Design Automation Conf.*, pp. 73–80, June 1988.

[8] D. Wong and C. Liu, "A New Algorithm for Floor-plan Design," *Proc. 24th Design Automation Conf.*, pp. 101–107, June 1986.

[9] B. T. Preas and W. M. vanCleemput, "Placement Algorithms for Arbitrarily Shaped Blocks," in *Proc. 16th Design Automation Conf.*, pp. 474–480, June 1979.

Table 3: Experimental results for the benchmark circuits.

Circuit	xerox (BBL1) Case 1	Case 2	Case 3	ami33 (BBL2)	apte (BBL3)	ami49 (BBL4)	hp (BBL5)
#Blocks	10	←	←	33	9	49	11
#Nets	203	←	←	123	97	408	83
#I/O	2	←	←	42	73	22	45
#Critical Nets	16	←	0	0	0	0	0
Width Goal [μm]	4500	←	—	—	—	5000	2300
Aspect Ratio Goal	—	—	—	1	1	—	—
Width [μm]	4222	4278	6672	1480	7219	4692	2455
Height [μm]	6301	6325	3922	1513	7487	10976	4951
Aspect Ratio	0.67	0.68	1.70	0.98	0.96	0.43	0.50
Chip Area [mm^2]	26.60	27.06	26.17	2.239	54.05	51.49	12.15
Wire Length [mm]	566	602	628	109	460	1021	278

[10] W. M. Dai and E. S. Kuh, "Simultaneous Floor Planning and Global Routing for Hierarchical Building-Block Layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 828–837, Sept. 1987.

[11] H. Onodera, Y. Taniguchi, and K. Tamaru, "Branch-and-Bound Placement for Building Block Layout," *International Workshop on Layout Synthesis*, May. 1990.

[12] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa, "Generation of Performance Constraints for Layout," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 860–874, Aug. 1989.

[13] M. Schlag, Y. Z. Liao, and C. K. Wong, "An Algorithm for Optimal Two-Dimensional Compaction of VLSI Layouts," *Integration*, vol. 1, pp. 179–209, Sept. 1983.

[14] G. Kedem and H. Watanabe, "Graph-Optimization Techniques for IC Layout and Compaction," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 12–20, Jan. 1984.

[15] W. M. Dai, B. Eschermann, E. S. Kuh, and M. Pedram, "Hierarchical Placement and Floorplanning in BEAR," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 1335–1349, Dec. 1989.

[16] M. Upton, K. Samii, and S. Sugiyama, "Integrated Placement for Mixed Macro Cell and Standard Cell," *Proc. 27th Design Automation Conf.*, pp. 32–35, June 1990.

Appendix Method for Hierarchical Placement

Hierarchical Placement

In the current implementation of the branch-and-bound placement, the manageable number of blocks is experimentally found to be six if no constraints are imposed on a placement. We thus decompose a circuit into 6 sets of blocks (clusters) if the circuit contains more than 6 blocks. We set the maximum number of blocks in a cluster to N_c, where N_c is the minimum integer greater than or equal to $N_b/6$ (N_b represents the total number of blocks). When N_c is less than or equal to 6, ie. N_b is less than or equal to 36, one level of decomposition is sufficient. On the other hand, when N_c is 7 or more, there exists at lease one cluster which contains 7 or more blocks. The same procedure is recursively applied to such a cluster till the number of blocks in a cluster becomes 6 or less.

A placement is obtained in a bottom-up manner. At each level of the hierarchy, the branch-and-bound placement is applied to each sub-problem. In order to avoid extreme shape irregularity among clusters, which may cause unnecessary dead space when placing the clusters, we impose loose shape constraints on a placement for each cluster. In our experiment, we search for a placement that has an aspect ratio ranging from 0.5 to 2. When no solution exists due to severe constraints on critical nets, we relax the shape constraints until a solution is obtained. We then place the clusters under specified shape constraints to form a final solution.

Circuit Decomposition

The method we used for circuit decomposition is a simplified version of a metric allocation method [1]. Relying on the high ability for shape adaptation of the branch-and-bound placement, we partition blocks to form clusters only on the basis of their connectivity as described below.

We calculate the weight of each net w_p as

$$w_p = \begin{cases} \frac{1}{t_p - 1} & \text{for non-critical nets} \\ \frac{10}{t_p - 1}\frac{L}{L_p} & \text{for critical nets} \end{cases}$$

where t_p represents the number of pins that net p connects, L_p is the maximum allowable length for critical net p, and L is the maximum of the maximum allowable lengths for critical nets. A critical net is weighted at least 10 times than a non-critical net. A critical net with shorter allowable length is weighted more, which enhances the possibility of accommodating blocks connected with the critical net into the same cluster. Using the weight, we define the connectivity W_{ij} between blocks i and j as

$$W_{ij} = \sum_{k \in N_{ij}} w_k$$

$$(i, j = 1, 2, \ldots, N_b; i < j)$$

where N_b represents the number of blocks and N_{ij} stands for the set of nets spanning blocks i and j.

We decompose the circuit into six clusters based on the connectivity W_{ij} in the following manner.

(1) Assign each block to a cluster. The number of cluster is N_b. $C(i)$ is the cluster that block i belongs to.

(2) Repeat the following for each pair of blocks i and j in decreasing order of the connectivity W_{ij}.

> If $C(i) \neq C(j)$ and the total number of blocks in $C(i)$ and $C(j)$ is less than or equal to N_c, then merge the two clusters. If the number of clusters is 6, then end.

(3) If the number of clusters is 6, then end.

(4) Take the smallest cluster (the cluster with the smallest number of blocks), and attempt to merge it with any of the largest 6 clusters. If the total number of blocks after merging is 6 or less, then merge the two clusters, and go to Step 3.

(5) Break the smallest cluster into two clusters, and go to Step 4.

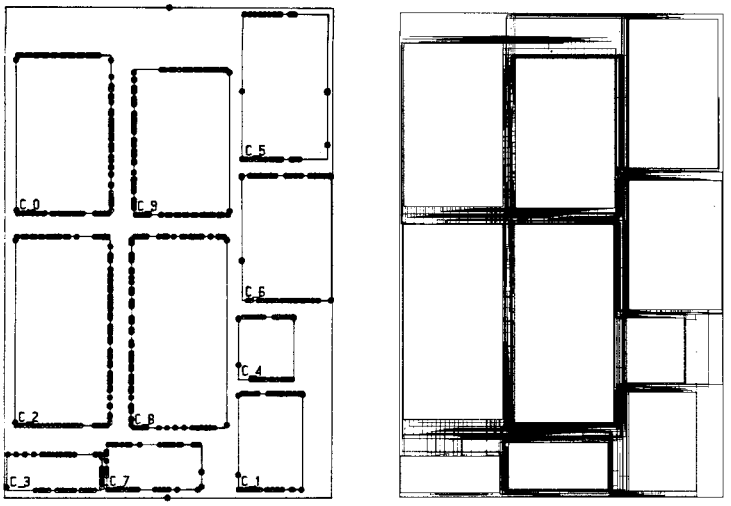

Figure 6: *Xerox*(BBL1) benchmark example.
Left; after placement. Right; after detailed routing.

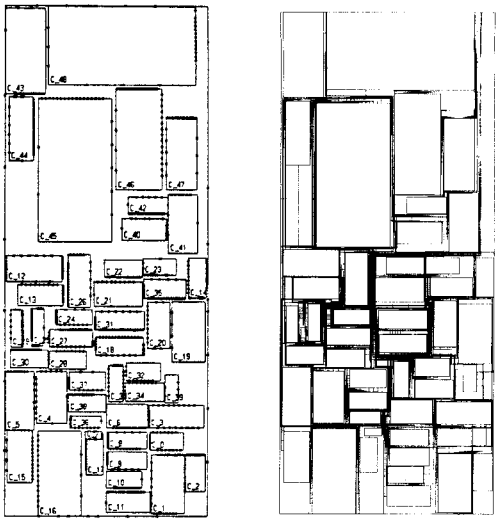

Figure 10: *Hp*(BBL5) benchmark example.
Left; after placement. Right; after detailed routing.

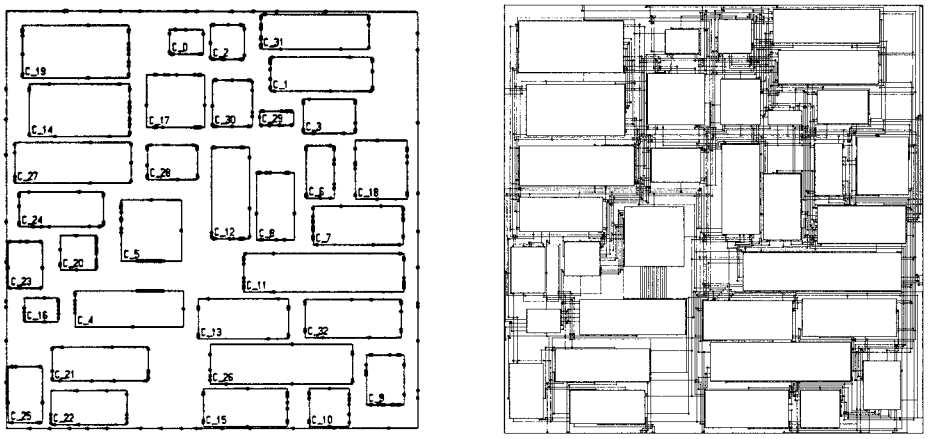

Figure 7: *Ami33*(BBL2) benchmark example.
Left; after placement. Right; after detailed routing.

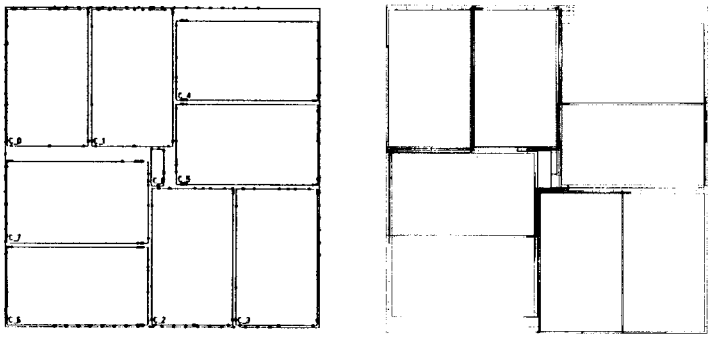

Figure 8: *Apte*(BBL3) benchmark example.
Left; after placement. Right; after detailed routing.

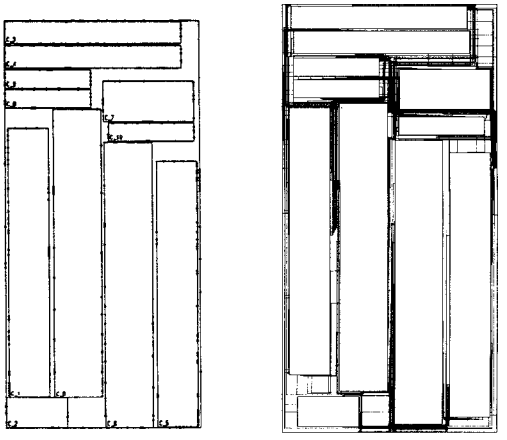

Figure 9: *Ami49*(BBL4) benchmark example.
Left; after placement. Right; after detailed routing.