Stairway Compaction using Corner Block List and Its Applications with Rectilinear Blocks

Yuchun Ma[1], Xianlong Hong[1], Sheqin Dong[1], Yici Cai[1], Chung-Kuan Cheng[2], Jun Gu[3]
[1]*Department of Computer Science and Technology, Tsinghua University, Beijing, China*
Email: {hxl-dcs}{dongsq}@tsinghua.edu.cn
[2]*Department of Computer Science and Engineering, University of California,San Diego*
La Jolla,CA 92093-0114,USA
[3]*Department of Computer Science, Science & Technology University of HongKong*

Abstract

Corner Block List(CBL) was recently proposed as an efficient representation for MOSAIC packing of rectangles. Although the original method is really innovative, there still remains room of improvement for our purpose. This paper proposes a compact algorithm for placement based on corner block list. By introducing the dummy blocks in CBL, our algorithm can intellective employ dummy blocks in the packing to represent the placement including empty rooms, which corner block list cannot represent. Our algorithm can obtain the fast convergence to an optimal solution. Based on the compact approach, we propose a new way to handle arbitrary shaped rectilinear modules. The experimental results are demonstrated by some benchmark data and the performance shows effectiveness of the proposed method.

1. Introduction

Building block placement is becoming more and more important for VLSI physical design, because circuit sizes are growing rapidly and hierarchical design with IP blocks is now widely used to reduce the design complexity. Two categories of placement, slicing[6] and non-slicing, are identified. For general placement including both slicing and non-slicing, several encoding schemes were recently proposed, namely, SP[1], BSG[2], O-tree[3], Corner Block List(CBL)[5], B*-tree[4], and TCG[17]. All of them except O-tree and B*-tree employ topological representations of placement configurations, where cell positions are specified based on encoded topological relations.

Corner block list was recently presented as an efficient topological representation. Compared with the previous representations, CBL has a smaller upper bound on the number of possible configurations, needs only linear computation effort to generate a corresponding placement and decreases the redundancies in the previous representations. Although the original method is really innovative, there still remains room of improvement for our purpose. Given an n-block set, it divides the chip into n rooms and assigns one and only one block to each room. Therefore, there is a certain kind of packings which can not be represented as shown in Fig.1.

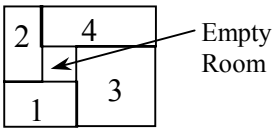


Figure 1. A packing with empty room

Recently, Extended Corner Block List (ECBL)[8] extends CBL representation by adding certain amount of empty blocks. But the extending factor λ takes great effect to the final result and the run-time. The run-time for transforming $ECBL_\lambda$ to its corresponding placement is λ times to the run-time for CBL. To represent general non-slicing floorplans, n^2-n empty rooms should be added and the computation complexity increases to $O(n^2)$ since the total number of the blocks is n^2.

In this paper, we devise a compact algorithm based on CBL to compact the placement as much as possible at the same time of the packing process. The empty rooms are added into the given CBL determinately to obtain a tight packing and the number of the empty rooms is no more than n-3. Unlike the algorithm of O-tree, whose compact process is a post process which takes repeated transformations until the convergence is achieved, our algorithm of the compact process is embedded in the process of packing and the computation complexity of the translation from a given CBL to its corresponding compact placement remains $O(n)$.

With recent advent of deep submicron technology and new packaging schemes, integrated circuit components are not limited to rectangular blocks. Several methods using slicing structure[16], and nonslicing structure such as SP[13,14,15], BSG[12] and O-tree[11] have been proposed. In this paper we extend the stairway compact approach based on CBL representation to handle the placement with arbitrary shaped rectilinear blocks. Different with the previous methods, our algorithm has several properties: 1) **Directness,** for each CBL there are

a modified CBL corresponding to a feasible packing with rectilinear blocks; 2) **Efficiency**: the transformation from a modified CBL to its placement takes only linear time. It is not necessary to do any operations to restore the shapes of the rectilinear blocks since their shapes are maintained while packing. The experimental examples prove the effectiveness of our approach.

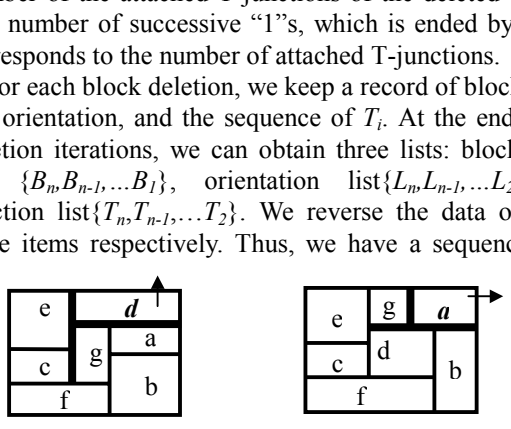
The rest of the paper is composed as follows: Sect.2 is a brief review of the CBL model. The compact algorithm is presented in Sect.3. Sect.4 extends the compact algorithm to handle the packing with rectilinear blocks. The experimental results are shown in Sec.5. Finally, the conclusion is given.

2. CBL Representation

CBL is derived from a simplified version of general placement called mosaic structure, which have no empty space and the block is represented by the room with only topological relationship between each other. CBL represents the topological relations in mosaic structure by a triple list of (S, L, T). It divides the chip into rectangular rooms and assigns one and only one block to each room according to (S, L, T).

The Corner Block(CB) is the block packed at the upper right corner of the placement. The joint of the left and bottom segments of the CB is contained in a T-junction named corner T-junction and the CB's orientation is defined by the orientation of the corner T-junction. The T-junction has only two kinds of orientations: **T** rotated by 90°(Fig.2(a)) and by 180°(Fig.2(b)) counterclockwise respectively. If **T** is rotated by 90° counterclockwise, we define the CB to be vertical oriented, and denote it by a "0". Otherwise, the CB is horizontal oriented, and denote it by a "1". The CBL is constructed from the record of a recursive CB deletion. In Fig.3, the CB *d* is deleted and the attached T-junctions, whose crossing segments are the non-crossing segment of corner T-junction, are pulled up to the top boundary of the chip. The insertion of CB is the inverse of the deletion. We use a binary list T_i to record the number of the attached T-junctions of the deleted CB M_i. The number of successive "1"s, which is ended by a "0", corresponds to the number of attached T-junctions.

For each block deletion, we keep a record of block name, CB orientation, and the sequence of T_i. At the end of the deletion iterations, we can obtain three lists: block name list $\{B_n,B_{n-1},...B_1\}$, orientation list$\{L_n,L_{n-1},...L_2\}$, T-junction list$\{T_n,T_{n-1},...T_2\}$. We reverse the data of these three items respectively. Thus, we have a sequence *S* of



(a) the vertical ***CB*** (b) the horizontal ***CB***

Figure 2. The orientation of corner block(CB)

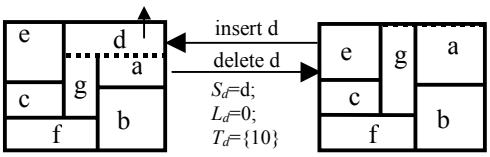
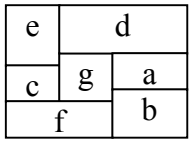


Figure 3. Corner block *d* is deleted/inserted

block names, a list *L* of orientations, and a list of $\{T_2,T_3,...T_n\}$ which is combined into a binary sequence *T*. The three element triple (*S*,*L*,*T*) is a corner block list. The insertion process of corner block based on given (*S*,*L*,*T*) can construct the corresponding placement. Fig.4 is a non-slicing placement and its corresponding CBL.



Corner Block List:
S=(fcegbad)
L=(001100)
T=(0 0 10 10 0 10)

Figure 4. A non-slicing placement and its CBL list

3. The Compact algorithm for CBL

Corner Block List can represent mosaic structure, which includes *n* rooms with one and only one block in each room, where *n* is the number of blocks. However, the area usage and interconnect may be affected by this stringent assumption. Given a CBL list, we expand the packing from the lower left to upper right by inserting the blocks in list *S* in turn. Every insertion of the block may generate some more dead space. In order to make things clear, we restrict the objective to be the total area of the placement. Our algorithm could be extended to minimize a different objective, such as the wire length cost.

3.1 The stair outline of the packing

Given a CBL list, the packing is expanded from the lower left to the upper right by inserting the blocks in list *S* in turn. The blocks can not be packed at the empty area at the right of or below the previous packed blocks. Thus we define the stair outline of the packing to propose the compaction process.

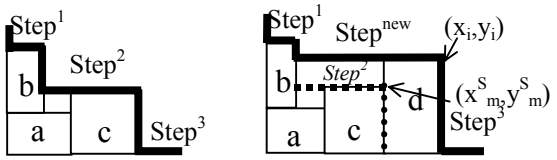
Definition 1: **Stair Outline**: Given a placement of a subset of blocks, the stair outlines enclose the subset of the blocks in the bottom left corner of the chip area. The stair outlines form a rectilinear curve that goes monotonically from up left toward down right.

The stair outline is the descensive stairs which are composed of several steps$\{Step^1, Step^2...Step^k\}$.

Property 1: Suppose that the position of the upper right corner for each $Step^i$ is (x^S_i, y^S_i). For two steps $Step^m$ (x^S_m, y^S_m) and $Step^n$(x^S_n, y^S_n), if $m>n$ then $x^S_m > x^S_n$ and $y^S_m < y^S_n$. The upper right corner of last step is (∞, 0).

The operations of stair outline can be operated as following(Fig.5):

Suppose that $\{Step^1, Step^2, .. Step^k\}$ is the stair outline before M_i is packed and the upper right corner of block M_i after packing is (x_i, y_i).



(a)The stair outline before block *d* is packed

(b)The stair outline after block *d* is packed.

Figure 5. The operation of stair outline

1. A new step $Step^{new}$ whose upper right corner is (x_i, y_i) is added into the list of stair outline;
2. For each step $Step^m$ (whose upper right corner is at the position of (x^S_m, y^S_m)), if $Step^m$ is lower and at the left of the new step ($x_i > x^S_m$ and $y_i > y^S_m$), the $Step^m$ should be replaced by the new step $Step^{new}$ in the list of stair outline.

Definition 2: **Corner Step** is the step whose upper right corner is the upper right corner of the corner block.

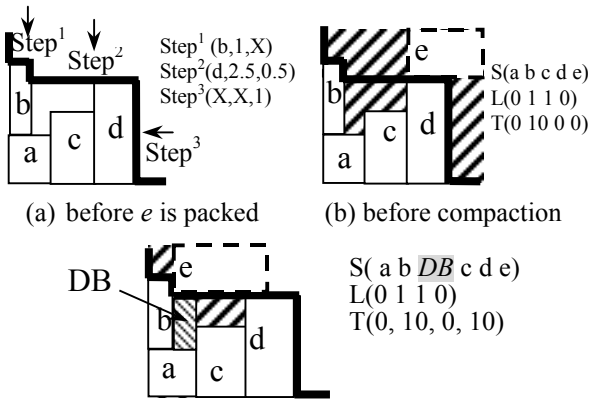
Lemma 1: Suppose that Corner step is the k^{th} step $Step^k$. The next block M_{next} is packed along the i^{th} step $Step^i$.

- if the next block M_{next} is horizontal, M_{next} should be located along the i^{th} step after the corner step ($i > k$);
- if the next block M_{next} is vertical, M_{next} should be located along the i^{th} step which is the corner step or the step before the corner step ($i \le k$).

As shown in Fig.6(a), before block *e* is packed, the stair outline is {$Step^1$, $Step^2$, $Step^3$}, the corner step is $Step^2$. The blocks along $Step^1$ and $Step^2$ should be vertical, while the blocks along $Step^3$ should be horizontal.

3.2 The compaction during the packing process

Since we do not want to loose the topological relations in the given CBL list, we should maintain the packing sequence in list *S* while doing the compaction. Thus the area below the stair outline will be settled no matter how the latter blocks will be packed. To compact the placement, every block should be packed at a left-bottom corner of the step. As in Fig. 6(b), according to the given CBL, block '*e*' is packed above block '*d*' along $Step^2$, the shadowed area means the dead space which can not be utilized by the latter blocks. No matter how the latter blocks will be placed according to the given CBL, it is inevitable that the shadowed space will become the dead space in the final packing. But if block '*e*' can be pushed to the left-bottom corner of $Step^2$, the shadowed area will decrease a lot to obtain a tight packing(Fig. 6(c)). Since the compaction may generate some empty rooms which do not contain any block in it, we modify the given CBL by adding some dummy blocks.



(a) before *e* is packed

(b) before compaction

(c) After compaction

Figure 6. The compaction method

Definition 3: **Dummy Block** is a false block with zero area, whose function is to generate an empty room with no block in it.

Since the dummy blocks are used to provide positions to be covered by other blocks and the topological relations between other blocks will not be affected by the inserted dummy blocks, dummy blocks would not necessarily appear in the CBL list. Therefore, when we count the T-junctions covered by blocks, we omit the T-junctions composed by dummy blocks.

Corresponding to the packing process of CB, each step of the stairs is defined by (B^S_i,W^S_i,H^S_i) where B^S_i is the block whose upper tight corner is the upper right corner of $Step^i$; W^S_i and H^S_i are the width and height of the step respectively indicating how many blocks are covered by the step vertically and horizontally. If the step is along the boundary, we use '**X**' to indicate no block available to be covered. If the step covers a block partially, we use '**0.5**' to represent it(Fig.6(a)). The triple of (B^S_i,W^S_i,H^S_i) can be easily obtained while packing.

Using the stair outline while packing, the compact process should follow lemma 2(Fig. 6(c)).

Lemma 2: Suppose that before M_i is packed, the stair outline is {$Step^1$, $Step^2$,…$Step^n$} in sequence and the corner step is the m^{th} step $Step^m$, M_i should be packed along the step $Step^k$ according to the given CBL and TN_i is the number of T-junctions covered by M_i. For block M_i, we use the position of its bottom left corner (x_i,y_i) to define its placement. To compact the block M_i to a feasible position, M_i should be moved to the left-bottom corner of $Step^k$. Thus $x_i = x^S_{k-1}$ and $y_i = y^S_k$. And the CBL list should be modified according to the following:

- if L_i=0, $TN_i = \sum_{i=k}^{m} \lfloor W_i^S \rfloor - 1$ and if W^S_k is not an integer, a dummy block(*DB*) should be added at the right of block B^s_{k-1} : the orientation of *DB* should be L_{DB}=1; the T-junctions covered by *DB* should be $TN_{DB} = \lceil H^S_k \rceil - 1$;
- if L_i=1, $TN_i = \sum_{i=m+1}^{k} \lfloor H_i^S \rfloor - 1$ and if H^S_k is not an integer, a dummy block(*DB*) should be added above block B^s_k: the orientation of *DB* should be L_{DB}=0; the T-junctions covered by *DB* should be $TN_{DB} = \lceil W^S_k \rceil - 1$;

($\lfloor A \rfloor$ is the largest integer which is not greater than A. $\lceil A \rceil$ is the smallest integer which is not smaller than A.)

In Fig.6(c), we can see the block '*e*' is packed at the lower left corner of $Step^2$ (*d*, 2.5, 0.5) , corner step is $Step^2$ and TN_e should be modified according to Lemma 2. Since block '*e*' is vertically inserted, B^S_{k-1} is block '*b*' and $Step^2$ covers a block partially, a dummy block should be added after block '*b*':

L_{DB}=1;

$TN_{DB} = \lceil H^S_k \rceil$ -1=0;

$TN_e = \sum_{i=k}^{m} \lfloor W_i^S \rfloor$-1 = $\sum_{i=2}^{2} \lfloor W_i^S \rfloor$ -1 =1,

Lemma 3: The total number of the dummy blocks is no more than *n-3*, where *n* is the number of the blocks.

Proof: Since the first three blocks need no dummy blocks inserted in them and for the other blocks, there should be no more than one dummy block inserted for each block, the total number of the inserted dummy block should be no more than *n-3*.

Based on *Lemma 1* and *Lemma 2*, we devise the compact algorithm as following:

Algorithm *Compact packing*
Begin
Initialize the stair outline and the packing;
For *i from 2 to n*
M_i should be placed along the k th step $Step^k$ in the stair outline:
Compact M_i according to ***Lemma 2****;*
Reconstruct the stair outline.
EndFor
End

The compaction process modifies the given CBL into a better solution within its neighbor solutions space. Since our algorithm gives the corresponding CBL to the final packing after the compaction process, the topological relations are contained in the modified CBL. Therefore, our algorithm can be extended to handle topological constraint problem.

4. Compact packing with rectilinear blocks

Based on the compact process, we extend the CBL representation to handle the packing problem with arbitrary shaped rectilinear blocks.

4.1 The partition of rectilinear blocks

Each rectilinear blocks is partitioned into a serial rectangular blocks by horizontal segments or vertical segments. The lower left sub block is defined as the master block and the other sub blocks are sequenced by a feasible packing sequence from lower left to upper right(Fig.7). The packing sequence of the sub blocks is not unique and the relative positions for each sub block are recorded by the difference between the coordinates of the lower left corner of a sub block and its previous sub block.

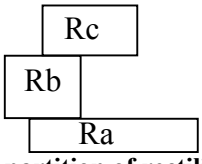


Figure 7. The partition of rectilinear blocks: the sequence of sub blocks : (Ra,Rb,Rc)

4.2 packing with no overlapping

Since the sequence of the sub blocks is the sequence of packing, once the master block is packed, the packing positions for other sub blocks can be settled iteratively by the configuration of the stair outline and the relative positions between sub blocks.

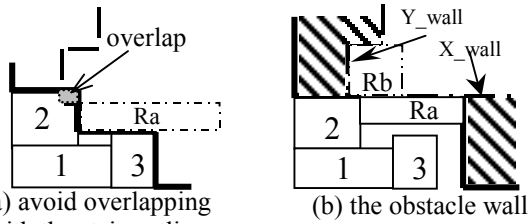
Lemma 4: Suppose that before master block is packed, the stair outline is {$Step^1$, $Step^2$,…$Step^m$} in sequence. Then the master block is packed and the feasible packing positions for other sub blocks are settled. If there is no sub block whose lower left is at (x_k,y_k) that {$x_k < x^S_i$ and $y_k < y^S_i$ | (x^S_i,y^S_i) is the coordinates for $Step^i$, $i \le m$}, the sub blocks will not overlap with the stair outline.

To amend the overlapping, the master block should be moved horizontally by max({$x^S_i - x_k$| $x_k < x^S_i$, x^S_i is the x coordinate of $Step^i$ }) or vertically by max({$y^S_i - y_k$| $y_k < y^S_i$, y^S_i is the y coordinate of $Step^i$ }). As shown in Fig.8(a), if sub block *Ra* was settled in the lower left corner of the step, it will generate overlapping between the steps and the other sub blocks. The block *Ra* should be moved upwards or to the right to avoid the overlapping.

To avoid the overlapping with the blocks packed after the master block, we set the obstacle walls to protect the feasible packing area of the next sub blocks.

Definition 4: **Obstacle wall**: the obstacle wall is the virtual line which the steps of the stair outline should not exceed unless the next sub block is packed.

When a sub block B^1 is packed in its feasible position, the next sub block B^2 should be at the relative positions. Correspondingly, there are two obstacle walls: one is vertical(*Y_wall*), and the other is horizontal(*X_wall*) which are the left outline and bottom outline of the rest unpacked sub blocks. Then the blocks between B^1 and B^2 should not exceed these two obstacle walls. Once a block B^m or a set of rectilinear sub blocks exceeds the walls, we change the position of B^2 in CBL list to the position before the block B^m or the set of the sub blocks and pack the block B^2 at the desired position. The corresponding CBL representation can be obtained by the configuration of the stair outline. To pack the rectilinear block *R_block*, we devise the procedure which can be called during the



(a) avoid overlapping with the stair outline (b) the obstacle wall

Figure 8. The packing of sub blocks

packing process.

Procedure *packing_ rectilinear:*
Input: the stair outline before rectilinear block
R_block is packed;
the number of the sub block is sub_n;
Begin
Pack the master block following lemma 4;
For *i* from 2 to *sub_n*:
Construct the obstacle walls;
While not (the obstacle walls are exceeded or the next block is a sub block of *R_block*):
Packing the blocks according to given CBL:
If (the next block is a sub block for the other rectilinear block *R_block2* and *R_block2* will not overlap with *R_block*)
Merge *R_block2* into *R_block* with their relative positions considered and change the number of *sub_n*;
EndIf
EndWhile;
Pack the i^{th} sub block to its feasible position and modify the CBL according Lemma 2;
EndFor;
End.

5. Experimental Results

We have implemented the placement algorithm in C programming language, and all experiments are performed on a SUN sparc20 workstation. Some MCNC benchmarks are used in the experiments. Our experiments do not include soft blocks. Using area as the objective function, our algorithm can get better results in shorter time compared with ECBL and CBL. The most particular character is that the results of our algorithm is very stable even the annealing schedule is devised to limit the running time in a very small range. From the results, we can see for the data ami33, the area usage can reach 90.0% in 2.22 seconds and reach 94.9% in 8.74 seconds. And for data ami49, the area usage can reach 90.508% in 9.86 seconds and reach 93.6% in 20.15 seconds. It is unreachable for the other algorithms to obtain the fast convergence to an optimal solution as our algorithm can do. Fig.9 is a result packing of ami49. The area usage reaches 96.7%; the running time is only 101.9 seconds. The comparison between CBL, $ECBL_2$(CBL is extended by adding 2**n* empty blocks) and our algorithm with the relationship between the solution quality and running time of ami33 is shown in Fig.11. Compared with CBL and ECBL, our algorithm needs shorter running time to get comparable good results. Table.1 is the block placement results and Table 2 is the results compared with other published algorithms[3,4,5,7,8,17].

To test the packing algorithm with the rectilinear blocks, we expand several blocks to rectilinear shaped block. Fig.10 is the packing results of modified ami49 and the area usage are above 90% and the running time is less than 200 seconds. We can see from the results of the experiments that both concave and convex blocks can be handled and the performance is quite good

6. Conclusion

In this paper, we have proposed a compact algorithm for placement based on CBL. By introducing the dummy blocks in CBL, our algorithm can represent the placement including empty rooms, which CBL cannot represent. And the dummy block is employed intellective only when the solutions can be improved by adding the dummy blocks. Our algorithm can compact the placement during the packing process based on the outline shape of the packing and the CBL list can be changed accordingly. Therefore, our algorithm has the same complexity as the original CBL algorithm. Since our algorithm can obtain the fast convergence to an optimal solution, our algorithm maintains the high quality of the original algorithm at a fraction of the CPU time. Based on the configuration of the stair outline, we can handle the problem with rectilinear blocks, including both convex and concave rectilinear blocks. For each CBL, there is a modified CBL corresponding to a feasible packing with rectilinear blocks; and the transformation from a modified CBL to its placement takes only linear time. It is not necessary to do any operations to restore the shapes of the rectilinear blocks since their shapes are maintained while packing.

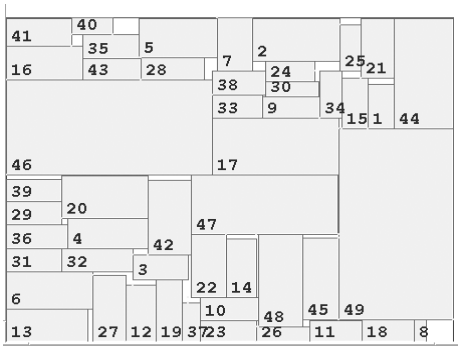


Figure 9. A packing result of ami49. The area is 36.64 mm^2 and the area usage is 96.7%; the running time is only 101.9 seconds

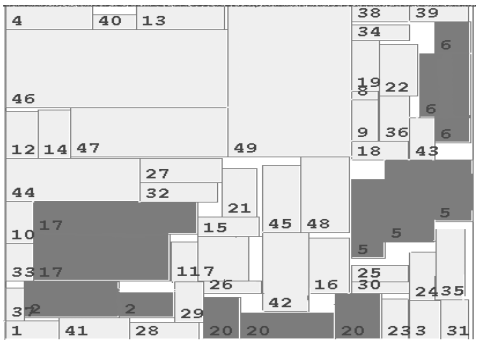


Figure 10. The result packing of ami49 with 7 rectilinear blocks: the area usage is 91.4% and the running time is 128 seconds.

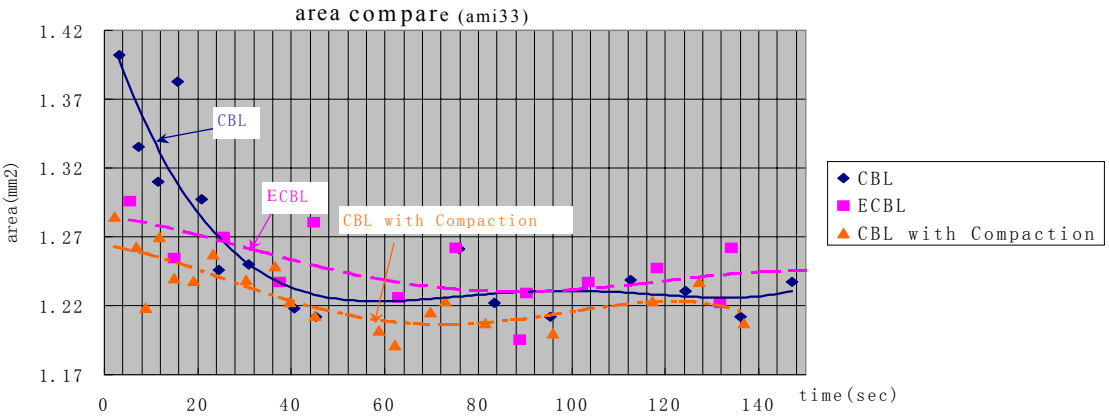


Figure 11. Comparison with the solution quality and run-time

The experimental results prove the effectiveness of our approach. Since our algorithm gives the corresponding CBL to the final packing, different developments of our algorithm can be done to answer the problem arisen from the requirement of wires, design rules and so on.

Reference

[1] Hiroshi Murata, Kunihiro Fujiyoshi, S.Nakatake and Y.Kajitani, "VLSI Block Placement Based on Rectangle-Packing by the Sequence Pair" in *IEEE Trans. on CAD,vol.15, NO. 15,* pp 1518-1524,1996.

[2] S.Nakatake, H.Murata, K.Fujiyoshi and Y.Kajitani, "Block Placement on BSG-structure and IC layout application" in *Proc. of International Conference on Computer Aided Design*, pp 484-490,1996.

[3] P.N.Guo,and C,K,Cheng, "An O-tree representation of non-slicing floorplan and its applications", in *ACM/IEEE Design Automation Conference*,1999.

[4] Yun-Chih Chang, Yao-Wen Chang, Guang-*Mi*ng Wu, and Shu-Wei Wu."B*-Trees: A New Representation for Non-Slicing Floorplans" in *ACM/IEEE DAC*,pp.458-464, 2000.

[5] Hong Xianlong, Huang Gang et al. "Corner Block List: An Effective and Efficient Topological Representation of Non-slicing Floorplan" *ICCAD*'2000.

[6] D.F.Wong, C.L.Liu, "A new algorithm for floorplan design", in *Proc. of 23rd ACM/IEEE DAC*, pp.101-107, 1986.

[7] Jin Xu, Pei-Ning Guo, Chung-Kuan Cheng, "Cluster Refinement for Block Placement", in *DAC*'97.

[8] Shuo Zhou, Sheqin Dong, Xianlong Hong, Yici Cai,Chung-Kuan Cheng, Jun Gu "ECBL: An Extended Corner Block List with O(*n*) complexity and solution space including optimum placement", *ISPD*' 2001

[9] Yuchun Ma, Sheqin Dong, Xianlong Hong, Yici Cai,Chung-Kuan Cheng, Jun Gu " VLSI Floorplanning with Boundary Constraints Based on Corner Block List" *ASPDAC*'2001.

[10] Yuchun Ma, Xianlong Hong, Sheqin Dong, Yici Cai, Chung-Kuan Cheng, Jun Gu "Floorplanning with Abutment Constraints and L-shaped/T-shaped Blocks Based on Corner Block List", *DAC* 2001,pp.770-776.

[11] Yingxin Pang, Chung-Kuan Cheng, Koen Lampaert, Weize Xie" Rectilinear Block Packing Using O-tree Representation", *ISPD* 2001

[12] M.Kang, W.W.M.Dai," General Floorplanning with L-shaped, T-shaped and Soft Block Based on Bounded Slicing Grid Structure", ASPDAC'1997, pp. 265-270,1997

[13] Kunihiro Fujiyoshi, Hiroshi Murata, "Arbitrary Convex and Concave Rectilinear Block Packing using Sequence-pair", *ISPD*'1999

[14] M.Kang, W.W.M.Dai, "Arbitrary Rectilinear Block Packing Based on Sequence Pair" *ICCAD*'1998

[15] J.Xu, P.N.Guo, C.K.Cheng: "Rectilinear Block Placement Using Sequence-Pair" *ISPD*'1998

[16] F.Y.Young, Hannah H.Yang, D.F. Wong: "On extending slicing floorplans to handle L/T-shaped blocks and abutment constraints", *WCC*'2000

[17] Jai-Ming Lin and Yao-Wen Chang: "TCG: A Transitive Closure Graph-Based Representation for Non-slcing Floorplans", in *DAC'2001*,pp.764-769.

Table 1. The results of the placement

circuits	CBL with Compaction	
	Average area(mm^2)/time(*Sec*)	Minimum area(mm^2)/time(*Sec*)
apte	47.16/3.2	46.85/5.2
xerox	20.45/3.52	19.95/5.1
hp	9.240/6.76	9.12/12.3
Ami33	1.21/35.43	1.191/62.7
Ami49	37.29/50.46	36.64/101.9

Table 2. The Area(mm^2) / Time(*Sec*) comparison among CBL with compaction(on Sparc 20), ECBL(on Sparc 20), CBL(on Sparc 20),O-tree(on Ultra60),B*-tree(on Ultra-I),TCG(on Ultra 60),Cluster refinement(on Sparc 20)

circuits	CBL with compaction	ECBL	CBL	O-tree	B*-tree	TCG	Cluster (size = 4 blocks)
Ami33	1.192/62	1.192/73	1.201/36	1.242/119	1.27/3417	1.20/306	1.207/603.4
Ami49	36.64/101	36.70/117	38.58/65	37.73/526	36.8/4752	36.77/434	37.69/1861.7