ECBL: An Extended Corner Block List with Solution Space including Optimum Placement

Shuo Zhou, Sheqin Dong, Xianlong Hong, Yici Cai
Dept. of Computer Science and Technology, Tsinghua University, Beijing 100084 P.R. China
zhoushuo99@mails.tsinghua.edu.cn

Chung-kuan Cheng
Department of Computer Science and Engineering, University of California, San Diego La Jolla, CA 92093-0114, USA

Jun Gu
Department of Computer Science, Science & Technology University of Hong Kong

ABSTRACT
A Non-Slicing floorplanning algorithm based on CBL[1], corner block list, was presented recently. It can represent non-slicing floorplans without empty rooms. In this paper, we propose an extended corner block list structure, ECBLλ, to represent general non-slicing floorplans, which may include empty rooms. By setting \( \lambda \in [1..3] \), where \( \lambda \) is the extending ratio, our algorithm can translate a topological floorplan to its corresponding placement in \( O(n) \) time, where \( n \) is the number of blocks. Also, based on the optimum solution theorem of bounded-sliceline grid in [2], we proved that the solution space of ECBLλ contains the optimum block placement, which has the minimum area. Experimental results on MCNC benchmarks show promising performance with 7% improvement in wire length and 2% decrease in dead space over algorithms based on CBL. Meanwhile, compared with other algorithms, our algorithm can get better results with less runtime.

1 INTRODUCTION
Floorplaning[3] is an important step in physical design of VLSI circuits. One of the key factors of most floorplanners is the representation structure, which affects the effectiveness and the efficiency of the optimization algorithm directly. The reason is that the representation structure determines the complexity of constructing a placement from a topological representation, which is the basic operation in a stochastic algorithm.

Two categories of floorplan, slicing structure and non-slicing structure, are identified. For a slicing floorplan, we can use a binary tree representation [4]. But it covers a limited design space. For non-slicing structure, many solutions have been proposed. In [2],[5], two topological representations, sequence pair and bounded-sliceline grid, were proposed to handle non-slicing floorplan. It is proved in [2] that the solution space of BSG\( _{op} \) includes the optimum placement. But their time complexity of transforming a topological representation to its corresponding placement is too high.

In [6], J. Xu proposed an iterative approach to optimize area and interconnection by cluster refinement. This approach is CPU-intensive and difficult to handle with a large cluster size. Guo at el[7] presented an ordered tree to represent non-slicing floorplans. It has very small solution space and \( O(n) \) complexity[8], but it is not a topological representation.

Recently, a non-slicing approach called corner block list has been proposed. The run-time for transforming a corner block list to its corresponding placement is linear with respect to the number of blocks, i.e. \( O(n) \). But it represents a subset of non-slicing floorplan. Given an \( n \)-block set, it divides the chip into \( n \) rooms and assigns one and only one block to each room. Therefore, floorplans with more than \( n \) rooms, some of which are empty, cannot be represented. Due to this limited solution space, the final floorplan may not be optimal.

In this paper, we propose an extended corner block list, ECBL\( _{\lambda} \), which can represent more general floorplan by extending the corner block list with an extending factor \( \lambda \). Like corner block list, ECBL\( _{\lambda} \) represents a floorplan by drawing rectangle dissection and assigning blocks to each room. But it allows more than \( n \) rooms, some of which are empty rooms. In block assignment, a false block with zero width and height is assigned into the empty room. Based on experimental results, we set the extending factor \( \lambda \) as a real number between 1 and 3. For a constant \( \lambda \), the ECBL\( _{\lambda} \)-based algorithm maintains the \( O(n) \) time complexity. It has also been proven that each BSG\( _{op} \)-based assignment can be represented by an ECBL\( _{\lambda} \)-based algorithm in this paper. So the solution space of ECBL\( _{\lambda} \) must contain the optimum block placement since there exists a BSG\( _{op} \)-based packing corresponding to the optimum placement[2]. To test the efficiency of ECBL\( _{\lambda} \), we apply it on MCNC benchmarks and compare the experiment results with algorithms based on other structures [1],[6],[7],[8]. Our algorithm always gets better results with comparable less CPU time for different objective functions.

The rest of this paper is organized as follows. Section 2 reviews the floorplan by CBL. Section 3 defines ECBL\( _{\lambda} \) structure and presents an \( O(n) \) complexity algorithm by analyzing the relation between \( \lambda \) and the quality of experiment results. Section 4 proves that optimum floorplan is included in ECBL\( _{\lambda} \). A reasonable range for the parameter \( \lambda \) is also given in this section. Experiment results are reported in Section 5, followed by concluding remarks in section 6.

2 CORNER BLOCK LIST
Corner Block List represents floorplan by a triple list of \((S, L, T)\). It divides the chip into rectangular rooms and assigns one and

*This work is supported by National Science Foundation(No. 60076016) and 973 National Key Project (No.G1998030403)
only one block to each room according to (S, L, T), where S stands for block assignment, L and T stand for orthogonal line segment. This section describes the floorplan, which CBL can represent, and the algorithms to transform between floorplan and corner block list.

2.1 MOSAIC Floorplan
In [1], a class of floorplan named MOSAIC Floorplan, is defined. A floorplan belongs to this class if and only if it observes the following properties: 1) Floorplan of No Empty Space. There is no empty space in the floorplan, i.e. each room is assigned one and only one block. 2) Topological Equivalence on Segment Sliding. The topology is defined to be equivalent before and after the non-crossing segment of the T-junction slides (Figure 1 (a) and (b)). 3) Non-Degenerate Topology. There is no degenerate case where two distinct T-junctions meet at the same point. If a degenerate situation happens, we separate two T-junctions by sliding a non-crossing segment of a T-junction by a small distance.

CBL can represent MOSAIC Floorplan only.

2.2 Constraint Graph And Corner Block
A constraint graph [2][7] for a floorplan is a graph \( G = (V, E) \), where the nodes in \( V \) are segments which slice the space and form the rooms of the floorplan with additional nodes used for boundaries of the placement, and the edges in \( E \) are the rooms of placement blocks. The edges in \( E \) are directed. There are two kinds of edges: horizontal edges and vertical edges. There are two kinds of constraint graphs: the horizontal constraint graph (HCG) and the vertical constraint graph (VCG). Nodes in HCG include those representing the vertical segments, a west pole “W” and an east pole “E” representing the left and right boundary of the chip. Edges in HCG are horizontal edges, which represent the horizontal relation directed from left to right. Adding a south pole “S” and a north pole “N”, the VCG is defined similarly. Figure 2 illustrates the corresponding horizontal constraint graph and vertical constraint graph of the floorplan in figure 3 (a).

For an edge that points to east or north pole, we define it to be a corner edge. The block that its two constraint edges are corner edges in both HCG and VCG is defined to be a corner block. For example, in figure 3 (a), “c” is a corner block, while “a”, “b”, “d” and “e” are not. There exists a unique corner block in a mosaic floorplan of one or more blocks.

In a no empty space floorplan, the internal segments intersect and form T-junctions. A T-junction is composed of two segments: a non-crossing segment and a crossing segment. The non-crossing segment has one end touching point in the interval of the crossing segment. We define the orientation of a corner block according to the joint of its left and bottom segments and the T-junction containing the joint. At this point, T-junction has only two kinds of orientations: T rotated by 90 degrees counterclockwise and by 180 degrees counterclockwise. If T is rotated by 90 degrees counterclockwise, we define the corner block to be vertical oriented, and it is denoted by a “0”. Otherwise, we define the corner block to be horizontal oriented, and it is denoted by a “1”. For example, in figure 3 (a), the orientation of corner block “c” is vertical and is denoted by “0”.

![Figure 1 Mosaic Equivalent](image1)

![Figure 2 HCG&VCG](image2)

2.3 Transform Between CBL And Floorplan
The block deletion is used to construct the triple list of (S, L, T) corresponding to a mosaic floorplan. The block insertion is used to divide the chip and assign blocks to rooms according to (S, L, T).

2.3.1 Block Deletion
A corner block of a mosaic floorplan is deleted as following: If it is horizontal oriented, we shift its left segment to the right boundary of the chip and pull the attached T-junctions along with the segment. If it is vertical oriented, we shift its bottom segment to the top boundary of the chip, and pull the attached T-junctions along with the segment. After deletion, the HCG and VCG should be reconstructed according to the new floorplan. Figure 3(a) and 3(b) illustrate the deletion of corner block “c”.

Because the revised floorplan after corner block deletion remains to be mosaic, we can operate \( n \) times corner block deletion until all blocks are deleted. For each block deletion, we keep a record of block name, corner block orientation and the number of T-junctions it uncovered. We use binary T subsequence to record the number of attached T-junctions of the deleted corner block. The number of 1s corresponding to the number of attached T-junctions it uncovered. We use binary T subsequence to record the number of attached T-junctions of the deleted corner block. The number of 1s corresponding to the number of attached T-junctions. Each string of 1s is ended with a 0. For the last block deleted, the orientation and the T-junction information do not matter. So only block name is recorded, and the orientation and number of T-junction are not included in L and T. At last, we concatenate the data of these three items in a reversed order. Thus, we have a sequence S of block names, a list L of orientations, and a list T of T-junction information. The complexity of the deletion algorithm is \( O(n) \), where \( n \) is the number of blocks.

2.3.2 Block Insertion
A corner block is inserted as the insertion process of deletion: If the inserting corner block is vertical oriented, we push down the horizontal segment at the topside of the chip covering a designated set of T-junctions, and then get a room for the inserting corner block. If the inserting block is horizontal oriented, we push left the vertical segment at the right of the chip. Figure 3 (b) and 3(c) illustrate the insertion of corner block “c”.

The complexity of the insertion algorithm \( O(n) \), where \( n \) is the number of blocks.
From the insertion algorithm, a floorplan is constructed by dividing the chip into rooms according to S, L and T. S determines the times of block insertion, which equals to the number of rooms in the floorplan. For each block insertion, a segment is formulated, which slices the space and forms a new room. L determined the segment orientation. T determined the constraint relations between the new room and the existing rooms.

### 3 Extended Corner Block List

Corner Block List structure can represent mosaic floorplan, which includes \( n \) rooms with one and only one block in each room, where \( n \) is the number of blocks. However, area usage and interconnect may be affected by this stringent assumption. For example in figure 4, given a block set including four blocks, the floorplan of five rooms (figure 4a), has less area than that of four rooms (figure 4b). Apparently the floorplan of figure 4 (a) cannot be represented by a corner block list. So it is necessary to make some extension to corner block list.

In this section we propose the Extend Corner Block List, which can represent floorplan with more than \( n \) rooms. We also revise the deletion and insertion algorithm of corner block list and analyze the time complexity of the revised algorithm.

**Algorithm 3.1 Deletion:**

Add a false block to each empty room of the floorplan;

While there is a corner block available, repeat

1. Delete the corner block.
2. If the corner block is not a false block, record (block name, orientation, T-subsequence). Otherwise, record (FB, orientation, T-subsequence).

Concatenate all records in a reverse order of deletion to construct the list \( \langle S, L, T \rangle \).

If there are \( \lceil \lambda n \rceil \) rooms in the floorplan, the complexity of this algorithm is \( O(\lceil \lambda n \rceil) \). The number of false blocks in \( S \) equals to the number of empty rooms in the floorplan. Figure 5 illustrates the deletion procedure of a floorplan including empty rooms.

**Definition [ECBL](: Extended Corner Block List]:** Given a \( n \)-block set, an Extended Corner Block List with an extending factor \( \lambda \), denoted as \( ECBL_{\lambda} \), is a triple \( (S, L, T) \) generated by applying the deletion algorithm 3.1 over a floorplan including \( \lceil \lambda n \rceil \) rooms, among which are \( (\lceil \lambda n \rceil - n) \) empty rooms.

### 3.2 From ECBL To Floorplan

Given an \( ECBL_{\lambda} \), we can use the same insertion operation described before to insert FB and real blocks. When all the \( n \) real blocks are inserted, the algorithm will terminate no matter how many false blocks in \( S \) are left. Figure 6 illustrates the procedure.

**Algorithm 3.2 Insertion:**

Initialize the floorplan with block \( S[1] \);

For \( i = 2 \) to \( \lceil \lambda n \rceil \) do

- If all the real blocks are inserted
  - Exit and output the floorplan
- Else
  - Insert block \( S[i] \) with orientation \( L[i] \) and covering the T-junctions according to the record in list \( T \).

If the number of covering T-junctions is more than the T-junctions available, exit and report error.

### 3.3 The Complexity

According to Theorem 3.6 in [1], if we divide the chip into \( \lceil \lambda n \rceil \) rooms, we have sequence \( S \) with \( \lceil \lambda n \rceil \) blocks and orientation list \( L \) with \( \lceil \lambda n \rceil -1 \) bits, while list \( T \) has no more than \( 2\lceil \lambda n \rceil -3 \) bits. We derive the number of combinations of \( ECBL_{\lambda} \) as following.

**Theorem 1:** The number of combinations of \( ECBL_{\lambda} \) can be expressed as \( O(C^n_{\lceil \lambda n \rceil}2^{\lceil \lambda n \rceil-3}/\lceil \lambda n \rceil^{\lceil \lambda n \rceil}) \).

**Proof:** If the assignment of all the real blocks is not changed, \( S \) is not changed. So the number of combinations of \( S \) is selecting \( n \) rooms from \( \lceil \lambda n \rceil \) rooms and assigning \( n \) real blocks in permutation. There are \( 2\lceil \lambda n \rceil -4 \) bits in \( L \) and \( T \). Each bit can be 0 or 1. So the number of combinations of \( L \) and \( T \) is \( 2^{2\lceil \lambda n \rceil-4} \).

**Theorem 2:** The complexity of algorithm 3.2 for \( ECBL_{\lambda} \) is \( O(\lceil \lambda n \rceil) \).

**Proof:** To insert block \( S[i] \), we will search \( T[i] \) block on the top or right boundary, where \( T \) is the corresponding \( T \) subsequence. So the complexity of inserting all the blocks is linear to \( \sum_{i=1}^{\lceil \lambda n \rceil} T[i] \).

That is the length of sequence \( T \), which is \( 2\lceil \lambda n \rceil -3 \) at most.

Theorem 2 indicates that the complexity of deletion algorithm 3.2 is affected by the extending factor \( \lambda \), which determines the number of false blocks added. In section 4, we will show that we can get fairly good results by setting \( \lambda \in [1.5 .. 3] \), according to experiments on MCNC benchmarks shown in figure 8 and figure 9. When \( \lambda \) is determined as a constant, the complexity of our algorithm is reduced to \( O(n) \).
3.4 New Solution Generation
Because the number of combinations of ECBL₃ is very large, the simulated annealing algorithm is used to generate the new solution. Each new solution is generated in two steps:

Step1. A new triple list (S, L, T) is generated in any of following two ways: 1) Exchange two blocks (maybe FB or real block) in sequence S randomly, i.e. block assignment is changed. 2) Change some bits in sequence L and T, i.e. the topological relation of rooms is changed.

Step2. Use algorithm 3.2 to transform the new triple list into the new floorplan and get the chip area.

The initial solution of the algorithm is given randomly.

4 THE OPTIMUM SOLUTION SPACE
Because the size of the solution space of extended corner block list is determined by λ, two problems arise: 1) whether there exist a λ, whose solution space includes the optimum solution? If there is such λ, then 2) how much is it?

As a structure based on meta-grid, the bounded-sliceline grid structure has some similarity with extended corner block list. We will prove that a BSG-based packing can be represented by an extended corner block list structure through a transformation. Based on optimum solution theorem of bounded-sliceline grid in [2] and the transformation, we will get a λ and the corresponding solution space of ECBL₃ including optimum solution.

In this section, we review the bounded-sliceline grid first, then we will prove the theorem of optimum solution space.

4.1 Bounded-Sliceline Grid
A BSG, Bounded-Sliceline Grid, consists of horizontal and vertical bounded length lines, called the BSG-segs, which dissect the plane regularly into rooms, called the BSG-rooms. Each BSG-room is bounded by a pair of adjacent half-shifted horizontal BSG-segs, and a pair of adjacent half-shifted vertical BSG-segs. The BSG consisting p rooms horizontally and q rooms vertically is denoted as BSGₚₓₚ. Figure 7 (a) illustrates BSG₃×₃. The topological relation of BSG-rooms can be represented by the HCG and VCG defined before. Figure 7 (b) and (c) is the HCG and VCG of BSG₃×₃ in figure 7 (a).

The BSG-based packing algorithm consists of two steps:

Step1. Blocks are assigned to the BSG-rooms, at most one block to one BSG-room. A room to which no block is assigned is empty room.

Step2. If a room is assigned with a block, the weight of the corresponding edge in HCG is the width of the block. Analogously, edges of VCG are weighted according to the height of the modules. Edges corresponding to empty rooms are weighted 0. The physical position of each block is the longest path in HCG and VCG from source node to the node corresponding to that block.

Figure 7 (b) is the BSG-based packing of assignment in figure 7 (a). It is proved that the time complexity of a BSG-based packing algorithm is O(pq). Paper [2] has proved the theorem of optimum solution: There exists an assignment that defines packing of the minimum area, if the size of the BSG is not smaller than \( p \times q \).

Figure 7 BSG-based packing

4.2 From BSG-Based Assignment To ECBL
According to observation on the BSG, we conclude that a BSG-based assignment is a floorplan, which can be represented by an extended corner block list, based on following reasons:

1) A BSG dissect the chip into BSG-rooms by BSG-segs and represents the topological relation of the BSG-rooms by HCG and VCG. One and only one block is assigned to each BSG-room. If the number of BSG-rooms in a BSG is larger than the number of blocks, some rooms are left empty after block assignment. After false blocks are assigned to empty rooms, the corner block of the BSG can be determined based on HCG and VCG.

2) The left and bottom BSG-seg of a BSG-room form horizontal or vertical oriented T-junctions as defined in section 2. We can define the direction of a BSG-room according to the direction of its T-junctions. If a BSG-room has horizontal T-junctions, its direction is horizontal. Otherwise it is vertical.

Since the corner block and its direction can be determined, a BSG-based assignment can be mapped to an extended corner block list through algorithm 3.1. So we have theorem3 below.

Theorem 3: A BSGₚₓₚ-based assignment can be represented by an ECBLₚₓₚ

Because the number of BSG-rooms in BSGₚₓₚ is \( p \times q \), the number of blocks in S after deletion will be \( p \times q \). Then the extending factor \( λ = \frac{p \times q}{n} \). The BSGₚₓₚ-based assignment in figure 7 (a) can be represented by the ECBL = \{d FB₁, a FB₂, b FB₃, c FB₄, d FB₅, c, b, E, d, c, FB₁, FB₂, FB₃, FB₄\}, 10011001, 010010101000).

From the BSG structure, we can get two properties of the mapping from BSG-based assignment to an ECBLₚₓₚ.

Property 1: The mapping between BSGₚₓₚ-based assignment and ECBLₚₓₚ is not one to one.

Because a BSG-room covers at most one T-junction, the T subsequence derived from BSG-room deletion is limited to be 0 or 10. But for ECBLₚₓₚ, we do not have such limitation, i.e. some ECBLₚₓₚ does not represent any BSGₚₓₚ-based assignment. So for BSG-based assignment and an ECBL representation including the same number of rooms, the number of combinations of ECBL is larger than that of BSG.

Property 2: When \( p \) and \( q \) is definite, different BSGₚₓₚ-based assignment can be represented by ECBLₚₓₚ which only differs in the permutation S of Corner Block List (S, L, T).

In ECBLₚₓₚ, L and T only determine the topological relations of rooms, and S determines the number of rooms and the block
assignment. When \( p \) and \( q \) is definite, both the number of BSG-rooms and the topological relations of these rooms are definite, i.e., only the block assignment can be changed. So for different BSG-based assignment with definite \( p \) and \( q \), \( L \), \( T \) and the number of blocks in \( S \) are definite, the order of blocks in \( S \) is different.

4.3 Optimum Solution And Selection Of \( \lambda \)

With the mapping from BSG-based assignment to ECBL, we can determine the \( \lambda \), so the solution space of ECBL\( _\lambda \) includes the optimum solution based on the optimum solution theorem in [2].

**Theorem 4**: Given \( n \) blocks, there exists a solution in the solution space of ECBL\( _{\lambda n} \), which represents a floorplan with the minimum area.

**Proof**: According to the optimum theorem [2], there exists an assignment on BSG\( _{\lambda n} \) that defines a packing of the minimum area. Based on Theorem 3, this assignment can be represented by an ECBL\( _n \).

Although the solution space of ECBL\( _n \) includes the optimum solution, it is impractical to set \( \lambda = n \). First, the solution space of ECBL\( _{\lambda n} \) is too large for a simulated annealing method, according to the number of combinations given in theorem 1. Second, the time complexity of transforming a representation to a placement is also affected by \( \lambda \). When \( \lambda \) equals \( n \), the complexity is \( O(n^2) \).

Then, we have to determine the feasible \( \lambda \) both for performance and running time. Paper [2] has proven that by setting \( p \) and \( q \) to some integers less than \( n \), BSG\( _{pq} \) can give fairly good results. If we set \( \lambda \) as \( \lceil pq/n \rceil \) according to the \( pq \) given in paper [2], we would also get some good results with the mapping from BSG-based assignment to ECBL. Because the number of rooms in ECBL\( _{pq/n} \) and BSG\( _{pq} \)-based assignment is both \( pq \), the algorithms based on them have the same complexity \( O(pq) \).

However, we would also get fairly good results by setting \( \lambda \) less than \( \lceil pq/n \rceil \). That is to say, we can get comparable good results in shorter time by ECBL\( _{\lambda n} \), which includes rooms less than \( pq \). We make such conclusion according to **property 1** and **property 2**.

The BSG\( _{pq} \)-based simulated annealing algorithm set the size of BSG with \( p \) and \( q \). According to **property 2**, with definite \( p \) and \( q \), only the block assignment is changeable. The ECBL\( _{\lambda} \)-based simulated annealing algorithm set the size of ECBL with \( \lambda \). With a definite \( \lambda \), the number of rooms is definite. We can change the block assignment by changing the block order in \( S \). Also we can modify the topological relation between the rooms by changing \( L \) and \( T \).

For an ECBL representation and a BSG-based assignment with the same number of rooms, the simulated annealing algorithm based on ECBL can search a larger solution space. However, the algorithms, which transform these representations to placements, have the same time complexity.

Figure 8 illustrates the experimental results using the same amount of iteration to search the solution. Apparently, when \( \lambda = 1.5 \), the quality of results is much better than \( \lambda = 1 \). But if \( \lambda \) is larger than \( 4 \), the performance of the result is not better, but worse. Theoretically speaking, larger \( \lambda \) will increase the possibility of the solution space including an optimum solution. However, the solution space becomes huge. Therefore we cannot find a good result in practical runtime. Figure 9 illustrates the relationship between the solution quality and runtime when \( \lambda \) becomes larger. Compared with ECBL\( _2 \), ECBL\( _3 \) needs longer running time to get comparable good results. Based on these experiments, we set \( \lambda \in [1.5, 3] \). Since \( \lambda \) is such a small constant, the complexity of our algorithm is reduced to \( O(n) \).

5 EXPERIMENTAL RESULTS

We have implemented the floorplanning and placement algorithm in C programming language. All experiments are performed on a SUN spark20 workstation. Some MCNC benchmarks are used in the experiments. Our experiments do not include soft blocks. \( \lambda \) of ECBL is set between 1.8 and 2.5 in all of the experiments.
Table 1 objective function is area, $\lambda \in [1.8 .. 2.5]$

<table>
<thead>
<tr>
<th>Circuits</th>
<th>ECBL</th>
<th>Aver area/time</th>
<th>Min area/time</th>
</tr>
</thead>
<tbody>
<tr>
<td>apte</td>
<td>45.93/3</td>
<td>46.86/3</td>
<td>45.93/3</td>
</tr>
<tr>
<td>xerox</td>
<td>20.15/3</td>
<td>20.15/3</td>
<td>19.91/3</td>
</tr>
<tr>
<td>hp</td>
<td>9.140/11</td>
<td>9.140/11</td>
<td>8.918/11</td>
</tr>
<tr>
<td>Ami33</td>
<td>1.208/42</td>
<td>1.208/42</td>
<td>1.192/73</td>
</tr>
<tr>
<td>Ami49</td>
<td>37.36/64</td>
<td>37.36/64</td>
<td>36.70/117</td>
</tr>
</tbody>
</table>

Table 2 compared with other structures: CBL(Sun sparc20), O-trees (Sun Ultra60), Cluster (Sun Sparc20, size = 4 blocks)

<table>
<thead>
<tr>
<th>Circuits</th>
<th>ECBL</th>
<th>CBL</th>
<th>O-tree</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>area/time</td>
<td>area/time</td>
<td>area/time</td>
<td>area/time</td>
</tr>
<tr>
<td>apte</td>
<td>45.93/3</td>
<td>47.27/25</td>
<td>46.92/11</td>
<td>48.42/223.8</td>
</tr>
<tr>
<td>xerox</td>
<td>19.91/3</td>
<td>20.96/30</td>
<td>20.21/38</td>
<td>20.30/18.8</td>
</tr>
<tr>
<td>Ami33</td>
<td>1.192/73</td>
<td>1.242/119</td>
<td>1.207/603.4</td>
<td></td>
</tr>
<tr>
<td>Ami49</td>
<td>36.70/117</td>
<td>37.68/65</td>
<td>37.73/526</td>
<td>37.69/1861.7</td>
</tr>
</tbody>
</table>

Table 1 is block placement results, using area as the objective function. Table 2 reports the placement results on ECBL and other structures. Compared with O-tree and Cluster refinement, it can get better results in shorter time. Compared with CBL, the run time of ECBL is longer. But even the run time of CBL is lengthened as that of ECBL, the performance of CBL can not be improved. Here, we do not give the results based on BSG, because there are not such results on MCNC experiments in published papers. Figure 10 illustrates a placement result of ami33.yal. Figure 11 illustrates an example including 147 modules.

Table 3 shows placement results, using both area and total wire length with different weight as the objective function. Here, the wire length is the sum of half-bounding box of all nets in the circuit. Compared with CBL, the total wire length is improved over 7%, and the decrease of dead space is over 2%. Figure 12 illustrates a placement result of ami49.yal with equal weight for the area and total wire length.

6 CONCLUSION
We have proposed the extended corner block list structure, ECBL$_\lambda$, by enlarging the solution space of the corner block list, where $\lambda$ is the enlarging ratio. The extended corner block list can represent a kind of floorplan including empty room, which corner block list cannot represent. However, by setting factor $\lambda$, the ECBL$_\lambda$ based-algorithm has the same complexity as that of corner block list. Based on bounded-sliceline grid structure, we proved that the solution space of ECBL$_\lambda$ includes the optimum solution, where $n$ is the number of blocks. The experiment results demonstrate that the algorithm based on the extended corner block list is quite promising.

7 REFERENCES

Table 3 Minimum/average area and wirelength distribution with different weights

<table>
<thead>
<tr>
<th>circuits</th>
<th>Area with high weight</th>
<th>Equal weight</th>
<th>Wire length with high weight</th>
<th>Improved ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>area</td>
<td>wire</td>
<td>area</td>
<td>wire</td>
</tr>
<tr>
<td>Ami33</td>
<td>1.20/1.21</td>
<td>51.92/56.51</td>
<td>1.22/1.22</td>
<td>50.76/53.73</td>
</tr>
<tr>
<td>Ami49</td>
<td>37.34/38.14</td>
<td>792.92/1023.16</td>
<td>38.15/38.77</td>
<td>721.06/775.56</td>
</tr>
</tbody>
</table>