Simultaneous Handling of Symmetry, Common Centroid, and General Placement Constraints

Qiang Ma, Linfu Xiao, Yiu-Cheong Tam, and Evangeline F. Y. Young

***Abstract*—In today's system-on-chip designs, both digital and analog parts of a circuit will be implemented on the same chip. Parasitic mismatch induced by layout will affect circuit performance significantly for analog designs. Consideration of symmetry and common centroid constraints during placement can help to reduce these errors. Besides these two specific types of placement constraints, other constraints, such as alignment, abutment, preplace, and maximum separation, are also essential in circuit placement. In this paper, we will present a placement methodology that can handle all these constraints at the same time. To the best of our knowledge, this is the first piece of work that can handle symmetry constraint, common centroid constraint, and other general placement constraints, simultaneously. Experimental results do confirm the effectiveness and scalability of our approach in solving this mixed constraint-driven placement problem.**

***Index Terms*—Analog placement, common centroid constraints, constraint graph, corner block list, sequence pair (SP), symmetry constraints.**

I. Introduction

In TODAY'S system-on-chip designs, both digital and analog parts of a circuit will be implemented on the same chip. Placement of the analog parts is an error-prone and time-consuming process. In analog placement, parasitic mismatch induced by the layout will affect the circuit performance significantly. Consideration of symmetry and common centroid constraints during placement can help to reduce these errors. For symmetry constraint, pairs of cells are required to be placed symmetrically with respect to a horizontal or vertical axis. For common centroid constraint, devices will be split into a number of smaller sub-devices and placed in rotational symmetry about a common center point. An example is shown in Fig. 1. In Fig. 1, there is a common centroid group with two devices. Each device is divided into smaller sub-devices and layout in an interleaving manner to satisfy the common centroid requirement. Besides common centroid and symmetry, other general constraints, such as alignment, abutment, preplace, and maximum separation, are also essential in circuit placement. In this paper, we will present a methodology that can handle all these constraints, simultaneously. To the best of our knowledge, this is the first piece of work that can handle symmetry constraint, common centroid constraint, and other general placement constraints at the same time.

The problem of placing devices with symmetry constraint has been extensively studied [1], [2], [4], [9], [11]–[13], [15]–[17], [19], [20]. Most previous studies used simulated annealing as an optimization engine based on a packing representation. We can classify those representations into two categories: 1) absolute representation, and 2) topological representation. In an absolute representation [4], [9], [13], modules are represented by their absolute coordinates on the chip plane. However, the size of the solution space is huge in this case which will affect the solution quality given a limited amount of searching time. In a topological representation, the relative positions between the modules are encoded. The solution space is much smaller in comparison with that of an absolute representation, but complicated computations are needed for checking symmetry feasibility and adjusting the module positions to satisfy the constraints. Topological representations such as sequence-pairs [14], O-tree [6], B*-trees [3], and TCG-S [10] have been applied to handle symmetry constraints [1], [2], [11], [12], [15]–[17], [19], [20]. Most of these previous works handle only symmetry constraints while [12] handles common centroid constraint and [19] and [20] handle both. Strasser *et al.* [19] presented an algorithm called *Plantage* that makes use of a hierarchically bounded enumeration of basic building blocks and the B*-tree representation to perform analog placement taking into account symmetry, common centroid, proximity, and minimum distance constraints. The last two constraints are about the maximum and minimum distances between two modules. *Plantage*, however, cannot handle other placement constraints in general. The work in [20] handles both symmetry and common centroid constraints efficiently based on the symmetric feasible sequence pair (SP) representation.

As mentioned above, component mismatch has significant adverse effects on many analog circuits. This kind of mismatch can be effectively suppressed by a common centroid layout, which refers to a layout style in which a set of devices have a common center point. Devices can be split into a number of smaller ones and placed with the same center point. The

Manuscript received December 8, 2009; revised March 19, 2010 and June 3, 2010; accepted July 6, 2010. Date of current version December 17, 2010. The preliminary versions of this paper appeared in the Proceedings of ICCAD'06 [16] and ICCAD'07 [12]. This work was supported in part by the Research Grants Council of the Hong Kong Special Administrative Region, China, under Project 418908. This paper was recommended by Associate Editor Y.-W. Chang.

Q. Ma is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61821 USA (e-mail: qiangma1@illinois.edu).

L. Xiao, Y.-C. Tam, and E. F. Y. Young are with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong (e-mail: lfxiao@cse.cuhk.edu.hk; yctam@cse.cuhk.edu.hk; fyyoung@cse.cuhk.edu.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2010.2064490

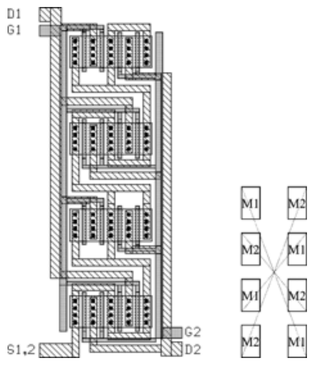


Fig. 1. Layout of a differential pair (M1 and M2). Both M1 and M2 are divided into four sub-devices.

devices can be arranged with a common centroid in one dimension as for a differential pair, or in two dimensions as for a capacitor or resistor array in data converters. 2-D arrangement is desired especially when the number of devices is large since close proximity is desirable for better electrical properties.

In placement of digital and analog circuits, there are also other placement constraints, such as maximum separation (an upper bound on the separation distance between pairs of critically matched devices), alignment, abutment, boundary, preplace, and range. There is no previous work on handling common centroid constraint, symmetry constraint, and other general placement constraints, simultaneously. In this paper, we try to address this mixed constraint-driven placement problem with all different kinds of constraints, simultaneously. A global SP with center-based corner block list (C-CBL) [12] is used as the representation in the simulated annealing engine. Instead of handling the constraints passively by having a penalty term in the cost function to penalize violations, we will try to satisfy all the constraints constructively during the placement realization step. Each symmetry or common centroid group will be packed as close to each other as possible. Experimental results do confirm the effectiveness and scalability of our approach in handling different types of constraints at the same time.[1]

II. Problem Definition and Preliminaries

We are given a set of n blocks of areas A_i and aspect ratio bounds $[l_i, u_i]$ where $i = 1 \ldots n$, together with a set of m nets $N_1, N_2 \ldots N_m$. We are also given the following:

1) a set of p common centroid groups $G_1, G_2 \ldots G_p$, where each group G_i contains s_i devices labeled by $\{g_{i1}, g_{i2} \ldots g_{is_i}\}$ and g_{ij} denotes the area of the jth device in group G_i;
2) a set of q symmetry groups $H_1, H_2 \ldots H_q$ where each symmetry group H_i contains $self(H_i)$ self-symmetry blocks and $pair(H_i)$ symmetry pairs;
3) a set of r placement constraints $C_1, C_2 \ldots, C_r$ where each placement constraint C_i denotes a constraint in placement between two arbitrary blocks.

[1]Executable and data sets can be downloaded from http://www.cse.cuhk.edu.hk/fyyoung/project/source/analog.html.

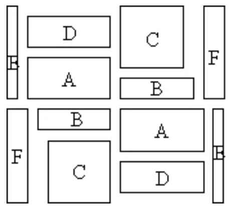


Fig. 2. Example of the common centroid constraint.

Our objective is to obtain a placement F of the circuit satisfying all the common centroid, symmetry, and general placement constraints, while minimizing a cost function $cost(F) = area(F) + \lambda \times wire(F)$, where $area(F)$ is the total area of F, $wire(F)$ is the total wire length of F measured by the half-perimeter estimation, and λ is a parameter specifying the relative importance between these two terms. General placement constraints include alignment, abutment, maximum separation, boundary, preplace, and range constraint.

A. Common Centroid Constraint

Each common centroid group G_i contains s_i devices with areas $\{g_{i1}, g_{i2} \ldots g_{is_i}\}$. We want to place these devices in such a way that they will all have the same center point. To achieve this purpose, each device will be split into a number of smaller devices of the same size and dimensions. An example is shown in Fig. 2. Users can specify the number of smaller devices split from each device and we will assume it to be two if no specifications are given. Users can also specify their requirements on the shapes of these sub-devices, if there is any, by providing the aspect ratio bounds. We will treat these smaller devices in pairs[2] and require them to have the same center point. In this way, all the original devices will also have the same center point. Throughout this paper, we have used n_i to denote the total number of smaller devices in G_i after the splitting process, and each original device g_{ij} is called a *subgroup* of G_i.

B. Symmetry Constraint

Each symmetry group H_i has $self(H_i)$ self-symmetry blocks and $pair(H_i)$ symmetry pairs. All of them are required to be placed symmetrically with respect to a horizontal or a vertical axis. An example is shown in Fig. 3. As for the common centroid constraint, users can specify their requirements on the shapes of these blocks by giving the aspect ratio bounds. They can also specify the direction of the symmetry axis, and both directions are feasible if no specifications are given.

III. Overview of Our Approach

Simulated annealing is employed as the basic searching engine in our approach using SP as the representation [14]. A global SP is used to represent the placement. In order to

[2]If the number of sub-devices is not even, we can further split one of them into two. In this way, we can still satisfy the common centroid constraint, but may not be able to handle some special cases, e.g., when there are three triplets, in a good way. In practice, we can consider identifying and handling these special cases separately.

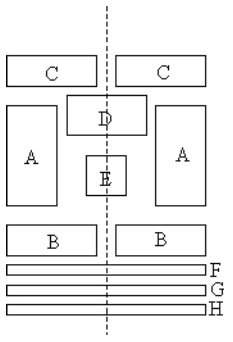


Fig. 3. Example of the symmetry constraint.

place all the devices of a common centroid group in close proximity without interleaving with other blocks not from the same group, each common centroid group will be treated as a super-block[3] in the SP. The internal structure of each common centroid group will be handled specifically in order to satisfy the common centroid constraint. During the annealing process, a set of random moves will be performed to perturb: 1) the global SP that contains the super-blocks representing the common centroid groups and the other blocks of the circuits, and 2) the internal structures of the common centroid groups. In each step of the annealing process, a global SP $x = (s_1, s_2)$ will be generated. We will first have an initial scan to check if x can be a feasible solution satisfying the constraints. After this initial scan, the common centroid groups will be packed. Then, a pair of constraint graphs (G_h, G_v) will be built according to the global SP to represent the relative positions among the modules and the superblocks. New nodes and constraint edges will be added into G_h and G_v to satisfy the symmetry constraint and the other general placement constraints.[4] Some of these newly inserted edges will have variable weights and we need to determine their weights to minimize the packing area and to get rid of positive cycles. Finally, if no positive cycles exist in the graphs, all constraints can be satisfied simultaneously and we will pack accordingly to obtain one feasible placement solution. Details on how we handle the general placement constraints, the symmetry constraint, and the common centroid constraint will be given in the following sections.

IV. Handling of General Placement Constraints

In our problem, we will handle the following general placement constraints.

1) *Alignment:* we use the notation $align(x, A, B)$ where $x \in \{l, r, t, b\}$ to denote that two blocks A and B are required to align vertically along the left $(x = l)$ or the right $(x = r)$ side, or to align horizontally along the top $(x = t)$ or the bottom $(x = b)$ side (Fig. 4).

2) *Abutment:* we use the notation $abut(x, A, B)$ where $x \in \{v, h\}$ to denote that two blocks A and B are required to abut horizontally $(x = h)$ with A on the left and B on the right, or to abut vertically $(x = v)$ with A at the bottom and B on top (Fig. 5). In our definition of abutment constraint, the shorter abuting side must abut completely with the longer abuting side.

3) *Maximum separation:* we use the notation $maxsep(x, A, B, y)$ where $x \in \{v, h\}$ and y is a positive real number to denote that two blocks A and B can at most be separated by a distance y horizontally $(x = h)$ or vertically $(x = v)$.

4) *Boundary:* we use the notation $boundary(x, A)$ where $x \in \{l, r, t, b\}$ to denote that block A is required to abut with the left $(x = l)$, right $(x = r)$, top $(x = t)$, or bottom $(x = b)$ boundary of the whole chip.

5) *Preplace:* we use the notation $preplace(x, y, A)$ where x, y are real numbers to denote that block A is required to be placed with its lower left (LL) corner at the coordinates (x, y).

6) *Range:* we use the notation $range(x, y, x_1, y_1, A)$ where x, x_1, y, y_1 are real numbers and $x_1 \geq x$ and $y_1 \geq y$ to denote that block A is required to be placed with its LL corner lying in the range from (x, y) to (x_1, y_1).

[3]The super-blocks are rectangular in shape and some area might thus be wasted. However, in order to reduce undesirable parasitic effects, devices in the same common centroid group are better well separated from other blocks, e.g., guard rings are sometimes used to separate the groups. It is thus reasonable to treat each common centroid group as a super-block.

[4]These placement constraints can also be applied on those super-blocks representing common centroid groups, e.g., we can have a shielding surrounding a common centroid group by creating four dummy modules and can impose an appropriate set of placement constraints between those dummy modules and the common centroid group.

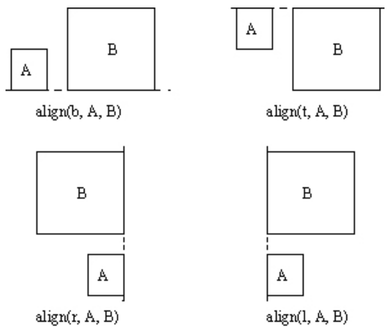


Fig. 4. Alignment constraint.

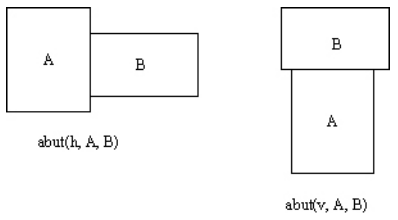


Fig. 5. Abutment constraint.

We will make use of the approach in [18] of adding pairs of edges into the constraint graphs to handle these general placement constraints. More details of the methodology can be found in [18]. However, we will first perform an initial scan on a candidate global SP to identify some of those infeasible solutions. These checkings are only used to screen out some infeasible solutions, but those remaining may still be infeasible, and we cannot identify them until the constraint graphs are built. The conditions being checked are as follows.

Alignment condition Q1: If block A is required to align with block B horizontally (vertically), the order of A and B in s_1 and s_2 must be the same (reversed).

Abutment condition Q2: If block A is required to abut with block B horizontally with A on the left (right), s_1 and s_2 must be of the form $s_1 = \ldots A \ldots B \ldots$ $(\ldots B \ldots A \ldots)$ and $s_2 = \ldots A \ldots B \ldots$ $(\ldots B \ldots A \ldots)$, respectively. Similarly, we can derive the condition for vertical abutment.

Boundary condition Q3: If block A is required to abut with the left (right) boundary of the chip, there should not be any block B such that B is before (after) A in both s_1 and s_2. Similarly, if block A is required to abut with the bottom (top) boundary, there should not be any block B such that B is before (after) A in s_1 and after (before) A in s_2.

V. Handling of Symmetry Constraint

In order to handle symmetry constraint and other general placement constraints simultaneously in a unified framework, we will also augment the constraint graphs to enforce symmetry constraint. For each symmetry group H_i containing $r_i = self(H_i)$ self-symmetry blocks $Z_1, Z_2 \ldots Z_{r_i}$ and $s_i = pair(H_i)$ symmetry pairs $(X_1, Y_1), (X_2, Y_2) \ldots (X_{s_i}, Y_{s_i})$, we will first check if H_i should be symmetric horizontally or vertically given a candidate global SP in an initial scan as described below.[5] W.l.o.g., we assume that the symmetry axis of H_i is vertical in the following discussion. First of all, we need to add constraint edges to the vertical constraint graph to align the symmetry pairs in H_i horizontally. A dummy node d_i will then be added to the horizontal constraint graph to represent the symmetry axis of H_i. In order to ensure equidistant between the symmetry pairs with respect to the axis, four edges, $e(X_j, d_i)$, $e(d_i, X_j)$, $e(d_i, Y_j)$, and $e(Y_j, d_i)$, will be added with weights x_{ij}, $-x_{ij}$, $x_{ij} - w(Y_j)$, and $-(x_{ij} - w(Y_j))$, respectively, for each $j = 1 \ldots s_i$ where $w(Y_j)$ is the width of block Y_j [notice that $w(X_j) = w(Y_j)$] and $x_{ij} \geq w(Y_j)$ is a positive real number. For each self symmetry block Z_j where $j = 1 \ldots r_i$, a pair of edges, $e(Z_j, d_i)$ and $e(d_i, Z_j)$ of weights $w(Z_j)/2$ and $-w(Z_j)/2$ will be added to ensure that Z_j will be lying symmetrically on the axis. After adding these dummy nodes and new constraint edges, we will determine the value of x_{ij} for $j = 1 \ldots s_i$ such that no positive cycles exist in the graph and $\max_{1 \leq j \leq s_i} x_{ij}$ is minimized. We will discuss these steps in detail in the following sections.

A. Initial Scan

It has been shown in [1] that a sufficient symmetry feasible condition [8] in a SP (s_1, s_2) is the following.

Symmetry condition Q4:

$$s_1^{-1}(A) < s_1^{-1}(B) \Leftrightarrow s_2^{-1}(sym(B)) < s_2^{-1}(sym(A))$$

for horizontal symmetric groups where A and B are any two distinct blocks in the group, $s_1^{-1}(X)$ $[s_2^{-1}(X)]$ denotes the position of block X in s_1 (s_2), and $sym(X)$ denotes the symmetry counterpart of X [$sym(X)$ of a self-symmetry block X is X itself]. Notice that the above condition holds for any two blocks in the group, e.g., we can put B as $sym(A)$ and the condition requires that A is on the left of $sym(A)$. According to this symmetry condition, the blocks of a horizontal symmetric group will appear in a *mirror* form in a SP, e.g., $s_1 = \ldots A_1 \ldots A_2 \ldots$ and $s_2 = \ldots sym(A_2) \ldots sym(A_1) \ldots$. For example, for a symmetry group with two symmetry pairs, (a_1, b_1) and (a_2, b_2), and one self-symmetry block, c, a feasible SP allowing this group to be arranged symmetrically about a vertical axis will be $s_1 = \ldots a_1 \ldots a_2 \ldots c \ldots$ and $s_2 = \ldots c \ldots b_2 \ldots b_1 \ldots$. Similarly, a sufficient symmetry feasible condition for vertical symmetric groups is as follows:

$$s_1^{-1}(A) < s_1^{-1}(B) \Leftrightarrow s_2^{-1}(sym(A)) < s_2^{-1}(sym(B))$$

where A and B are any two distinct blocks in the group.

[5]If the direction of symmetry is specified by the user, this initial scan will be used to check whether the candidate global SP is a feasible one.

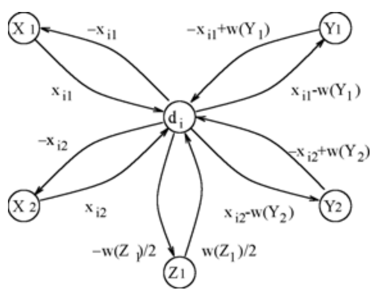


Fig. 6. Dummy nodes and additional constraint edges for a symmetry group.

In our implementation, we will only generate those global SP satisfying the symmetry condition $Q4$ because this condition is complicated while checking for violation of the other three conditions ($Q1$–$Q3$) can be done very effectively. This is done by initializing a SP solution that trivially satisfies the above condition for every symmetry group at the beginning of the annealing process, and then by maintaining these relationships throughout all the subsequent moving steps.

B. Bounds on the Variable Edge Weights

Fig. 6 shows the scenario of a simple symmetry group H_i with only two symmetry pairs (X_1, Y_1) and (X_2, Y_2), and one self-symmetry block Z_1. Now, we want to determine the values of x_{i1} and x_{i2} such that no positive cycles will be created and the value $\max\{x_{i1}, x_{i2}\}$ is minimized in order to obtain a more compacted solution. Consider any positive cycle *possibly forming*, the cycle must contain the dummy node d_i. In the following, we will enumerate all these potential positive cycles involving only one symmetry group and possibly other modules that are not in any other symmetry group. The variable $dist(A, B)$ denotes the longest path length (can be negative) from node A to node B in the original constraint graph before adding those dummy nodes and additional constraint edges for symmetry groups and is equal to $-\infty$ if there are no such paths. There are in total four types of positive cycles possibly forming enumerated as follows.

1) A cycle $X_j \rightarrow Y_j \rightarrow d_i \rightarrow X_j$ $(Y_j \rightarrow X_j \rightarrow d_i \rightarrow Y_j)$ for some $j = 1 \ldots s_i$ of total weight $dist(X_j, Y_j) - (x_{ij} -$

$w(Y_j)) - x_{ij}$ $(dist(Y_j, X_j) + x_{ij} + (x_{ij} - w(Y_j)))$ may be formed. To avoid positive cycles, it is required to have $2x_{ij} \geq dist(X_j, Y_j) + w(Y_j)$ $(2x_{ij} \leq w(Y_j) - dist(Y_j, X_j))$.

2) A cycle $X_j \to Y_k \to d_i \to X_j$ $(Y_k \to X_j \to d_i \to Y_k)$ for some $j, k = 1 \ldots s_i$ and $j \neq k$ of total weight $dist(X_j, Y_k) - (x_{ik} - w(Y_k)) - x_{ij}$ $(dist(Y_k, X_j) + x_{ij} + (x_{ik} - w(Y_k)))$ may be formed. To avoid positive cycles, it is required to have $x_{ij} + x_{ik} \geq dist(X_j, Y_k) + w(Y_k)$ $(x_{ij} + x_{ik} \leq w(Y_k) - dist(Y_k, X_j))$.

3) A cycle $X_j \to Z_k \to d_i \to X_j$ $(Z_k \to X_j \to d_i \to Z_k)$ for some $j = 1 \ldots s_i$ and $k = 1 \ldots r_i$ of total weight $dist(X_j, Z_k) + w(Z_k)/2 - x_{ij}$ $(dist(Z_k, X_j) + x_{ij} - w(Z_k)/2)$ may be formed. To avoid positive cycles, it is required to have $x_{ij} \geq dist(X_j, Z_k) + w(Z_k)/2$ $(x_{ij} \leq w(Z_k)/2 - dist(Z_k, X_j))$. Similarly, there may be cycles between Y_j and Z_k resulting in the constraints $x_{ij} \geq dist(Z_k, Y_j) - w(Z_k)/2 + w(Y_j)$ and $x_{ij} \leq w(Y_j) - w(Z_k)/2 - dist(Y_j, Z_k)$.

4) A cycle $X_j \to d_i \to X_k \to X_j$ $(Y_j \to d_i \to Y_k \to Y_j)$ for some $j, k = 1 \ldots s_i$ and $j \neq k$ of total weight $x_{ij} - x_{ik} + dist(X_k, X_j)$ $(-x_{ij} + w(Y_j) + x_{ik} - w(Y_k) + dist(Y_k, Y_j))$ may be formed. To avoid positive cycles, it is required to have $x_{ik} - x_{ij} \geq dist(X_k, X_j)$ $(x_{ij} - x_{ik} \geq w(Y_j) - w(Y_k) + dist(Y_k, Y_j))$.

The total number of such inequalities will be $2 \times pair(H) + 4 \times pair(H)^2 + 4 \times pair(H) \times self(H)$ for a symmetry group H, where $pair(H)$ is the number of symmetry pairs in H and $self(H)$ is the number of self-symmetry blocks in H. The first term $2 \times pair(H)$ is due to those type 1 cycles, the second terms $4 \times pair(H)^2$ is due to type 2 and type 4 cycles, and the last term $4 \times pair(H) \times self(H)$ is due to those type 3 cycles.

C. Computations of the Variable Edge Weights

From the above analysis, we can obtain upper and lower bounds for a single variable x_{ij}, for the sum of two variables $x_{ij} + x_{ik}$ and for the difference of two variables $x_{ij} - x_{ik}$. The bounds involve some pair-wise longest paths in the original acyclic constraint graph and the widths of some blocks, which can be computed effectively. Our goal is to evaluate all x_{ij}'s satisfying these bounds and minimizing $\max_{1 \leq j \leq s_i} x_{ij}$. This can of course be solved optimally by a linear solver but it will be too expensive to invoke a solver in every iteration of the annealing process. Therefore, we will solve this system of linear equations directly. Our approach can obtain the optimal solution when there are only lower bound constraints, e.g., when there are no general placement constraints. When there are both upper and lower bound constraints, the solution obtained by our method may be suboptimal, but this occurs very rarely as verified by the experiments (8 out of 2626 trials).

When there are only lower bounds, we can basically increase the values of the variables until all the lower bounds are satisfied. However, we do also want to minimize the value $\max_{1 \leq j \leq s_i} x_{ij}$. This can be achieved by carefully accounting a *slack* for each variable (how much a variable can be increased without increasing the value of the objective function value). When there are both upper and lower bounds, we will keep account of a *largest possible slack* for each variable due to the upper bound constraints. For example, consider two

Pseudocode *SolveConstraints()*

```
// Given a set $U = U_1 + U_2$ of upper bound constraints involving
// a single variable ($U_1$) or the sum of two variables ($U_2$) and a set
// $L = L_1 + L_2 + L_3$ of lower bound constraints involving a single
// variable ($L_1$), the sum of two variables ($L_2$) or the difference of
// two variables ($L_3$), we want to assign a value to each variable
// so that all the constraints are satisfied and the objective function,
// the largest variable value, is the smallest.

1.  For each variable $x$, set it to its smallest possible value
    according to $L_1$.
2.  Construct a constraint graph $G$ with vertices representing
    variables according to the difference constraints in $L_3$.
3.  For each variable $x$, update its value and its $slack(x)$
    according to $G$.
4.  For each variable $x$, initialize its largest possible slack,
    $Lslack(x)$, as $\infty$.
5.  For each variable $x$, update its $Lslack(x)$ as $\min\{Lslack(x),$
    $\min_{(x \leq a) \in U_1}\{a - x\}, \min_{(x+y \leq b) \in U_2}\{b - x - y\}\}$
6.  While some constraints in $L_2$ are not satisfied yet
7.    For each constraint $C : (x + y \geq b) \in L_2$
8.      If $Lslack(w) < 0$ for some variable $w$, return "failure"
        and exit.
9.      If $C$ is not satisfied:
10.       $x = \min\{x + slack(x), x + Lslack(x), b - y\}$
11.       Update $Lslack(x)$ according to the formula in step 5.
12.       Update $Lslack(z)$ according to the formula in step 5
          for every variable $z$ forming a constraint with $x$ in $U_2$.
13.       Update the value of each variable $w$ and its $slack(w)$
          from $G$.
14.     If $C$ is still not satisfied:
15.       $y = \min\{y + slack(y), y + Lslack(y), b - x\}$
16.       Update $Lslack(y)$ according to the formula in step 5.
17.       Update $Lslack(z)$ according to the formula in step 5
          for every variable $z$ forming a constraint with $y$ in $U_2$.
18.       Update the value of each variable $w$ and its $slack(w)$
          from $G$.
19.     If $C$ is still not satisfied:
20.       Increase both $x$ and $y$ by $(b - x - y)/2$.
21.       Update $Lslack(x)$ and $Lslack(y)$ according to the
          formula in step 5.
22.       Update $Lslack(z)$ according to the formula in step 5 for
          every variable $z$ forming a constraint with $x$ or $y$ in $U_2$.
23.       Update the value of each variable $w$ and its $slack(w)$
          from $G$.
```

upper bound constraints $x_{ij} \leq a$ and $x_{ij} + x_{ik} \leq b$ (notice that a difference constraint can always be written as a lower bound constraint), the largest possible slack of x_{ij} will be $\min\{a - x'_{ij}, b - x'_{ij} - x'_{ik}\}$ where x'_{ij} and x'_{ik} are the current values of x_{ij} and x_{ik}, respectively. Then, by adjusting the values of the variables according to the slacks and the largest possible slacks (which are updated dynamically), we can obtain a solution for the system of linear equations efficiently. The pseudocode is shown above.

D. Summary

The pseudocode at the top of the next page shows a summary of the steps to handle symmetry constraint. Notice that step 9 above is needed since there may be cycles formed between different symmetry groups after inserting those dummy nodes and additional constraint edges. For efficiency purpose, we have chosen to determine the edge weights of each group separately and check for positive cycles once at the end. An

Pseudocode *Symmetry()*

```
// Given a pair of acyclic constraint graphs ($G'_h$, $G'_v$) which are
// already augmented with edges to handle other general place-
// ment constraints, this procedure either announces that the
// symmetry constraint cannot be satisfied simultaneously or
// further augments them to satisfy the symmetry constraint.

1. For each symmetry group $H_i$
2.     Determine the longest path between every pair of blocks
       in $H_i$ in the constraint graphs ($G'_h$, $G'_v$).
3. For each symmetry group $H_i$
4.     Insert a dummy node $d_i$ to $G'_h$ ($G'_v$).
       /* Assume that the symmetry axis of $H_i$ is vertical. */
5.     Insert additional constraint edges between $d_i$ and the blocks
       in $H_i$ according to Section V.
6.     Determine the weights of the new constraint edges.
7.     If no solutions is obtained, return(fail).
8. Check for positive cycles in ($G'_h$, $G'_v$).
9. If positive cycles found in ($G'_h$, $G'_v$), return(fail).
10. Return($G'_h$, $G'_v$).
```

alternative will be solving a system of linear equations involving *all* the variable edge weights. In that case, we must invoke a solver since the upper and lower bound constraints will be more general, e.g., involving many variables. Notice that if different symmetry groups do not interleave and no negatively weighted paths exist between different symmetry groups in (G'_h, G'_v), e.g., no other general placement constraints between symmetry groups, the two approaches are the same, i.e., the variable edge weights in different groups will not affect each other.

VI. Handling of Common Centroid Constraint

One straightforward way to handle common centroid constraint is by extending the approach in Section V on symmetry constraint. The extension can be done easily by adding dummy vertices and new edges with variable weights to both the vertical and horizontal constraint graphs to ensure equidistance between the block pairs with respect to the center point in both the x and y directions. However, this approach is time consuming and we will demonstrate this in Section VIII on experimental results. In the following, we will propose a more effective approach to handle the common centroid constraint, by simply representing each common centroid group with a data structure that *must* correspond to a common centroid way of packing. Similar to the symmetry constraint, we will split each device into two smaller ones of the same dimension, and require all these pairs from the same group to be placed with a common centroid.[6] This approach is good since the total number of devices after splitting will be as small as possible. Now, each common centroid group G_i contains s_i pairs of devices $(a_{i1}, b_{i1}), (a_{i2}, b_{i2}) \dots (a_{is_i}, b_{is_i})$ with sizes $\frac{g_{i1}}{2}, \frac{g_{i2}}{2} \dots \frac{g_{is_i}}{2}$. In order to represent a common centroid placement of all these s_i pairs of blocks, a new representation called C-CBL [12] is used. C-CBL is a natural extension of the corner block list (CBL) representation [7] to handle common centroid groups. In the following, we will first briefly review CBL and then define C-CBL.

[6]The users can specify the number of smaller devices split from each device, and we can then handle them similarly in pairs to satisfy the common centroid constraint. If the specified number is not even, we can just split one of them into two.

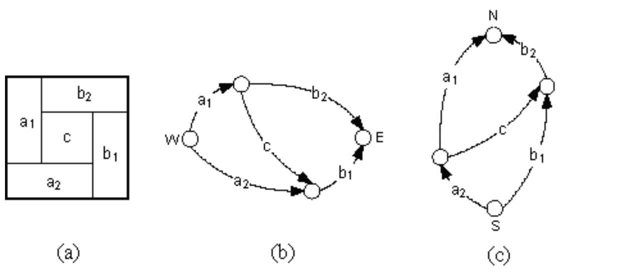


Fig. 7. (a) Mosaic packing. (b) Horizontal constraint graph. (c) Vertical constraint graph.

A. C-CBL

Fig. 7 shows a mosaic packing and its horizontal and vertical constraint graphs. In these constraint graphs, the nodes represent the vertical or horizontal segments in the packing while the edges represent the rooms for placing the modules. Additional nodes, labeled by W, E, S, and N, are inserted to represent the west, east, south, and north sides. A block is called a *corner block* if its corresponding edges in the two constraint graphs are pointing to the E and N nodes. The *orientation* of a corner block is defined by the orientation of the T-junction at its LL corner. If the T-junction is rotated by 90° counterclockwise, its orientation is vertical and is denoted by a bit zero; if the T-junction is rotated by 180° counterclockwise, it is horizontal and is denoted by a bit one.

The C-CBL [12] extends the basic CBL to represent the placement of a common centroid group. It is different from the original CBL that C-CBL works on *corner block pairs*. Given a mosaic packing, we can define the following.

Definition 1: UR corner block: A block is a upper right (UR) corner block if its corresponding edges in the horizontal and vertical constraint graphs are pointing to the E and N nodes, respectively. Its orientation is horizontal (vertical) if the T-junction at its LL corner is rotated by 180° (90°) counterclockwise. A UR corner block can be deleted from the packing by sliding the non-crossing segment of its LL T-junction to the right (top) if it is horizontal (vertical). The T-junction information is the number of T-junctions uncovered if the corner block is deleted from the packing. When there is only one pair of blocks in the packing, the UR corner block is horizontal (vertical) if its counterpart is on its left (below it).

Definition 2: LL corner block: A block is a LL corner block if its corresponding edges in the horizontal and vertical constraint graphs are pointing from the W and S nodes, respectively. Its orientation is horizontal (vertical) if the T-junction at its UR corner is rotated by 0° (270°) counterclockwise. An LL corner block can be deleted from the packing by sliding the non-crossing segment of its UR T-junction to the left (bottom) if it is horizontal (vertical). When there is only one pair of blocks in the packing, the LL corner block is horizontal (vertical) if its counterpart is on its right (above it).

Definition 3: Corner block pair: The UR corner block and the LL corner block of a packing are called a corner block

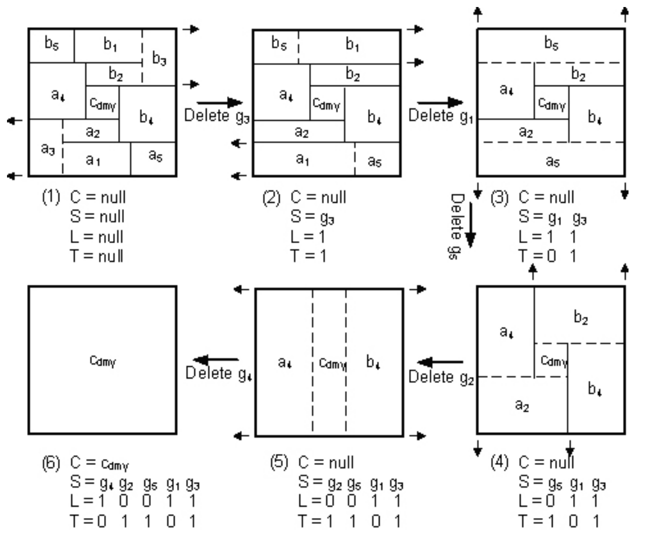

Delete g3
Delete g1
Delete g5
Delete g2
Delete g4
(1) C = null S = null L = null T = null
(2) C = null S = g3 L = 1 T = 1
(3) C = null S = g1 g3 L = 1 1 T = 0 1
(4) C = null S = g5 g1 g3 L = 0 1 1 T = 1 0 1
(5) C = null S = g2 g5 g1 g3 L = 0 0 1 1 T = 1 1 0 1
(6) C = cdmy S = g4 g2 g5 g1 g3 L = 1 0 0 1 1 T = 0 1 1 0 1

Fig. 8. Obtaining a C-CBL from a packing.

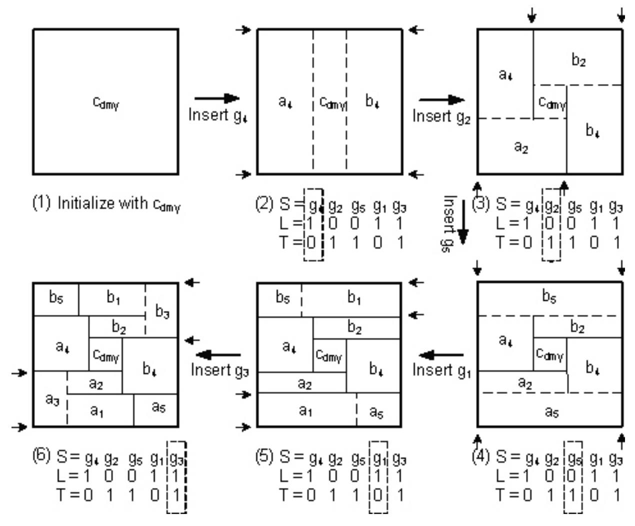

(1) Initialize with cdmy
Insert g4
Insert g2
Insert g5
Insert g1
Insert g3
(2) S = g4 g2 g5 g1 g3 L = 1 0 0 1 1 T = 0 1 1 0 1
(3) S = g4 g2 g5 g1 g3 L = 1 0 0 1 1 T = 0 1 1 0 1
(4) S = g4 g2 g5 g1 g3 L = 1 0 0 1 1 T = 0 1 1 0 1
(5) S = g4 g2 g5 g1 g3 L = 1 0 0 1 1 T = 0 1 1 0 1
(6) S = g4 g2 g5 g1 g3 L = 1 0 0 1 1 T = 0 1 1 0 1

Fig. 9. Obtaining a packing from a C-CBL.

pair if and only if they have the same orientation and have the same number of T-junctions uncovered after their deletions.

Given a mosaic packing of n pairs of blocks[7] satisfying the common centroid constraint, we can decompose the packing iteratively by removing its UR and LL corner blocks. Each removed pairs, representing a pair of devices a_{ij} and b_{ij}, will have the same orientation and T-junction information, i.e., they form a corner block pair. The C-CBL representation can thus be defined naturally according to this deletion process. In the following definition, we assume that there is always a single block located at the center position of the placement, for the sake of unified representation. This center block will be a dummy one with zero width and height when there are only n pairs of blocks.

Definition 4: C-CBL: A C-CBL is a four-tuple (C, S, L, T) where:

1) C is the name of the center block;
2) S is a sequence of x block names;
3) L is a list of x bits representing the orientations;
4) T is a list of x integers representing the T-junction information.

B. Obtaining C-CBL from a Packing

Given a mosaic packing satisfying the common centroid constraint, we can obtain the corresponding C-CBL by recursively deleting the corner block pairs. For example, suppose that we are now given a common centroid placement containing five pairs of blocks $\{g_1(a_1,\ b_1), g_2(a_2,\ b_2), g_3(a_3,\ b_3), g_4(a_4,\ b_4), g_5(a_5,\ b_5)\}$. Fig. 8 shows the process to obtain the corresponding C-CBL by iteratively performing corner block pair deletion. Note that the center block labeled as c_{dmy} denotes a dummy center. We can prove the following theorem constructively.

Theorem 1: A unique C-CBL can be obtained from a mosaic packing with n pairs of blocks (or n pairs plus one block) satisfying the common centroid constraint.

[7]The C-CBL can also handle n pairs of blocks plus one single block.

Proof: The proof can be done by induction on the number of blocks. The base case is the case when there is only one block C (this can be a dummy block with zero size). For this base case, the C-CBL is obviously $(C, \{\}, \{\}, \{\})$. Now, consider the inductive case that there are $k+1$ pairs plus one blocks in the mosaic packing P. Since P satisfies the common central constraint, the blocks occupying the rooms at the UR corner and the LL corner of P must belong to the same device Y. According to the symmetric property, these two blocks must have the same orientation and cover the same number of T-junctions. Consider deleting these two blocks from P by sliding them out of the packing either horizontally or vertically, depending on their orientations. Now, we are left with another mosaic packing with k pairs plus one blocks, which can be represented uniquely by a C-CBL (C, S, L, T) according to the inductive hypothesis. Let S' be obtained by appending Y to the end of S, L' be obtained by appending to L, a bit that represents the orientation of the two blocks, and T' be obtained by appending to T a positive integer that represents the number of T-junctions covered by the blocks. Then, the packing P can be represented by the C-CBL (C, S', L', T') uniquely.

C. Obtaining a Packing from a C-CBL

A mosaic packing satisfying the common centroid constraint can be constructed from a C-CBL(C, S, L, T) by initializing the packing with the center block C and then inserting the corner block pairs one by one according to the lists' information. Fig. 9 shows this packing procedure for the previous example, with the C-CBL given as $(C = c_{dmy}, S = (g_4, g_2, g_5, g_1, g_3), L = (1, 0, 0, 1, 1), T = (0, 1, 1, 0, 1))$. Similar to the CBL representation, an arbitrary four-tuple may not correspond to a feasible packing since the number of available T-junctions may be less than the required number of T-junctions to be covered (according to the T list) when we try to insert the UR and LL corner blocks. However, this condition can be checked and maintained easily in the implementation.

D. Realization of a Common Centroid Placement

To obtain a valid packing of a common centroid group G_i from a C-CBL, we need to adjust the coordinates of the blocks since they may not satisfy the common centroid constraint after the bottom-left-compacted packing process. We will recompute the coordinates of those UR corner blocks (assume that they are b_{ij}) as follows:

$$b_{ij}.x = w(G_i) - a_{ij}.x - width(b_{ij})$$
$$b_{ij}.y = h(G_i) - a_{ij}.y - height(b_{ij})$$

where $w(G)$ and $h(G_i)$ are the width and height of G_i, $(b_{i,j}.x, b_{i,j}.y)$ are the coordinates of block $b_{i,j}$, and $width(b_{i,j})$ and $height(b_{i,j})$ are the width and height of $b_{i,j}$. In addition, the coordinates of the center block b_c (if it is not a dummy) can be computed as $(w(G_i)/2 - w(b_c)/2, h(G_i)/2 - h(b_c)/2)$. Obviously, after this adjustment, the blocks in G_i will satisfy the common centroid constraints, while staying within the smallest possible region $w(G_i) \times h(G_i)$ without any overlapping. Similar to the floorplan realization step for the corner block list representation [7], this procedure will take $O(n)$ time where n is the number of blocks. This step will be performed whenever the C-CBL of any common centroid group G_i is perturbed, after which G_i will be regarded as a super-block with width $w(G_i)$ and height $h(G_i)$ in the global SP describing the whole circuit. We can thus prove the following theorem.

Theorem 2: The packing of a common centroid group G_i constructed from a C-CBL as described in Section VI-D satisfies the common centroid constraint.

Proof: First of all, consider any pair of blocks a_{ij} and b_{ij} in this group. The x-coordinate of a_{ij}'s center is $a_{ij}.x + width(a_{ij})/2$ and that of b_{ij}'s center is $b_{ij}.x + width(b_{ij})/2 = w(G_i) - a_{ij}.x - width(a_{ij})/2$. The center of these two blocks is thus at $x = w(G_i)/2$. Similarly, we can compute the y-coordinate of the center of these two blocks and it is at $y = h(G_i)/2$. Therefore, all pairs will have the same center, which is $(w(G_i)/2, h(G_i)/2)$. Last, for the single block b_c at the center, if there exists such a block, the coordinates of its LL corner will be computed as $(w(G_i)/2 - w(b_c)/2, h(G_i)/2 - h(b_c)/2)$ as described above. Thus, its center will also be at $(w(G_i)/2, h(G_i)/2)$. Therefore, the packing of this group will satisfy the common centroid constraint.

VII. Simulated Annealing

Globally, we use SP as the representation. In the global SP, each block is denoted by a label, except that each common centroid group G_i is regarded as one super-block, whose internal structure is manipulated by the aforementioned C-CBL representation. Simulated annealing is used as our basic searching engine.

A. Set of Moves

We employ the following set of moves to perturb a current candidate solution. The moves can be divided into two categories.

TABLE I
Benchmark Circuits for Comparison

Data	Block	# of Sym. Mod.	Total Area
ami33	33	6	$1.16\,\text{mm}^2$
ami49	49	4	$35.45\,\text{mm}^2$
biasynth_2p4g	65	$8+12+5$	$4.7\,\mu\text{m}^2$
lnamixbias_2p4g	110	$16+6+6+12+4$	$46.00\,\mu\text{m}^2$

1) *Global SP Perturbations:* This category of moves is used to perturb the global SP representation. Starting with a SP satisfying the symmetry condition $Q4$, another candidate SP satisfying $Q4$ can be generated.

1) *Swapping two symmetry groups:* two symmetry groups are picked randomly and swapped. Notice that we do not consider interleaving of symmetry groups in our implementation, so this operation is well-defined.
2) *Swapping two blocks of the same symmetry group:* two blocks A and B, which are not symmetry pair of each other, are picked randomly from a symmetry group. We then swap A and B in s_1 and swap $sym(A)$ and $sym(B)$ in s_2. Notice that the blocks A and B can be self-symmetry or belong to a symmetry pair.
3) *Moving an asymmetric block:* in this move, an asymmetric block (including the super-blocks) is picked randomly and its position in the sequence s_1 or s_2 is modified.
4) *Rotating a symmetry group:* a symmetry group is picked randomly and its orientation is changed (from horizontal to vertical, or vice versa). To perform this rotation, we only need to reverse the order of the related blocks in s_2.
5) *Changing aspect ratio of a soft block:* a soft block A that is not a super-block is picked randomly and its aspect ratio is changed. If A belongs to a symmetry pair, we also need to make the corresponding change to $sym(A)$.

2) *C-CBL Perturbations:* This category of moves modifies the internal structure of a common centroid group by perturbing its C-CBL representation and the shape of the blocks in this group.

1) *Changing aspect ratio:* a randomly chosen pair of blocks in an arbitrary common centroid group are selected and have their shapes changed.
2) *Swapping two devices:* two pairs are randomly chosen from an arbitrary group and their positions in the corresponding S list are swapped.
3) *Changing orientation:* a randomly chosen bit in the L list of an arbitrary group is toggled.
4) *Changing T-junction information:* the value at a randomly chosen position in the T list of an arbitrary group is changed. The new value must not exceed the number of T-junctions available.

B. Cost Function and Annealing Schedule

We use the cost function $cost(F) = area(F) + \lambda \times wire(F)$ to evaluate a packing F where $area(F)$ is the area of F, and $wire(F)$ is the total wire length estimated by the half perimeter

TABLE II

COMPARISON WITH PREVIOUS WORKS

Data	SP [1]		SP+LP [8]		Plantage [19]		SFSP [20]		This Paper	
	Area (mm^2)	Time (s)	Area (mm^2)	Time (s)	Area (mm^2)	Time (s)	Area (mm^2)	Time (s)	Area (mm^2)	Time (s)
ami33	1.24	684	–	–	–	–	1.22	2	1.24	23
ami49	37.82	2038	–	–	–	–	37.05	5.1	38.32	29
biasynth_2p4g	–	–	4960	206	4933	337	4945	13.5	5570	134
lnamixbias_2p4g	–	–	50 150	3027	49 533	387	48 530	53	52 210	227

All experiments were performed on Pentium 4 3.2 GHz 1 GB RAM, except SP on Sun Sparc Ultra-60 433 MHz and symmetric feasible sequence pair (SFSP) on Intel Xeon 2.2 GHz 1 GB RAM.

TABLE III

COMPARING C-CBL APPROACH AND THE DUMMY NODE APPROACH

Data Set	Block No.	CC Groups		Net No.	C-CBL				Dummy Node Approach[a]			
		No.	Device No.		Area	Dead Space (%)	HPWL	Run Time (s)	Area	Dead Space (%)	HPWL	Run Time (s)
c1	30	1	8	36	513.4	4.32	326.3	4.92	518.3	5.23	327.4	98.50
c2	50	1	10	62	821.6	6.83	574.6	10.72	825.5	7.26	573.2	215.7
c3	70	2	8, 10	104	1241	6.92	612.3	20.53	1287	10.3	653.9	451.6
c4	100	3	8, 9, 10	138	1835	8.63	685.7	40.46	1922	12.8	714.5	1272
c5	120	4	10, 10, 10, 10	187	2673	9.34	839.2	52.62	–	–	–	–
c6	150	5	10, 10, 10, 10, 10	242	3269	10.8	1135	108.3	–	–	–	–
c7	200	5	10, 10, 10, 10, 10	379	4103	12.6	1979	172.6	–	–	–	–
c8	300	6	10, 10, 10, 10, 10, 10	622	5062	16.1	2437	563.2	–	–	–	–
Average					–	9.44	–	–	–	–	–	–

[a]No result is reported for data c5–c8 because the running times are too long.

method. The parameter λ is a factor that specifies the relative importance between area and wire length. Before the annealing process, a random walk with 1000 moves will be performed to determine the value of the parameter λ in the cost function. In this preprocessing step, we will perform 1000 random moves to estimate the average area cost A_{avg} and the average wire length W_{avg}. We will then compute λ in such a way that the ratio of $A_{avg} : \lambda \times W_{avg}$ is approximately equal to 1 : 1. In our annealing engine, the temperature is set to 10^5 at the beginning and will drop at a rate of 0.99. At each temperature, n random moves are performed, where n is the number of blocks in the data set. The annealing process stops when the temperature falls below 10^{-5}.

VIII. EXPERIMENTAL RESULTS

Our placer was implemented in C and run on a Sun Blade 2500 with a 1.6 GHz CPU and 2 GB RAM. All the data sets used are either real industrial analog designs (*biasynth_2p4g* and *lnamixbias_2p4g*), derived from the Microelectronics Center of North Carolina (MCNC) benchmarks (*ami33* and *ami49*) or randomly generated by us for testing purpose (*c1-8*, *D40a-b*, *D70a-d*, *D100a-d*, and *c1_arr-c8_arr*).

A. Comparisons with Previous Approaches

We compare with several representative previous works on handling symmetry constraints. Table I describes detailed information of the data used for comparison. The first two data sets are based on the MCNC benchmarks and the last two are real industrial analog circuits commonly used in many previous works on analog placement. The second column shows the total number of blocks, the third column shows the symmetry group information, e.g., for data set *biasynth_2p4g*, there are three symmetry groups with 8, 12, and 5 modules, respectively. The fourth column shows the total area of all the modules. The comparison is shown in Table II. Some results are not available and they are denoted by "-." We can see that SFSP [20] performs the best in terms of both area and runtime. However, it does not consider other general placement constraints.

TABLE IV

HANDLING MIXED PLACEMENT CONSTRAINTS

Data Set	Block No.	CC Group No.	Sym. Group No.	General Const. No.	Area	Dead Space (%)	Run Time (s)
D40a	40	1	1	4	74 557	5.69	79.44
D40b	40	2	1	3	70 708	6.29	77.34
D70a	70	1	4	6	17 924	5.63	749.92
D70b	70	2	3	6	18 355	8.67	497.64
D70c	70	3	2	10	18 894	11.28	318.37
D70d	70	4	1	6	18 986	11.71	1098.63
D100a	100	2	4	6	29 860	9.31	1257.21
D100b	100	3	3	10	30 023	7.26	990.68
D100c	100	3	3	6	30 783	9.11	980.33
D100d	100	4	2	8	29 983	6.68	808.35
Average	–	2.5	2.4	6.5	–	8.163	685.791

B. Experiment One

In order to study the effectiveness of using the C-CBL representation for common centroid groups, we have implemented another direct method to solve the same problem. One straightforward way to handle common centroid constraint is

TABLE V

COMPARISON OF C-CBL WITH GRID-BASED APPROACH

Data Set	Block No.	CC Groups		Net No.	C-CBL Approach				Grid-based Approach			
		No.	Area Ratios of Each Group		Area	Dead Space (%)	HPWL	Run Time (s)	Area	Dead Space (%)	HPWL	Run Time (s)
c1_arr	30	1	{2, 4, 8, 16}	36	247.5	4.24	121.1	7.18	249.2	4.88	118.3	4.46
c2_arr	50	1	{2, 4, 8, 16}	62	266.8	3.54	165.9	18.7	265.3	3.02	152.6	13.17
c3_arr	70	1	{2, 4, 8, 16}	104	868.9	5.01	622.7	39.9	865.8	4.67	588.4	27.48
c4_arr	100	1	{2, 4, 8, 16}	138	1171	6.31	919.3	96.75	1165	5.88	920.1	85.0
c5_arr	120	2	{4, 8, 16}, {2, 4, 8, 16}	187	1267	5.78	1279	199.6	1270	6.03	1204	150.2
c6_arr	150	2	{4, 8, 16}, {2, 4, 8, 16}	242	1495	8.02	1795	352.4	1489	7.65	1680	257.3
c7_arr	200	2	{4, 8, 16}, {2, 4, 8, 16}	379	2093	6.98	3472	739.6	2096	7.11	3382	510.3
c8_arr	300	2	{4, 8, 16}, {2, 4, 8, 16}	622	3179	8.20	6482	2310	3162	7.72	6435	1857
Average					1323.5	6.01	1857.1	470.5	1320.2	5.87	1810	363.2

by extending the approach in Section V on symmetry constraint. The extension can be done by simply adding dummy vertices and edges with variable weights to both the vertical and horizontal constraint graphs. In symmetry constraint, a pair of blocks are required to be placed symmetrically with respect to an axis; thus, a pair of edges weighted zero are inserted between the two blocks in the vertical or horizontal constraint graph to align them, while in the other constraint graph, a dummy node representing the axis will be added together with some equally weighted constraint edges between the dummy node and the two blocks. To satisfy the common centroid constraint, pairs of blocks are placed symmetrically in both the x and y directions. This can be achieved by adding a dummy node d_i, representing the centroid $C(G_i)$, to both the horizontal and vertical constraint graphs and adding a set of constraint edges between d_i and the two blocks to ensure equidistance from the center point in both directions. However, this approach is time consuming since the variable edge weights have to be determined in each annealing iteration. Table III is a comparison between this direct method and our C-CBL approach. Results show that our C-CBL approach can perform much better. In Table III, the column *Block No.* refers to the number of blocks without common centroid constraints, and the column *CC Groups* shows the information of the common centroid groups, e.g., for data set *c3*, there are two common centroid groups with eight and ten devices, respectively (note that each of them will be split into two resulting in a total of eight and ten pairs). The deadspace percentage is 9.44% on average.

C. Experiment Two

In this second set of experiments, we want to study the performance of our placer when there is a mixed set of constraints. The results of this set of experiments are shown in Table IV, where the columns *CC Group No.*, *Sym. Group No.*, and *General Const. No.* display the number of common centroid groups, the number of symmetry groups, and the number of general placement constraints in the data sets, respectively. The average deadspace percentage is 7.16%, with all placement constraints satisfied, which confirm the effectiveness of our placer to handle the common centroid constraints, symmetry constraints, and other general placement constraints, simultaneously.

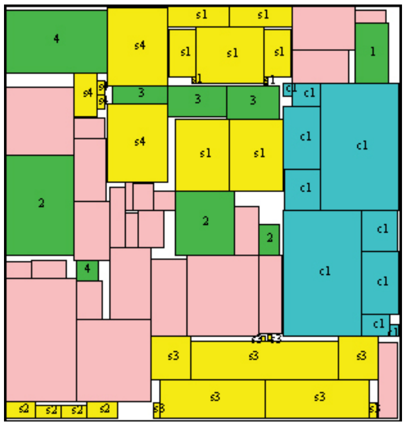


Fig. 10. Resultant packing of data set D70a with one common centroid group ($c1$), four symmetry groups ($s1$–$s4$), range constraint (group 1), alignment constraint (group 2), abutment constraint (group 3), and maximum separation constraint (group 4).

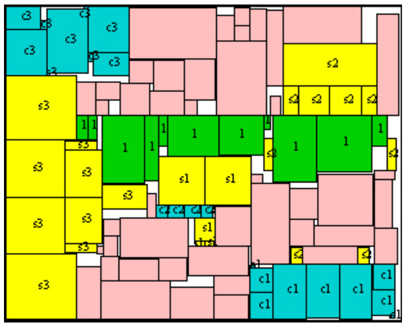


Fig. 11. Resultant packing of data set D100b with three common centroid groups ($c1$–$c3$), three symmetry groups ($s1$–$s3$), and alignment constraint (group 1).

Figs. 10 and 11 are several resultant packings generated by our placer. In these figures, the blocks in the common centroid groups are colored in light blue and labeled with "c#" ("#" denotes its group index), while the blocks with symmetry constraints are colored in yellow and are labeled with "s#." The blocks with general placement constraints are colored in green, and the blocks labeled with the same number belong to the same general constraint group, e.g., the blocks labeled with "3" in Fig. 10 belong to an abutment constraint group.

D. Comparison with the Grid-Based Approach in [12]

Finally, we compare the C-CBL approach with the grid-based approach in [12] on handling arrays of sub-devices with regular shape and size. Notice that the grid-based approach in [12] is designed specifically for this purpose of placing a set of regular sub-devices in a 2-D array. In the grid-based approach, a collection of feasible placement solutions for each group will first be generated and stored in a look-up table. Different solutions will be picked from the look-up table during the moves of the annealing process. The result is shown in Table V. We can see that the results in area and wirelength are very close. The C-CBL approach takes longer runtime. This is reasonable since the grid-based approach is designed specifically to handle regular sub-device arrays. However, this grid-based approach can only handle these cases when the sub-devices have the same shape and dimension, while the C-CBL approach is a general one that can handle sub-devices with arbitrary shapes and dimensions.

IX. Conclusion

In this paper, an analog placement framework was proposed to handle mixed placement constraints. The symmetry and general placement constraints were solved by inserting additional constraint edges properly into the constraint graphs. The common centroid groups were regarded as super-blocks, and their internal structures were manipulated by a novel C-CBL representation. Extensive experimental results demonstrated the effectiveness of our approach.

References

[1] F. Balasa and K. Lampaert, “Symmetry within the sequence-pair representation in the context of placement for analog design,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 7, pp. 712–731, Jul. 2000.

[2] F. Balasa, S. C. Maruvada, and K. Krishnamoorthy, “On the exploration of the solution space in analog placement with symmetry constraints,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 2, pp. 177–191, Feb. 2004.

[3] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, “B*-Trees: A new representation for non-slicing floorplans,” in *Proc. 37th ACM/IEEE Des. Automat. Conf.*, 2000, pp. 458–463.

[4] J. Cohn, D. Garrod, R. A. Rutenbar, and L.R. Carley, “KOAN/ANAGRAMII: New tools for device-level analog layout,” *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 330–342, Mar. 1991.

[5] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley, *Analog Device-Level Layout Automation*. Norwell, MA: Kluwer, 2000.

[6] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, “An O-Tree representation of non-slicing floorplan and its applications,” in *Proc. 36th ACM/IEEE Des. Automat. Conf.*, 1999, pp. 268–273.

[7] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu, “Corner block list: An effective and efficient topological representation of non-slicing floorplan,” in *Proc. Int. Conf. Comput.-Aided Des.*, 2000, pp. 8–12.

[8] S. Kouda, C. Kodama, and K. Fujiyoshi, “Improved method of cell placement with symmetry constraints for analog IC layout design,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 4, pp. 659–668, Apr. 2007.

[9] K. Lampaert, G. Gielen, and W. Sansen, “A performance-driven placement tool for analog integrated circuits,” *IEEE J. Solid-State Circuits*, vol. 30, no. 7, pp. 773–780, Jul. 1995.

[10] J.-M. Lin and Y.-W. Chang, “TCG-S: Orthogonal coupling of P*-admissible representations for general floorplans,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 6, pp. 968–980, Jun. 2004.

[11] P.-H. Lin, Y.-W. Chang, and S.-C. Lin, “Analog placement based on novel symmetry-island formulation,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 6, pp. 791–804, Jun. 2009.

[12] Q. Ma and E. F. Y. Young, “Analog placement with common centroid constraints,” in *Proc. Int. Conf. Comput.-Aided Des.*, 2007, pp. 579–585.

[13] E. Malavasi, E. Charbon, E. Felt, and A. Sangiovanni-Vincentelli, “Automation of IC layout with analog constraints,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 8, pp. 923–942, Aug. 1996.

[14] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, “Rectangle-packing-based module placement,” in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1995, pp. 472–479.

[15] Y.-X. Pang, F. Balasa, K. Lampaert, and C.-K. Cheng, “Block placement with symmetry constraints based on the O-tree nonslicing representation,” in *Proc. 37th ACM/IEEE Des. Automat. Conf.*, 2000, pp. 464–467.

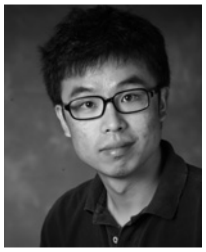
[16] Y. C. Tam, E. F. Y. Young, and C. C. N. Chu, “Analog placement with symmetry and other placement constraints,” in *Proc. Int. Conf. Comput.-Aided Des.*, 2006, pp. 349–354.

[17] G.-M. Wu, J.-M. Lin, Y.-W. Chang, and R.-H. Chuang, “Placement with symmetry constraints for analog layout design using TCG-S,” in *Proc. IEEE Asia South Pacific Des. Automat. Conf.*, 2005, pp. 1135–1138.

[18] E. F. Y. Young, C. C. N. Chu, and M. L. Ho, “Placement constraints in floorplan design,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 7, pp. 735–745, Jul. 2004.

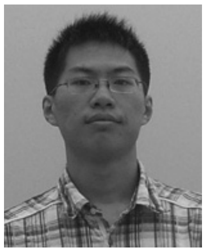
[19] M. Strasser, M. Eick, H. Grab, U. Schlichtmann, and F. M. Johannes, “Deterministic analog circuit placement using hierarchically bounded enumeration and enhanced shape functions,” in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 2008, pp. 306–313.

[20] L. Xiao and E. F. Y. Young, “Analog placement with common centroid and 1-D symmetry constraints,” in *Proc. IEEE Asia South Pacific Des. Automat. Conf.*, 2009, pp. 353–360.



Qiang Ma received the B.Eng. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2006, and the M.Phil. degree in computer science from the Chinese University of Hong Kong, Hong Kong, China, in 2008. He is currently pursuing the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana.

His current research interests include physical design of chips, packages, and printed circuit boards.



Linfu Xiao received the Bachelors degree in microelectronics from Fudan University, Shanghai, China, in 2007. He is currently pursuing the Ph.D. degree in computer science from the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong.

His current research interests include very large scale integration computer-aided design, physical design, and analog layout automation.

Yiu-Cheong Tam received the Bachelors degree from the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong, in 2005.

He became a Research Assistant with the VLSI CAD Laboratory, Department of Computer Science and Engineering, Chinese University of Hong Kong, in 2006.

Evangeline F. Y. Young received the B.S. and M.Phil. degrees in computer science from the Chinese University of Hong Kong (CUHK), Shatin, Hong Kong, and the Ph.D. degree from the University of Texas at Austin, Austin, in 1999.

She is currently an Associate Professor with the Department of Computer Science and Engineering, CUHK. Her current research interests include algorithms and computer-aided design of very large scale integration circuits. She is now working actively on floorplanning, placement, routing, and algorithmic designs.

Dr. Young has served on the technical program committees of several major conferences, including ICCAD, ASP-DAC, ISPD, and GLSVLSI, and also the editorial board of IEEE TCAD.