Floorplanning with Abutment Constraints and L-shaped/T-shaped Blocks based on Corner Block List

Yuchun Ma, Xianlong Hong, Sheqin Dong, Yici Cai
Department of Computer Science and Technology, Tsinghua University, Beijing, China
{hxl-dcs}{dongsq}@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 HongKong

ABSTRACT

The abutment constraint problem is one of the common constraints in practice to favor the transmission of data between blocks. Based on Corner Block List(CBL), a new algorithm to deal with abutment constraints is developed in this paper. We can obtain the abutment information by scanning the intermediate solutions represented by CBL in linear time during the simulated annealing process and fix the CBL in case the constraints are violated. Based on this algorithm, a new method to deal with L-shaped/T-shaped blocks is proposed. The shape flexibility of the soft blocks and the rotation and reflection of L-shaped/T-shaped blocks are exploited to obtain a tight packing. The experiment results are demonstrated by some benchmark data and the performance shows effectiveness of the proposed method.

1. INTRODUCTION

A good floorplanner must not only provide a good rectangle packing functionality but also the flexibility to handle a large variety of specific constraints For general floorplan including both slicing and non-slicing, several encoding schemes were recently proposed, namely, Sequence-Pair(SP)[1], Bound-Sliceline-Grid(BSG)[2], O-tree[3], B*-tree[4] and Corner Block List(CBL)[5]. 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. Different from other topological representations, CBL needs a smaller amount of encoding storage and linear time computation effort to generate each placement configuration. These advantages are good for handling placement constraints in general.

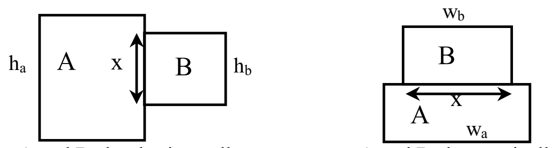
The abutment constraint problem is one of the common constraints in practice for the designer may want to have the logic blocks in a pipeline of a circuit to abut one after another to favor the transmission of data between them. But in most stochastic floorplanning algorithms, the relative position between two blocks is not known until the exact dimensions of blocks are taken into account. Recently, F.Y.Young [7] has proposed the algorithm to handle abutment constraints based on slicing structure. We propose a new abutment constraint algorithm based on CBL which can handle the abutment constraints for both slicing and non-slicing. In our algorithm, we check the abutment constraints by scanning the intermediate solutions represented by CBL in linear time during the simulated annealing process and fix the corner block list by heuristic method in case the constraints are violated.

With the recent advent of deep submicron technology and new packing schemes such as multichip modules, the L/T-shaped blocks have been specifiable and /or permissible for it is motivated by enforced colocation of a data path block with related control, or the shape of particular types of data path blocks. To handle L/T-shaped blocks, we propose a new method based on abutment constraint algorithm. An L/T-shaped block can be partitioned into a few of rectangular blocks called sub-blocks which have natural abutment relations with each other. We transform the L/T-shaped block problem into abutment constraint problem. The rotation and reflection of soft blocks and L/T-shaped blocks can be implemented and non-overlapped packing is guaranteed. Our algorithm has been implemented in C language and the experiment results are promising.

The rest of the paper is composed as follows: A formal definition of abutment constraint and the problem of L/T-shaped block are described in Section 2. Section 3 is a brief review of the CBL model. The new algorithm is presented in Section 4 and Section 5. The experimental results are shown in Section 6. Finally, conclusion is given.

2. PROBLEM DEFINITION

Each rectangular block M_i is defined by a tuple(h_i,w_i), where h_i and w_i are the height and the width of the block M_i, respectively. The aspect ratio of M_i is defined as h_i/w_i. There are two kinds of rectangular blocks: soft blocks and hard blocks. The soft blocks have fixed area with variable aspect ratio within a given range. The hard blocks have fixed area and aspect ratio.



A and B abut horizontally iff $x>=\min(h_a,h_b)$

A and B abut vertically iff $x>=\min(w_a,w_b)$

Figure.1 The definition of abutment

Definition 1: If a vertical boundary h_a of block A and a vertical boundary h_b of block B superpose each other and the shorter one of h_a and h_b is covered completely by the longer one, block A and block B are said to be abutted with each other horizontally. The block abutment in the vertical direction is defined similarly.

Definition 2: An L-shaped block is the rectilinear block which can be partitioned into two abutted rectangular sub-blocks L1 and L2, where one boundary of L1 and one boundary of L2 are aligned horizontally(vertically) (Fig.2.). A T-shaped block is the rectilinear block which can be partitioned into three abutted rectangular sub-blocks T1, T2 and T3, where three boundaries of T1, T2 and T3 respectively are aligned horizontally(vertically) and the height(width) of the middle sub-block is the greatest among the three blocks. (Fig.3).

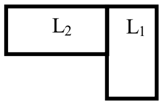


Figure.2. L-shaped block

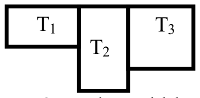


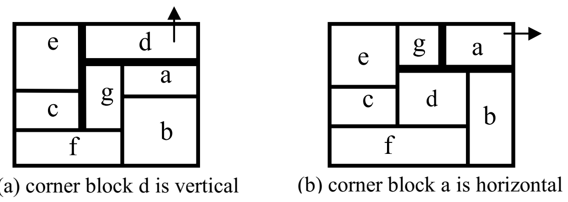
Figure3. T-shaped block

Definition 3: A feasible packing is a non-overlapping placement of both rectangular blocks and L /T-shaped blocks such that all the abutment constraints are satisfied and all L/T-shaped blocks are in their original shapes.

3. CORNER BLOCK LIST

A floorplan divides the chip into rectangular rooms with horizontal and vertical segments. Each room is assigned to no more than one block. Each pair of intersected segments forms a T-junction. 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. CBL is derived from a simplified version of general floorplan called mosaic floorplan. A floorplan belongs to the class of mosaic floorplan if and only if it observes the following three properties.

1. Floorplan of No Empty Space;
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;
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.


e
d
a
c
g
b
f
(a) corner block d is vertical
e
g
a
c
d
b
f
(b) corner block a is horizontal
Figure 4. The orientation of the corner block.

The Corner Block is the block packed in the upper right corner room of the floorplan. In the floorplan, the joint of the left and bottom segments of the corner block is contained in a T-junction named corner T-junction and the corner block's orientation is defined by the orientation of the corner T-junction(Fig.4). The T-junction has only two kinds of orientations: T rotated by 90 degrees (Fig 4(a)) and by 180 degrees(Fig 4(b)) counterclockwise respectively. If T is rotated by 90 degrees counterclockwise, we define the corner block to be vertical oriented, and denote it by a "0". Otherwise, the corner block is horizontal oriented, and denote it by a "1". The corner block list is constructed from the record of a recursive corner block deletion. In fig.5, the corner block 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 corner block 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 corner block M_i. The number of successive 1s, which is ended by a '0', corresponds to the number of attached T-junctions.

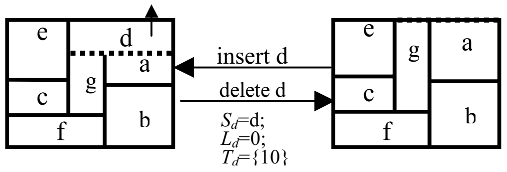
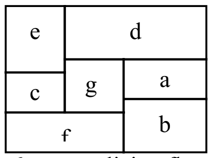


Figure 5 Corner block d is deleted/inserted

For each block deletion, we keep a record of block name, corner block orientation, and the sequence of T_i. At the end of deletion iterations, we can obtain three list: block name list $\{B_n,B_{n-1},...B_1\}$, orientation list$\{L_n,L_{n-1},...L_2\}$, T-junction list$\{T_n,T_{n-1}, \ldots,T_2\}$(n is the number of the blocks). We reverse the data of these three items respectively. Thus, we have a sequence S of block names, a list L of orientations, and a list of $\{T_2,T_3, \ldots,T_n\}$ which is combined into a binary sequence T. The three element triple (S, L, T) is called a corner block list. The insertion process of corner block based on given (S,L,T) can construct the corresponding floorplan. Fig. 6 is an example of a non-slicing floorplan and its corresponding CBL.

In some cases, the number of successive 1s in list T_i is greater than the number of available T-junctions when M_i is inserted, it will be an infeasible CBL. However, since the blocks along the left and bottom boundaries cover all the T-junctions available, the ending '0' in T_i list is not necessary. When corner block M_i along the left or bottom boundary is deleted, we omit the ending '0' in T_i to construct T'. Thus during the construction process, when the number of successive 1s in list T' is greater than or equal to the number of the available T-junctions, we just place the corner block M_i along the boundary covering all the available T-junctions and count '1's for M_i as many as the number of available T-junctions. Therefore, an arbitrary CBL is feasible which corresponds to a mosaic floorplan.



Corner Block List:
S=(fcegbad)
L=(001100)
T=(0 0 10 10 0 10)
T'=(10 1 0 10)

Figure 6. a non-slicing floorplan and its corresponding CBL list

The transformation from corner block list to floorplan can be achieved by scanning the CBL in linear time of $O(n)$. The number of combinations of CBL is $O(n!2^{3n-3}/n^{1.5})$ and CBL takes only $n(3+[lg\ n])$bit to describe.

4. ABUTMENT CONSTRAINTS OVER CBL

Abutment information is embodied in the blocks beside the segments, which divides the chip into rectangular rooms: the blocks beside a horizontal segment abut vertically; the blocks beside a vertical segment abut horizontally. Each block has four boundaries to abut with other blocks.

Definition 4: *HSEG* is one horizontal segment and thus T^h_{HSEG} and B^h_{HSEG} denote the sets of blocks lying above and below segment *HSEG* respectively; *VSEG* is one vertical segment and thus L^v_{VSEG} and R^v_{VSEG} denote the sets of blocks lying at the left and right of segment *VSEG* respectively.

Definition 5: L_abut[M_i], R_abut[M_i], T_abut[M_i], B_abut[M_i] denote the set of blocks' rooms lying along the left side, right side, top side and bottom side of the room of block *Mi*. And block M_i is called *master block* for all of its abutted blocks have relative abutment relation to M_i.

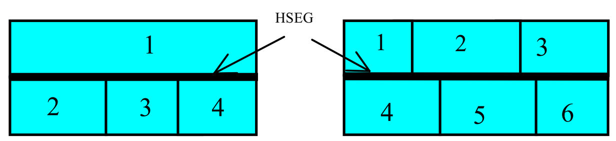
4.1 Finding the Abutment Information in CBL

In the mosaic floorplan, we can obtain all the affirmatory abutment information following lemma 1.

Lemma 1: *HSEG* is one horizontal segment. $B^h_{HSEG}=\{B^b_1,B^b_2,\ldots B^b_m\}$ denotes that *m* blocks below *HSEG* are arranged from left to right and $T^h_{HSEG}=\{B^a_1,B^a_2,\ldots B^a_n\}$ denotes that *n* blocks above *HSEG* are arranged from left to right. The abutment information is as following:

- If n=1, then the corresponding rooms of $\{B^b_1,B^b_2,\ldots B^b_m\}$ are lying immediately below the room of B^a_1; If m=1, then the corresponding room of B^b_1 is lying immediately below the rooms of $\{B^a_1,B^a_2,\ldots B^a_n\}$
- If $m \geqslant 2$ and $n \geqslant 2$, then the corresponding rooms of B^b_1 lies immediately below the room of B^a_1; the room of B^b_m lies immediately below the room of B^a_n (Fig.7b).

It is similar if L is vertical.



(a) Room 1 is the only block lying above HSEG
T_abut[2]=T_abut[3]=T_abut[4]={1}
B_abut[1]={2,3,4},

(b) B_abut[1]={4}, B_abut[3]={6}.
T_abut[4]={1}, T_abut[6]={3}

Figure.7 The abutment information embodied in segments.

In the floorplan, each segment is the line whose ends are on the boundaries of chip or on its intersection perpendicular. Suppose that all the horizontal segments are directed from left to right and all the vertical segments are directed from bottom to top.

Definition 6: *SEG* is one segment in floorplan. If the starting point of *SEG* is on SEG_s, SEG_s is called the starting segment of *SEG*. If the ending point of *SEG* is on SEG_e, SEG_e is called the ending segment of *SEG*(Fig.8).

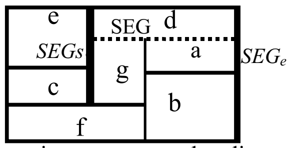


Figure 8. The starting segment and ending segment

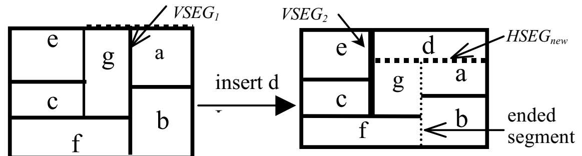
The blocks beside a segment can be determined by the blocks between the starting segment and the ending segment. If the segment is ended, the blocks beside the segment are fixed and the abutment information can be obtained according to Lemma 1.

M_i is the i_{th} blocks in block list *S*. L_i is the orientation of M_i and TN_i is the number of T-junctions to be covered by block M_i recorded in list T. Suppose that the *p* blocks along the top boundary of the floorplan $\{B^t_1,B^t_2,\ldots B^t_p\}$ are arranged from right to left, which are separated by the *p-1* vertical segments $\{VSEG_1,VSEG_2,\ldots VSEG_{p-1}\}$, and $p \geqslant TN_i+1$(which is guaranteed by the validity of CBL). The *q* blocks along the right boundary of the floorplan $\{B^r_1,B^r_2,\ldots B^r_q\}$ are arranged from top to bottom, which are separated by the *q-1* horizontal segments $\{HSEG_1,HSEG_2,\ldots HSEG_{q-1}\}$ and $q \geqslant TN_i+1$.

Lemma 2: when M_i is inserted as the corner block:

- If $Li=0$, then the first TN_i vertical segments $\{VSEG_1, VSEG_2,\ldots VSEG_{TNi}\}$ are ended by M_i's bottom boundary which is a new horizontal segment $HSEG_{new}$.
  $B^h_{HSEGnew}=\{B^t_{TNi+1}\ldots B^t_2,B^t_1,\}$, $T^h_{HSEGnew}=\{M_i\}$
  $R^v_{VSEG-TNi+1}=R^v_{VSEG-TNi+1}\cup\{M_i\}$;
- If Li=1, then the first TN_i horizontal segments $\{HSEG_1,HSEG_2,\ldots HSEG_{TNi}\}$are ended by *Mi*'s left boundary which is a new vertical segment $VSEG_{new}$.
  $L^v_{VSEGnew}=\{B^r_{TNi+1}\ldots B^r_2,B^r_1,\}$, $R^v_{VSEGnew}=\{M_i\}$
  $T^h_{HSEG-TNi+1}=T^h_{HSEG-TNi+1}\cup\{M_i\}$;

The function of ***Abutment*** is devised to find out the abutment relationships by scanning the given CBL from left to right. We use two stacks of H-segment and V-segment to record the unended segment during the scanning. Each element in H-segment contains two sets T^h_{HSEG} and B^h_{HSEG} denoting the sets of blocks lying above and below segment *HSEG* respectively. Each element in V-segment contains two sets L^v_{VSEG} and R^v_{VSEG} denoting the sets of blocks lying at the left and right of segment *VSEG* respectively.



(a) L^v[VSEG$_1$]=$\{f,g\}$,
R^v[VSEG$_1$]=$\{b,a\}$,
R^v[VSEG$_2$]={g}
$\{B^t_1,B^t_2\ B^t_2\}$={a,g,e},

(b)B^h[HSEG$_{new}$]=$\{g,a\}$,T^h[HSEG$_{new}$]={d}
R^v[VSEG$_2$]={g} ∪ {d},
R_abut[f]={b}, L_abut[b]={f},
R_abut[g]={a}, L_abut[a]={g}

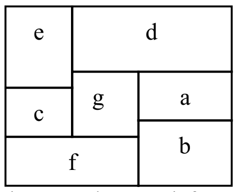
Figure.9. The confirmation of Lemma 2

Algorithm Abutment()

```
    block_covered: the set of the blocks covered by new corner block;
    V-segment_top: the top element in stack V-segment;
    H-segment_top: the top element in stack H-segment;
for each block $M_i$ do
{block_covered=null;
If L=0:
    TN=the number of the T-junctions covered;
    for k=1 to TN:
    {
        $L^v_{VSEG}$=V-segment_top.$L^v_{VSEG}$
        $R^v_{VSEG}$=V-segment_top. $R^v_{VSEG}$;
        Pop V-segment_top;
        Find out all the horizontal abutment relations in $L^v_{VSEG}$ and $R^v_{VSEG}$ according to lemma1;
        block_covered=block_covered ∪ the last block in $L^v_{VSEG}$ ∪ the last block in $R^v_{VSEG}$;
    }
    Reverse the blocks sequence in block_covered;
  Push a new element ($B^h_{HSEGnew}$=block_covered, $T^h_{HSEGnew}=\{M_i\}$) into stack H-segment according to lemma 2;
  If the buffer of V-segment is not null:
    $R^v_{V\text{-}segment\_top}=R^v_{V\text{-}segment\_top}\cup\{M_i\}$;
If L=1:
    TN=the number of the T-junctions covered;
    for k=1 to TN:
    {
        $T^h_{HSEG}$=H-segment_top.$T^h_{HSEG}$
        $B^h_{HSEG}$=H-segment_top. $B^h_{HSEG}$;
        Pop H-segment_top;
        Find out all the horizontal abutment relations in $T^h_{HSEG}$ and $B^h_{HSEG}$ according to lemma1;
        block_covered=block_covered ∪ the last block in $T^h_{HSEG}$ ∪ the last block in $B^h_{HSEG}$;
    }
```

```
        Reverse the blocks sequence in block_covered;
    Push a new element ($L^v_{VSEGnew}$=block_covered, $R^v_{VSEGnew}$={$M_i$}) into stack V-segment according to lemma 2;
    If the buffer of H-segment is not null:
        $T^h_{H\text{-}segment\_top} = T^h_{H\text{-}segment\_top} \cup \{M_i\}$;
}
Pop all the elements left in stacks and find out all the abutment relations.
```

Fig 10 is an example of the result of ***Abutment***. We can use this information to check and fix the abutment constraints.



	L_abut	R_abut	T_abut	B_abut
a	g	x	d	b
b	f	x	a	x
c	x	g	e	f
d	e	x	x	g,a
e	x	d	x	c
f	x	b	c,g	x
g	c	a	d	f

Figure.10.abutment information obtained by Abutment (X means null)

4.2. Heuristic Method of Fixing the Abutment Constraint

We combine some heuristic method into the process of annealing process. Since the abutment information can be obtained before packing, we can fix the CBL to satisfy the constraints as much as possible. One effective operation is swapping the blocks in list *S*. An example is shown in Fig. 11. The abutment constraint is violated in Fig. 11(a) for block *'e'* and block *'f'* do not abut with each other horizontally, as required. By the procedure of ***Abutment,*** the abutment information can be obtained as listed in fig 10. We can find that T_abut[f]={c,g} and B_abut[f]=null. We exchange block *'e'* with block *'c'* to fix the violated constraint. The complexity of this procedure is O(u) where u is the number of the violated constraints of the floorplan. However, the number of blocks which violate their constraints decreases rapidly during the annealing process. Therefore, the time taken in this process is actually very little in practice.

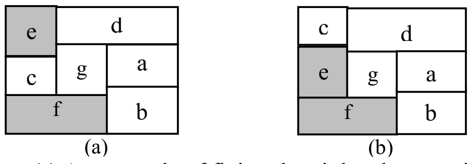


Figure 11.An example of fixing the violated constraints

4.3 Penalty Term

Some constraints may be still violated after all the possible shuffles. In some cases, one block B^c_a in the constrained pair(B^c_a, B^c_b) is fixed because of the other constraints and there are no available neighbors of B^c_a to be exchanged with B^c_b, the constraint will remain violated after the process of fixing the constraint. Therefore, we include a term in the cost function to penalize the remaining violated constraints.

Suppose that block A(ha,wa) and block B(hb,wb),where (ha,wa) and (hb,wb) are the height and width of block A and B respectively, are constrained to abut horizontally and the top-right positions of block A and block B are (xa,ya) and (xb,yb) respectively(Fig.12). The penalty term is

$$P=||\,xa\text{-}wa/2\text{-}(xb\text{-}wb/2)|\text{-}wa/2\text{-}wb/2|+ \max(0,(ya\text{-}yb)*(ya\text{-}ha\text{-}(yb\text{-}hb)))$$

Lemma 3: If penalty term *P=0,* the constraints are satisfied.

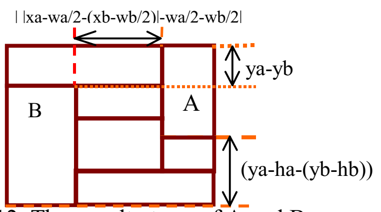


Figure 12. The penalty term of A and B

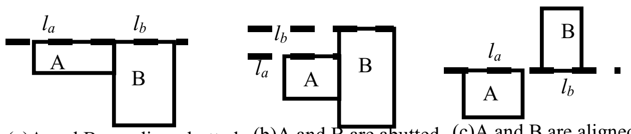
Proof: If | *xb-wa/2-(xb-wb/2)*|*-wa/2-wb/2* =0, the distance in horizontal orientation between block A and block B is 0. The difference between the top boundaries of block A and block B is *ya-yb*. And the difference between the bottom boundaries of block A and block B is *ya-ha-(yb-hb)*. If block A and block B superpose and the shorter one in vertical orientation is covered by the longer one, *(ya-yb)*(ya-ha-(yb-hb))* should be negative. Thus *max(0,(ya-yb)*(ya-ha-(yb-hb)))=0.* ¶

5. L /T-SHAPED BLOCK FLOORPLAN

An L-shaped/T-shaped block is partitioned into abutted sub-blocks with edges aligned. Each sub-block is handled as an individual block. After partitioning, we have transformed the L-shaped/T-shaped block problem into abutment constraint problem.

5.1 Align-abutment

Definition 7: If block A and block B abut with each other and one boundary l_a of A and one boundary l_b of B are aligned in the same segment, block A and block B are said to have align-abutment relation with each other (Fig.13(a)).



(a)A and B are align-abutted (b)A and B are abutted, but not aligned (c)A and B are aligned, but not abutted

Figure13. The definition of align-abutment: A and B are align-abutted in (a),but not in (b) or (c)

Each block has eight positions for align-abutment (Fig 14). Note that V1,V2,V3 and V4 are the positions for the blocks abut upon block M_i vertically; H1,H2, H3 and H4 are the positions for the blocks abut upon M_i horizontally.

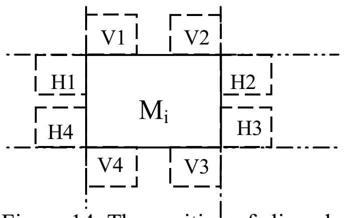


Figure 14. The position of align abutment

Lemma 4:

1. The first one in the L_abut[M_i] is at position H1; the last one in the L_abut[M_i] is at position H4;
2. The first one in the R_abut[M_i] is at position H2; the last one in the R_abut[M_i] is at position H3;
3. The first one in the T_abut[M_i] is at position V1; the last one in the T_abut[M_i] is at position V2;
4. The first one in the B_abut[*Mi*] is at position V4; the last one in the B_abut[M_i] is at position V3;

The problem of align-abutment constraint is a subset of the problem of abutment constraint and it can be handled similarly to the problem of abutment constraint while lemma 4 can be used to limit the constraints.

5.2 Partition of L-shaped/T-shaped Block

A L-shaped/T-shaped block can be partitioned into a set of sub-blocks with horizontal or vertical lines and the partitioned sub-blocks have align-abutment relations defined in *Definition 7*. There are eight ways in which the two sub-blocks can form an L-shaped block. And each pair of them has its specific align-abutment relation(Fig.15). It is similar to partition the T-shaped block. In fig.16, we take the middle one of sub-blocks as the master block. In fig 15 and fig 16, M is the master block and the positions of align-abutment relation are shown in the figures.

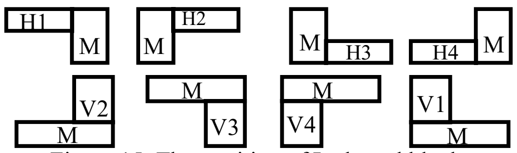


Figure.15. The partition of L-shaped blocks

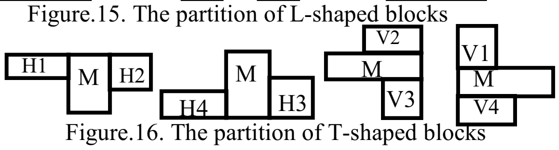


Figure.16. The partition of T-shaped blocks

5.3 The Alignment of L/T-shaped Block

The sub-blocks can be aligned to form the original shape of L-shaped/T-shaped blocks if their rooms have similar align-abutment relations. All the alignment operations are within the rooms of the sub-blocks and the positions of the other blocks are not affected. The aligned boundaries of the rooms simplify the alignment of the sub-blocks.

Lemma 5: Suppose that sub-blocks L1 and L2 have been packed with the align-abutment relation that L2 abut with L1 at the position H1 as required. The top-right positions of block L1 and block L2 are (x1,y1) and (x2,y2) respectively. And (h1,w1) and (h2,w2) are the height and width of L1 and L2 respectively. The coordinate alignments should be done as following and non-overlapping is guaranteed:

if h1>h2:

y1=max(y1,y2); y2=max(y1,y2); x2=x1-w1;

If h1<h2:

y1=max(y1,y2); y2=max(y1,y2); x1=x2+w2;

It is similar to L-shaped blocks of other directions.

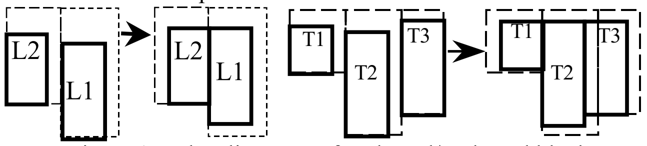


Figure.17. The alignment of L-shaped/T-shaped block

Lemma 6: Suppose that sub-blocks T1,T2 and T3 have been packed with the align-abutment relation that T1 and T3 abut with T2 at the positions H1 and H2 respectively. The top-right positions of block T1, block T2 and block T3 are (x1,y1), (x2,y2) and (x3,y3) respectively. And the(h1,w1), (h2,w2) and (h3,w3) are the height and width of T1, T2 and T3 respectively. The coordinate alignments should be done as following and non-overlapping is guaranteed:

y1=max(y1,y2,y3);y2=max(y1,y2,y3); y3=max(y1,y2,y3)

x1=x2-w2; x2=x2; x3=x2+w3;

It is similar to T-shaped blocks of other directions.

5.4 Rotation and Reflection of L/T-shaped Block

Rotation and reflection of L-shaped/T-shaped block need take the shape information into consideration. Since our algorithm is based on topology of the packing, rotation and reflection of L-shaped/T-shaped block can be solved by changing the corresponding constraints generated by partitioning. We take T-shaped blocks as example to explain the operation of L-shaped/T-shaped blocks(fig.18). The corresponding changes of align-abutted position are listed in table 1.

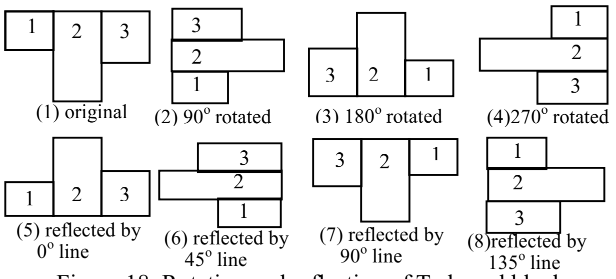


Figure 18. Rotation and reflection of T-shaped blocks

Table 1: the corresponding positions after rotation and reflection

	Rotation			Reflection			
	90^0	180^0	270^0	0^0	45^0	90^0	135^0
H1	V4	H3	V2	H4	V3	H2	V1
H2	V1	H4	V3	H3	V2	H1	V4
H3	V2	H1	V4	H2	V1	H4	V3
H4	V3	H2	V1	H1	V4	H3	V2
V1	H4	V3	H2	V4	H3	V2	H1
V2	H1	V4	H3	V3	H2	V1	H4
V3	H2	V1	H4	V2	H1	V4	H3
V4	H3	V2	H1	V1	H4	V3	H2

6. EXPERIMENTAL RESULTS

The floorplan algorithm with abutment constraints and L/T-shaped blocks has been implemented in the C programming language, and all experiments are performed on a SUN spark20 workstation. Some MCNC benchmarks are used for the examples. All the blocks in the experiments have shape flexibility that their aspect ratios distribute between 0.2 and 5 discretely. To test the abutment constraint algorithm, several blocks are required to abut horizontally or vertically. The average results for benchmark with different constraints are listed in Table 2. We can see from the result that our method can handle the abutment constraints efficiently. Compared with the abutment algorithm based on slicing floorplans[7] with the same experiment instances and the similar constraints, the results of run times and the deadspaces in our algorithm are as good as or even better than the results in [7], while our algorithm has a great diversity of floorplan structure including both non-slicing and slicing. Since the penalty terms devised in this paper evaluate the violations of the constraints accurately, we can ensure all the abutment constraints are satisfied in our experiment results. But the results in [7] shows that the constraints can not be satisfied even after a long-time annealing process. Especially with the large-scaled instances, our algorithm is much faster and more effective than the algorithm in [7]. The algorithm in [7] handles the packing of 62 modules in 453 seconds with the deadspace of 2.93% and two out of twelve abutment constraints are violated. While in our algorithm, we pack 99 modules in 199 seconds with the deadspace of 4.07% and all the twelve abutment constraints are satisfied.

To test the L-shaped/T-shaped block algorithm, we expand several blocks to L-shaped/T-shaped block. The results are list in Table 3 and Fig.21 show the results of some randomly generated examples. We can see from the results of the experiments that the performance is quite good.

7. SUMMARY AND CONCLUSIONS

This paper proposes a new algorithm to handle the abutment constraints not only with slicing structure, but also with non-slicing structure. Based on the abutment constraint algorithm, we design a new method to handle L-shaped/T-shaped block. The L-shaped/T-shaped block is partitioned into a collection of rectangle blocks with additional abutment constraints. The alignment is taken to maintain the original shapes of L-shaped/T-shaped blocks and non-overlapped packing is guaranteed. The rotation and reflection of L-shaped/T-shaped block are also fulfilled in our algorithm. The penalty term help to ensure all the constraints are satisfied by the end of the annealing process The experiment results demonstrate that our algorithm is quite promising.

8. ACKNOWLEDGEMENTS

This work was supported by China National Nature Science Funds under the grants 60076016, China 973 National Fundamental Funds under the grants No.G1998030403 and NSFC International Cooperative Major Program.

9. REFERENCES

[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,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 in press

[6] M.Kang and W.W.M.Dai, "General Floorplanning with L-shaped, T- shaped and Soft Blocks Based on Bounded Slicing Grid Structure", IEEE Asia and South Pacific Design Automation Conference, pp. 265-270,1997.

[7] F.Y.Young,Hannah H.Yang, D.F.Wong "On extending slicing floorplans to handle L/T-shaped blocks and abutment constraints" WCC'2000, pp.269-276,2000.

[8] Kunihiro Fujiyoshi, Hiroshi Murata: " Arbitrary Convex and Concave Rectilinear Block Packing using Sequence-pair", in International Symposium of Physical Design, pp.103-110,1999

[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, pp 509-514.2001

Table 2. Results of testing Abutment algorithm

Data	#	Resulting	Dead space	Run time
Ami33	8	1.185	2.58	92.3
Ami49	8	36.29	2.44	168.5
P65	11	2.39	4.01	161.81
P99	12	3.61	4.07	199.14
P147	15	113.7	6.46	370.4

Table 3. Results of testing L-shaped/T-shaped blocks

examples	#|M|	#|T |	#|L |	Net area	Area	Dead space	time
apte_lt	9	2	7	59.59	60.91	2.1	38.5
xerox_lt	10	4	6	26.07	27.15	3.6	43.85
hp_lt	11	3	6	11.5	12.04	4.6	47.7
Ami33_lt	33	4	8	1.295	1.368	5.4	169
Ami49_lt	49	6	11	37.91	40.04	5.4	253
P65_lt	65	5	11	2.433	2.528	3.7	306

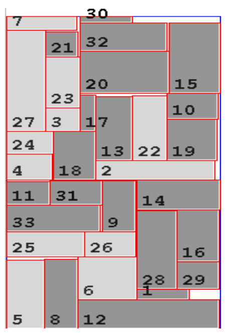


Figure 19. A result packing of ami33 where (2,22); (3,23); (4,24); (5,25); (6,26); (7,27) are constrained to abut vertically respectively.

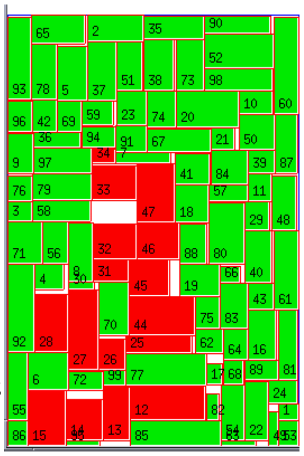


Figure 20. A result packing of P-99:blocks 12,13,14,15 and 25,26,27,28 are required to compose two horizontal chains and blocks 31,32; 33,34 and 44,45,46,47 are required to compose vertical chains.

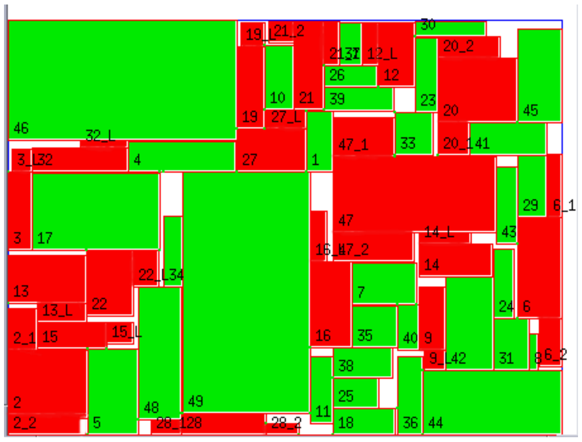


Figure 21. A result of ami49_lt with 11 L-shaped blocks and 6 T-shaped blocks