MB*-Tree: A Multilevel Floorplanner for Large-Scale Building-Module Design

Hsun-Cheng Lee, Yao-Wen Chang, *Member, IEEE*, and Hannah Honghua Yang

***Abstract*—In this paper, we present an agglomeratively multilevel floorplanning/placement framework based on the B*-tree representation called *MB*-tree* to handle the floorplanning and packing for large-scale building modules. The MB*-tree adopts a two-stage technique, i.e., clustering followed by declustering. The clustering stage iteratively groups a set of modules based on a cost metric guided by area utilization and module connectivity and at the same time establishes the geometric relations for the newly clustered modules by constructing a corresponding B*-tree for them. The declustering stage iteratively ungroups a set of the previously clustered modules (i.e., perform tree expansion) and then refines the floorplanning/placement solution by using a simulated annealing scheme. In particular, the MB*-tree preserves the geometric relations among modules during declustering, which makes the MB*-tree an ideal data structure for the multilevel floorplanning/placement framework. Experimental results show that the MB*-tree obtains significantly better silicon area and wirelength than previous works. Further, unlike previous works, the MB*-tree scales very well as the circuit size increases.**

***Index Terms*—Floorplanning, layout, multilevel framework, physical design, placement.**

I. Introduction

DESIGN complexities are growing at breathtaking speed with the continued improvement of nanometer IC technologies. On one hand, designs with billions of transistors are already in production (ICs with billions of transistors are even expected within this decade), Internet Protocol modules are widely reused, and a large number of buffer blocks are used for delay optimization as well as noise reduction in nanometer interconnect-driven floorplanning [3], [11], [19], [20], [23], [35], all of which drive the need of a tool to handle large-scale building modules. On the other hand, the highly competitive IC market requires faster design convergence, faster incremental design turnaround, and better silicon area utilization. Efficient and effective design methodology and tools capable of placing and optimizing large-scale modules are essential for such large designs.

Many floorplan representations have been proposed [9], [15], [24]–[28], [31]–[33], [36], [37] in the literature. However, traditional floorplanning/placement algorithms do not scale well as the design size, complexity, and constraints increase, which are mainly due to their inflexibility in handling nonslicing floorplans and/or intrinsically nonhierarchical data structures (representations). The B*-tree, in contrast, has shown an efficient, effective, and flexible data structure for nonslicing floorplans [9]. It is particularly suitable for representing a nonslicing floorplan with large-scale modules and for creating or incrementally updating a floorplan. What is more important is that its binary-tree-based structure directly corresponds to the framework of a hierarchical divide-and-conquer scheme, and thus, the properties inherited from the structure can substantially facilitate the operations for multilevel large-scale building-module floorplanning/placement.

Based on the B*-tree representation, we present an agglomeratively multilevel floorplanning/placement framework called *MB*-tree* to handle the floorplanning and packing for large-scale building modules with high efficiency and quality. MB*-tree is inspired by the success of the agglomeratively multilevel framework in graph/circuit partitioning such as Chaco [16], hMetis [21], and ML [4]; placement such as mPL [6]; hierarchical placement/floorplanning such as BEAR [13]; and routing such as MRS [10], MR [8], [29], MARS [12], and CMR [17], [18]. Unlike multilevel partitioners and placers, however, multilevel floorplanning poses unique difficulties as the *shapes* of modules to be clustered together can significantly affect the area utilization of a floorplan, and a floorplan design within a cluster needs to be explored along with the global floorplan optimization. The clustering approach also helps to directly address floorplan congestion and timing issues, since different clustering algorithms can be developed to localize intermodule communication and reduce the critical path length.

The MB*-tree algorithm adopts a two-stage technique, i.e., clustering followed by declustering. (See Fig. 1 for an illustration of the multilevel framework.) The clustering stage iteratively groups a set of modules (could be basic modules and/or previously clustered modules) based on a cost metric guided by area utilization and module connectivity and at the same time establishes the geometric relations for the newly clustered modules by constructing a corresponding B*-tree. The clustering procedure repeats until a single cluster containing all modules is formed, which is denoted by a one-node B*-tree that bookkeeps the entire multilevel clustering information. For soft modules, we apply Lagrangian relaxation during clustering to determine

Manuscript received April 5, 2003; revised February 28, 2004 and August 23, 2005. The work of Y.-W. Chang was supported by the National Science Council of Taiwan under Grants NSC 93-2215-E-002-009, NSC 93-2220-E-002-001, and NSC 93-2752-E-002-008-PAE. This paper was presented at the 40th ACM/IEEE Design Automation Conference, June 2003. This paper was recommended by Associate Editor T. Yoshimura.

H.-C. Lee is with Synopsys, Taipei 110, Taiwan, R.O.C. (e-mail: gis88526@cis.nctu.edu.tw).

Y.-W. Chang is with the Department of Electrical Engineering and Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C., and also with Waseda University, Tokyo 169-8050, Japan (e-mail: ywchang@cc.ee.ntu.edu.tw).

H. H. Yang is with Strategic CAD Laboratories, Intel Corporation, Hillsboro, OR 97124 USA (e-mail: hyang@ichips.intel.com).

Digital Object Identifier 10.1109/TCAD.2007.891368

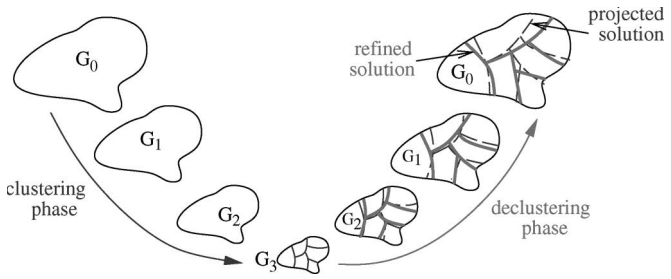


Fig. 1. Multilevel framework.

the module shapes. Then, the declustering stage iteratively ungroups a set of the previously clustered modules (i.e., expanding a node into a subtree according to the B*-tree topology constructed at the clustering stage) and then applies simulated annealing to refine the floorplanning/placement solution based on a cost metric defined by area utilization and wirelength. The refinement shall lead to a "better" B*-tree structure that guides the declustering at the next level. It is important to note that we always keep only one B*-tree for processing at each iteration, and the MB*-tree preserves the geometric relations among modules during declustering (i.e., the tree expansion), which makes the MB*-tree an ideal data structure for the multilevel floorplanning/placement framework. Note that our multilevel framework agglomeratively clusters solutions (i.e., cluster modules one by one) with a postrefinement at each level of the hierarchy, resulting in a linear number of levels. This framework is different from classical multilevel frameworks that simultaneously cluster solutions throughout the design, resulting in a logarithmic number of levels.

Experimental results show that the MB*-tree scales very well as the circuit size increases while the famous previous works, sequence pair (SP), O-tree, and B*-tree alone do not. For circuit sizes ranging from 49 to 9800 modules and from 408 to 81 600 nets, the MB*-tree consistently obtains high-quality floorplans with dead spaces of less than 3.7% in empirically linear runtime, while SP, O-tree, and B*-tree can handle only up to 196, 196, and 1960 modules in the same amount of runtime and result in dead spaces of as large as 13.00% (at 196 modules), 9.86% (at 196 modules), and 27.33% (at 1960 modules), respectively. We also performed experiments based on a large industrial design with 189 modules and 9777 nets. The results show that our MB*-tree algorithm obtained significantly better silicon area and wirelength than previous works.

The remainder of this paper is organized as follows: Section II formulates the module floorplanning/placement problem. Section III gives a brief overview on the B*-tree representation. Section IV presents our two-stage algorithm, i.e., clustering followed by declustering, for the problem addressed in this paper. Section V presents our approach for handling soft modules. Section VI gives the experimental results, and finally, the concluding remarks are given in Section VII.

II. Problem Formulation

Let $M = \{m_1, m_2, \ldots, m_n\}$ be a set of n rectangular modules. Each module $m_i \in M$ is associated with a three tuple (h_i, w_i, a_i), where h_i, w_i, and a_i denote the width, height, and aspect ratio of m_i, respectively. The area A_i of m_i is given by $h_i w_i$, and the aspect ratio a_i of m_i is given by h_i/w_i. Let $r_{i,\min}$ and $r_{i,\max}$ be the minimum and maximum aspect ratios, i.e., $h_i/w_i \in [r_{i,\min}, r_{i,\max}]$. A placement (floorplan) $P = \{(x_i, y_i) | m_i \in M\}$ is an assignment of rectangular modules m_i with the coordinates of their bottom-left corners being assigned to (x_i, y_i) so that no two modules overlap (and $h_i/w_i \in [r_{i,\min}, r_{i,\max}]$ $\forall i$). In this paper, we consider both *hard* and *soft* modules. A hard module is not flexible in its shape but free to rotate. A soft module is free to rotate and change its shape within the range $[r_{i,\min}, r_{i,\max}]$. The objective of placement/floorplanning is to minimize a specified cost metric such as a combination of the area A_{tot} and wirelength W_{tot} induced by the assignment of m_i, where A_{tot} is measured by the final enclosing rectangle of P, and W_{tot} is the summation of half the bounding box of pins for each net (or the center-to-center interconnections among all modules).

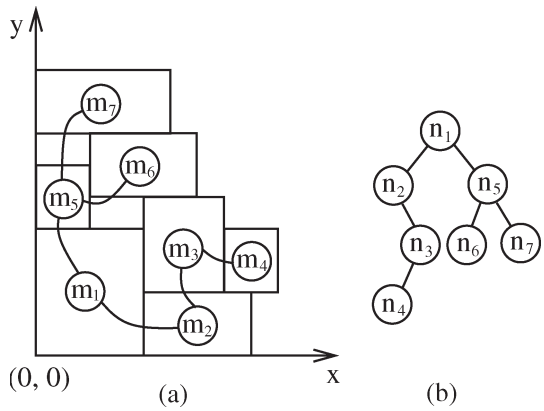


Fig. 2. Admissible placement and its corresponding B*-tree.

III. Review of the B*-Tree Representation

As mentioned earlier, we apply the B*-tree representation to handle the problem of multilevel large-scale building-module floorplanning/placement. Thus, we shall first give a review of the B*-tree representation.

Given a compacted placement P that can neither move down nor move left (called an *admissible placement* [15]), we can represent it by a unique B*-tree T [9]. [See Fig. 2(b) for the B*-tree representing the placement of Fig. 2(a).] A B*-tree is an ordered binary tree (a restriction of O-tree with faster and more flexible operations) whose root corresponds to the module on the bottom-left corner. Using the depth-first search (DFS) procedure, the B*-tree T for an admissible placement P can be constructed in a recursive fashion. Starting from the root, we first recursively construct the left subtree and then the right subtree. Let R_i denote the set of modules located on the right-hand side and adjacent to m_i. The left child of the node n_i corresponds to the lowest module in R_i that is unvisited. The right child of n_i represents the lowest module located above m_i, with its x-coordinate is equal to that of m_i.

As shown in Fig. 2, we make n_1 the root of T since m_1 is on the bottom-left corner. Constructing the left subtree of n_1 recursively, we make n_2 the left child of n_1. Since the left child of n_2 does not exist, we then construct the right subtree

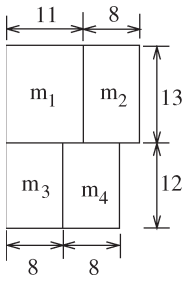


Fig. 3. Cluster with the four primitive modules, a, b, c, and d. The placement can be obtained by applying the clustering scheme $\{\{m_1, m_2\}, \{m_3, m_4\}\}$, resulting in a dead space of 36 units.

of n_2 (which is rooted by n_3). The construction is recursively performed in DFS order. After completing the left subtree of n_1, the same procedure applies to the right subtree of n_1.

Fig. 2(b) illustrates the resulting B*-tree for the placement shown in Fig. 2(a). The construction takes only linear time. The B*-tree keeps the geometric relationship between two modules as follows: If node n_j is the left child of node n_i, module m_j must be located on the right-hand side of m_i, with $x_j = x_i + w_i$. Besides, if node n_j is the right child of n_i, module m_j must be located above module m_i, with the x-coordinate of m_j equal to that of m_i, i.e., $x_j = x_i$. Also, since the root of T represents the bottom-left module, the coordinate of the module is $(x_{\text{root}}, y_{\text{root}}) = (0, 0)$.

Inheriting from the nice properties of ordered binary trees, the B*-tree is simple, efficient, effective, and flexible for handling nonslicing floorplans. It is particularly suitable for representing a nonslicing floorplan with various types of modules and for creating or incrementally updating a floorplan. What is more important is that its binary-tree-based structure directly corresponds to the framework of a hierarchical scheme, which makes it a superior data structure for multilevel large-scale building-module floorplanning/placement.

IV. MB*-Tree Algorithm

In this section, we shall present our MB*-tree algorithm for multilevel large-scale building-module floorplanning/placement. As mentioned earlier, the algorithm adopts a two-stage approach, i.e., clustering followed by declustering, by using the B*-tree representation.

The clustering operation results in two types of modules, namely: 1) *primitive modules* and 2) *cluster modules*. A primitive module m is a module given as an input (i.e., $m \in M$), while a cluster one is created by grouping two or more primitive modules. Each cluster module is created by a *clustering scheme* $\{m_i, m_j\}$, where m_i (m_j) denotes a primitive or a cluster module. Fig. 3 shows a cluster module with four primitive modules; a possible way to form the cluster module is by the clustering scheme $\{\{m_1, m_2\}, \{m_3, m_4\}\}$.

In the following subsections, we detail the two-stage approach of clustering followed by declustering for hard modules.

A. Clustering

The clustering stage iteratively groups a set of (primitive or cluster) modules (say, two modules) based on a cost metric

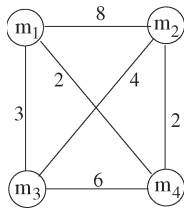


Fig. 4. Example connectivity between each pair of modules. We apply the clustering scheme $\{\{m_1, m_2\}, \{m_3, m_4\}\}$ based on connectivity density instead of $\{\{\{m_1, m_2\}, m_3\}, m_4\}$ (based on connectivity).

defined by area utilization, wirelength, and connectivity among modules, and at the same time establishes the geometric relations among the newly clustered modules by constructing a corresponding B*-subtree. The clustering procedure repeats until a single cluster containing all modules is formed (or the number of modules is smaller than a predefined threshold), which is denoted by a one-node B*-tree that bookkeeps the entire clustering scheme. We shall first consider the clustering metric.

1) Clustering Metric: The clustering metric is defined by the two criteria, namely: 1) area utilization (*dead space*) and 2) the *connectivity density* among modules.

1) Dead space: The area utilization for clustering two modules m_i and m_j can be measured by the resulting dead space s_{ij}, representing the unused area after clustering m_i and m_j. Let s_{tot} denote the dead space in the final floorplan P. We have $s_{\text{tot}} = A_{\text{tot}} - \sum_{m_i \in M} A_i$, where A_i denotes the area of module m_i, and A_{tot} denotes the area of the final enclosing rectangle of P. Since $\sum_{m_i \in M} A_i$ is a constant, minimizing A_{tot} is equivalent to minimizing the dead space s_{tot}. For the example shown in Fig. 3, $s_{12} = 0$, $s_{13} = 36$, and $s_{\text{tot}} = 36$.
2) Connectivity density: Let the connectivity c_{ij} denote the number of nets between two (primitive or cluster) modules m_i and m_j. The *connectivity density* d_{ij} between two modules m_i and m_j is given by

$$d_{ij} = c_{ij}/(n_i + n_j) \tag{1}$$

where n_i (n_j) denotes the number of primitive modules in m_i (m_j). Often, a bigger cluster implies a larger number of connections. The connectivity density considers not only the connectivity but also the sizes of clusters between two modules to avoid possible biases. For the example shown in Fig. 4, we apply the clustering scheme $\{\{m_1, m_2\}, \{m_3, m_4\}\}$ (based on connectivity density) instead of $\{\{\{m_1, m_2\}, m_3\}, m_4\}$ (based on connectivity).

Obviously, the cost function of dead space is for area optimization while that of connectivity density is for timing and wiring area optimization. Therefore, the metric for clustering two (primitive or cluster) modules m_i and m_j, $\phi : \{m_i, m_j\} \rightarrow \Re^+ \cup \{0\}$, is then given by

$$\phi(\{m_i, m_j\}) = \alpha \hat{s}_{ij} + \frac{\beta K}{\hat{d}_{ij}} \tag{2}$$

where $\hat{s}_{ij}$ and $K/\hat{d}_{ij}$ are respective normalized costs for s_{ij} and K/d_{ij}, and α, β, and K are user-specified parameters/constants. We set $K = \sum s_{ij} / \sum d_{ij}$ to normalize the dead

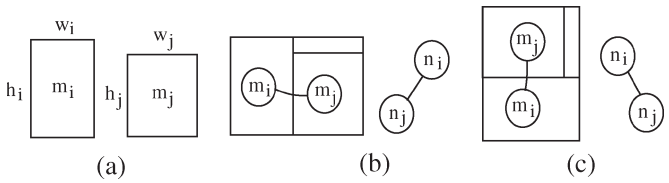


Fig. 5. Relation of two modules and their clustering. (a) Two candidate modules m_i and m_j. (b) Clustering and corresponding B*-subtree for the case where m_i is horizontally related to m_j. (c) Clustering and corresponding B*-subtree for the case where m_i is vertically related to m_j.

space and the connectivity cost, i.e., to make the ranges of the two normalized costs about the same. Note that we shall normalize the dead space and connectivity density to equally weigh the two costs. To calculate the normalization factors for $s_{i,j}$ and $d_{i,j}$, we can preprocess using simulated annealing to derive the initial temperature and then obtain the approximate ranges of the resulting area and connectivity density to normalize the costs. For example, we may perform 100 runs of simulated annealing to obtain the approximate ranges of the resulting costs (i.e., area and connectivity density here) and derive the factors (weights) to equally weigh the costs by making the ranges of the two costs about the same. By doing so, it is more meaningful to weigh the area and connectivity density costs through the controlling factors α and β.

2) Clustering Algorithm: Based on ϕ, we pick a set of modules (say, two modules) with the minimum clustering cost ϕ and cluster them into one. The procedure continues until a single cluster containing all primitive modules is formed or the number of modules is smaller than a given threshold (and thus can be easily handled by a classical floorplanner). During clustering, we shall record how two modules m_i and m_j are clustered into a new cluster module m_k. Fig. 5 shows two ways to cluster two modules m_i and m_j. If m_i is placed left to (below) m_j, then m_i is *horizontally (vertically) related* to m_j, which is denoted by $m_i \rightarrow (\uparrow) m_j$. If $m_i \rightarrow (\uparrow) m_j$, then n_j is the left (right) child of n_i in its corresponding B*-tree. The relation for each pair of modules in a cluster is established and recorded in the corresponding B*-subtree during clustering. It will be used for determining how to expand a node into a corresponding B*-subtree during declustering.

Fig. 6 shows our two-way clustering algorithm. Line 1 computes the initial cost matrix $\Phi = (\phi_{ij})$, where $\phi_{ij} = \alpha \hat{s}_{ij} + \beta K / \hat{d}_{ij}$. Line 2 assigns to n the number of input primitive modules. Lines 3–9 perform step-by-step clustering ($n-1$ steps in total). At Step k, we pick two modules m_i and m_j with the minimum ϕ_{ij} (Extract_Min(ϕ_{ij}) in Line 4) and then cluster them into a new cluster module m_{n+k} (cluster(m_i, m_j) in Line 5). Line 6 records the clustering scheme q_k for $\{m_i, m_j\}$. Line 7 randomly decides the relation of m_i and m_j, and constructs the corresponding B*-subtree. We then update the set of modules to cluster modules (Line 8) and the entries associated with m_{n+k} in the cost matrix Φ (Line 9). We repeat the two-way clustering process $n-1$ times until all modules are clustered into a single cluster. The clustering scheme q_{n-1} for the last two modules bookkeeps the entire clustering scheme Q. Thus, we assign q_{n-1} to Q (Line 10) and return the entire scheme (Line 11).

Algorithm: Clustering(M, α, β, K)
Input: M—the set of primitive modules;
Output: Q—the clustering scheme.

1 Compute the cost matrix, $\Phi = (\phi_{ij})$, where $\phi_{ij} = \alpha \hat{s}_{ij} + \frac{\beta K}{\hat{d}_{ij}}$;
2 $n \leftarrow |M|$;
3 **for** $k \leftarrow 1$ **to** $n-1$ **do**
4 Extract_Min(ϕ_{ij});
5 $m_{n+k} \leftarrow$ cluster(m_i, m_j);
6 $q_k \leftarrow \{m_i, m_j\}$;
7 Randomly decide the relation of m_i and m_j and construct the corresponding B*-subtree;
8 $M \leftarrow M \cup \{m_{n+k}\} \backslash \{m_i, m_j\}$;
9 Update Φ whose entries are associated with m_{n+k};
10 $Q \leftarrow q_{n-1}$;
11 **return** Q;

Fig. 6. Two-way clustering algorithm.

B. Declustering

The declustering stage iteratively ungroups a set of previously clustered modules (i.e., expanding a node into a subtree according to the B*-tree topology constructed at the clustering stage) and then refines the floorplan solution based on a simulated annealing scheme. The refinement shall lead to a "better" B*-tree structure that guides the declustering at the next level. It is important to note that we always keep only one B*-tree for processing at each iteration, and the agglomeratively multilevel B*-tree-based floorplanner preserves the geometric relations among modules during declustering (i.e., the tree expansion), which makes the B*-tree an ideal data structure for the multilevel floorplanning framework.

We shall first introduce the metric used in simulated annealing for refining floorplan/placement solutions.

1) Declustering Metric: The declustering metric is defined by the two criteria, namely: 1) area utilization (*dead space*) and 2) the *wirelength* among modules.

1) Dead space: Same as that defined in Section IV-A.
2) Wirelength: The wirelength of a net is measured by half the bounding box of all the pins of the net or by the length of the center-to-center interconnections between the modules if no pin positions are specified. The wirelength for clustering two modules m_i and m_j, i.e., w_{ij}, is measured by the total wirelength interconnecting the two modules. The total wirelength in the final floorplan P, i.e., w_{tot}, is the summation of the length of the wires interconnecting all modules.

Obviously, the cost function of dead space is for area optimization while that of wirelength is for timing and wiring area optimization. Therefore, the metric for refining a floorplan solution during declustering $\psi_{\text{tot}} : M \rightarrow \Re^+ \cup \{0\}$ is then given by

$$\psi_{\text{tot}} = \gamma \hat{s}_{\text{tot}} + \delta \hat{w}_{\text{tot}} \tag{3}$$

where $\hat{s}_{\text{tot}}$ and $\hat{w}_{\text{tot}}$ are the respective normalized costs for s_{tot} and w_{tot}, and γ and δ are user-specified parameters. Note that

Algorithm: Declustering(m_k, m_i, m_j)
Input: m_k—the cluster module;
m_i, m_j—two modules with m_i right to or below m_j;
1 $parent(n_i) \leftarrow parent(n_k)$;
2 **if** ($n_k = left(parent(n_k))$) **then**
3 $left(parent(n_k)) \leftarrow n_i$;
4 **else**
5 $right(parent(n_k)) \leftarrow n_i$;
6 **if** ($m_i \rightarrow m_j$) **then**
7 $left(n_i) \leftarrow n_j$;
8 $parent(n_j) \leftarrow n_i$;
9 $right(n_j) \leftarrow NIL$;
10 $right(n_i) \leftarrow right(n_k)$;
11 **if** ($right(n_k) \neq NIL$) **then**
12 $parent(right(n_k)) \leftarrow n_i$;
13 $left(n_j) \leftarrow left(n_k)$;
14 **if** ($left(n_k) \neq NIL$) **then**
15 $parent(left(n_k)) \leftarrow n_j$;
16 **if** ($m_i \uparrow m_j$) **then**
17 $right(n_i) \leftarrow n_j$;
18 $parent(n_j) \leftarrow n_i$;
19 $right(n_j) \leftarrow right(n_k)$;
20 **if** ($right(n_k) \neq NIL$) **then**
21 $parent(right(n_k)) \leftarrow n_j$;
22 let $n_a \in \{m_i, m_j\}$ and $a \neq b$ such that $h_a \geq h_b$;
23 $left(n_a) \leftarrow left(n_k)$;
24 **if** ($left(n_k) \neq NIL$) **then**
25 $parent(left(n_k)) \leftarrow n_a$;
26 $left(n_b) \leftarrow NIL$;

Fig. 7. Declustering algorithm.

the normalization procedure for s_{tot} and w_{tot} is similar to that described in Section IV-A1.

2) Declustering Algorithm: The declustering stage iteratively ungroups a set of previously clustered modules (i.e., expand a node into a subtree according to the B*-tree constructed at the clustering stage) and then refines the floorplan solution based on simulated annealing.

Fig. 7 shows the algorithm for declustering a cluster module m_k into two modules m_i and m_j that are clustered into m_k at the clustering stage. Without loss of generality, we make m_i right to or below m_j. In Algorithm Declustering (see Fig. 7), $parent(n_i)$, $right(n_i)$, and $left(n_i)$ denote the parent, right child, and left child of node n_i in a B*-tree, respectively. Line 1 updates the parent of n_k as that of n_i. Lines 2–5 make n_i a left (right) child if n_k is a left (right) child. Lines 6–13 deal with the case where m_i is horizontally related to m_j. If $m_i \rightarrow m_j$, then n_j is the left child of n_i, and thus, we update the corresponding links in Line 7. Lines 8–10 (11–13) update the links associated with the right (left) child of n_k. Similarly, Lines 14–23 cope with the case where m_i is vertically related to m_j.

Fig. 8 gives an illustration of this algorithm. Fig. 8(a) shows an instance of clustering and its corresponding B*-tree, for which we are preparing to decluster m_3 into m_6 and m_7 (i.e., the clustering scheme for m_3 is $\{m_6, m_7\}$). Fig. 8(b) shows four cases to decluster m_3, and their corresponding resulting B*-trees are illustrated in Fig. 8(c). Cases 1 and 2 correspond to Lines 6–13 of Fig. 7, and Cases 3 and 4 correspond to Lines 14–23.

Theorem 1: Each declustering operation takes $O(1)$ time, and the overall declustering stage takes $O(|M|)$ time, where $|M|$ is the number of input primitive modules.

Proof: As listed in Algorithm Declustering (see Fig. 7), each declustering operation requires updating only local links associated with the three involved modules (m_i, m_j, and m_k). Since there are only a constant number of such links, performing a declustering operation takes $O(1)$ time. Further, it is trivial that we perform $|M| - 1$ declustering operations to ungroup all modules, and the overall declustering complexity thus follows. ■

We proposed a simulated annealing-based algorithm to refine the solution at each level of declustering. We apply the following three operations to perturb a multilevel B*-tree (a feasible solution) to another.

1) Op1: Rotate a module.
2) Op2: Move a module to another place.
3) Op3: Swap two modules.

Op1 exchanges the width and height of a module. Op2 deletes a node of a B*-tree and inserts it into another position. Op3 deletes two nodes and inserts them into the corresponding positions in the B*-tree. Obviously, Op2 and Op3 need to perform the deletion and insertion operations on a B*-tree, which takes $O(h)$ time, where h is the height of the B*-tree.

The annealing procedure uses a parameter, i.e., temperature t, to control the probability of accepting an uphill move (an inferior solution). The initial temperature $t_0 = \Delta_{\text{avg}}/\ln(P)$, where Δ_{avg} is the average cost change for a set of randomly generated uphill moves, and P is the initial probability of accepting uphill moves. The temperature t is then decreased by a factor $r < 1$ (i.e., the temperature for the next iteration is rt). We terminate the annealing process when the temperature cools down to a user-defined value ε.

The simulated annealing algorithm starts by a B*-tree produced during declustering. Then, it perturbs a B*-tree (a feasible solution) to another B*-tree by Op1, Op2, and/or Op3 until a predefined "frozen" state is reached. At last, we transform the resulting B*-tree to the corresponding final admissible placement.

C. Overall MB*-Tree Algorithm

The MB*-tree algorithm integrates the aforementioned three algorithms and is summarized in Fig. 9. In Line 1, we first perform clustering to reduce the problem size level by a level based on the clustering metric described in Section IV-A1 and then enter the declustering stage. In the declustering stage, we perform floorplanning for the modules at each level using the simulated annealing-based algorithm B*-tree. At level i, we perform the declustering i^2 times and then perform simulated annealing with $i \times p$ tries per iteration, where p is a user-specified parameter. Therefore, the number of tries for each iteration of simulated annealing is proportional to the number of (primitive and cluster) modules at the current level, leading to a better tradeoff between scalability and solution quality since the MB*-tree can inherit a "good" solution from the previous level.

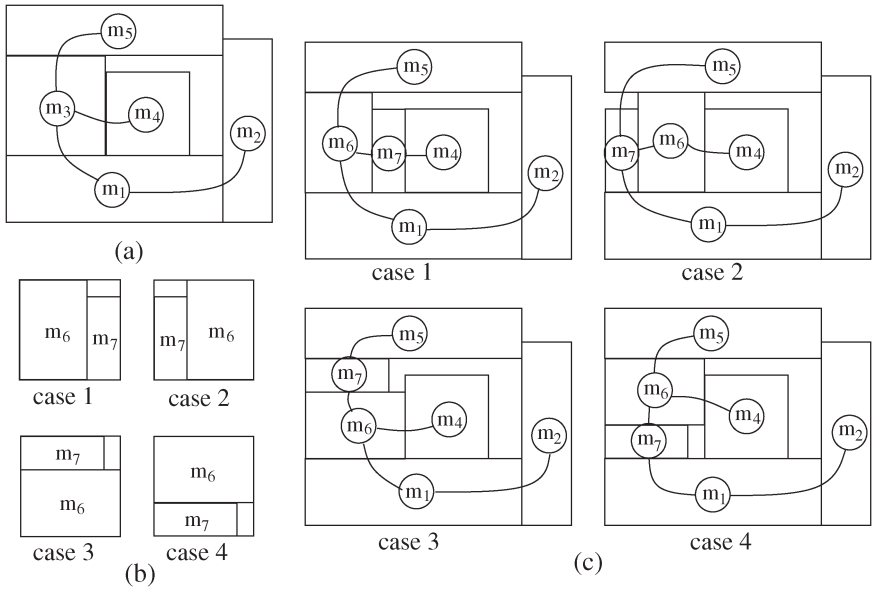


Fig. 8. Declustering m_3 into m_6 and m_7. (a) Configuration before declustering. (b) Four cases to decluster m_6 and m_7. (c) Placement and corresponding B*-tree topology after declustering.

Algorithm: MB*-tree(M, N)
Input: M—the primitive modules;
N—Nets;
Stage I: Clustering
Perform the clustering algorithm described in Section IV-A;
Stage II: Declustering
$i \leftarrow 1$; $k \leftarrow 0$; $n \leftarrow |M|$;
while $i \leq \lceil \lg n \rceil$ **do**
  $j \leftarrow 1$;
  while $j \leq i^2$ and $k \leq n-1$ **do**
    Perform the declustering algorithm described in Section IV-B;
    $j \leftarrow j+1$; $k \leftarrow k+1$;
  Perform the simulated annealing based algorithm, whose number of tries for each iteration is $i \times p$, where p is a user defined parameter;
  $i \leftarrow i+1$;
return

Fig. 9. MB*-tree algorithm.

Fig. 10 illustrates an execution of the MB*-tree algorithm. For explanation, we cluster three modules each time in Fig. 10. Fig. 10(a) lists seven modules to be packed, m_i, $1 \leq i \leq 7$. Fig. 10(b)–(d) illustrates the execution of the clustering algorithm. Fig. 10(b) shows the resulting configuration after clustering m_5, m_6, and m_7 into a new cluster module m_8 (i.e., the clustering scheme of m_8 is $\{\{m_5, m_6\}, m_7\}$). Similarly, we cluster m_1, m_2, and m_4 into m_9 by using the clustering scheme $\{\{m_2, m_4\}, m_1\}$. Finally, we cluster m_3, m_8, and m_9 into m_{10} by using the clustering scheme $\{\{m_3, m_8\}, m_9\}$. The clustering stage is thus done, and the declustering stage begins, in which simulated annealing is applied to do the floorplanning. In Fig. 10(e), we first decluster m_{10} into m_3, m_8, and m_9 [i.e., expand the node n_{10} into the B*-subtree illustrated in Fig. 10(e)]. We then move m_8 to the top of m_9 (perform Op2 for m_8) during simulated annealing [see Fig. 10(f)]. As shown in Fig. 10(g), we further decluster m_9 into m_1, m_2, and m_4, and then rotate m_2 and move m_3 on top of m_2 (perform Op1 on m_2 and Op2 on m_3), resulting in the configuration shown in Fig. 10(h). Finally, we decluster m_8 shown in Fig. 10(i) to m_5, m_6, and m_7, and move m_4 to the right of m_3 (perform Op2 for m_4), which results in the optimum placement shown in Fig. 10(j).

V. Extension to Soft Module Handling

In this section, we present our approach for handling soft modules. We first apply Lagrangian relaxation [38] to cluster soft modules at the clustering stage while keeping declustering the same as before. We then propose a network-flow-based algorithm for projecting Lagrange multipliers to satisfy their optimality conditions.

A. Clustering Metric for Soft Modules

The clustering metric for soft modules is defined by the two criteria, namely: 1) area utilization (*dead space*) and 2) the distance between modules obtained from the computation of Lagrangian relaxation.

1) Dead space: Same as that defined in Section IV-A.
2) Distance: In Lagrangian relaxation, we formulate dead space and wirelength as the objective function. Thus, after the computation of Lagrangian relaxation to be described in Section V-C, we can obtain the distances of two cluster modules i and j, denoted by t_{ij}, via their coordinates computed by Lagrangian relaxation.

Therefore, the metric for clustering two soft (primitive or cluster) modules m_i and m_j, i.e., $\phi_s : \{m_i, m_j\} \rightarrow \Re^+ \cup \{0\}$, is then given by

$$\phi_s(\{m_i, m_j\}) = \alpha \hat{s}_{ij} + \beta \hat{t}_{ij} \tag{4}$$

where $\hat{s}_{ij}$ and $\hat{t}_{ij}$ are the respective normalized costs for s_{ij} and t_{ij}, and α and β are the user-specified parameters. The

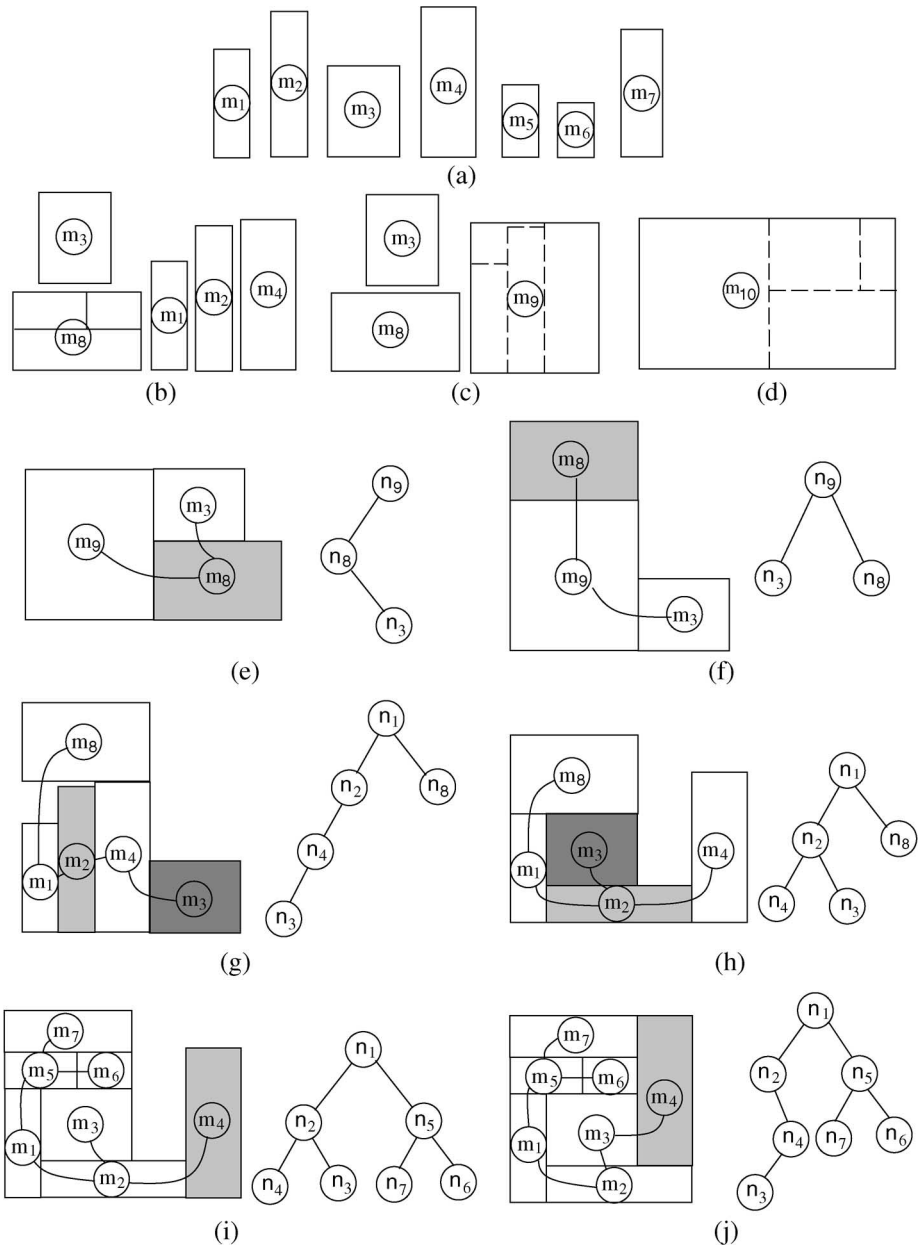


Fig. 10. (a) Given seven modules, m_i, $1 \leq i \leq 7$. (b) Clusters m_5, m_6, and m_7 into m_8. (c) Clusters m_1, m_2, and m_4 into m_9. (d) Clusters m_3, m_8, and m_9 into m_{10}. (e) Decluster m_{10} to m_3, m_8, and m_9. (f) Perform Op2 for m_8. (g) Decluster m_9 to m_1, m_2, and m_4. (h) Perform Op1 and Op2 for m_2 and m_3, respectively. (i) Decluster m_8 to m_5, m_6, and m_7. (j) Perform Op2 for m_4.

procedure to normalize the s_{ij} and t_{ij} costs is similar to that described in Section IV-A1.

Based on ϕ_s, we perform the clustering algorithm as before and then employ the simulated annealing-based algorithm, which is described in Section IV-B, for the floorplanning.

B. Formulation

Let $M = \{m_1, m_2, \ldots, m_n\}$ be a set of n primitive *soft* modules. Each primitive soft module $m_i \in M$ is associated with a three tuple (h_i, w_i, a_i), where h_i, w_i, and a_i denote the width, height, and aspect ratio of m_i, respectively. The area A_i of m_i is given by $h_i w_i$, and the aspect ratio a_i of m_i is given by $h_i/w_i \in [r_{i,\min}, r_{i,\max}]$. Let $L_i = \sqrt{A_i/r_{i,\min}}$ and $U_i = \sqrt{A_i/r_{i,\max}}$ denote the minimum and maximum width of m_i, respectively. We have $h_i = A_i/w_i$ and $L_i \leq w_i \leq U_i$.

A cluster module m_c is composed of a set of primitive soft modules M_p. m_c can be reshaped via reshaping the modules in M_p without violating the relations of the modules in M_p. We create two dummy modules m_s and m_t, and set $x_s = 0$, $y_s = 0$, $w_s = 0$, and $h_s = 0$. Then, we construct horizontal and vertical constraint subgraphs of m_c, denoted by $G_{\rm hc}$ and $G_{\rm vc}$, respectively. $G_{\rm hc}$ and $G_{\rm vc}$ are constructed as follows.

1) For m_s and m_t, create two vertices v_s and v_t in both $G_{\rm hc}$ and $G_{\rm vc}$.
2) For each $m_p \in M_p$, create a vertex v_p in $G_{\rm hc}$ and $G_{\rm vc}$.
3) For each m_p, $m_q \in M_p$, if m_p is left to (below) m_q, create an edge $e(p, q)$ from v_p to v_q in $G_{\rm hc}(G_{\rm vc})$.

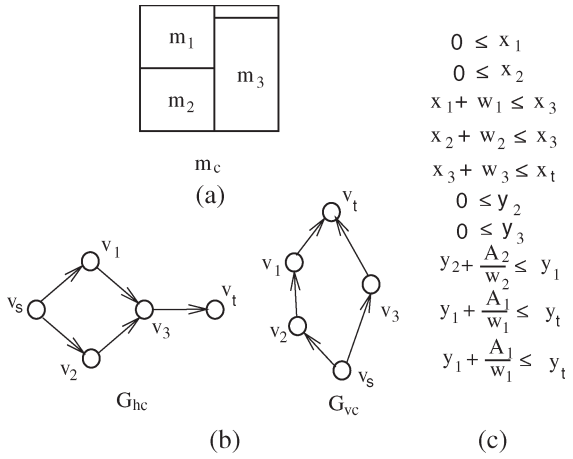


Fig. 11. (a) Cluster module m_c with the cluster scheme $\{\{m_1, m_2\}, m_3\}$. (b) m_c corresponding constraint subgraphs $G_{\rm hc}$ and $G_{\rm vc}$. (c) Constraints to ensure that no relation of modules is violated.

4) For each m_p that is placed at the left boundary (bottom boundary), create an edge $e(v_s, v_p)$ from v_s to v_p in $G_{\rm hc}(G_{\rm vc})$.
5) For each m_p that is placed at the right boundary (top boundary), create an edge $e(v_p, v_t)$ from v_p to v_t in $G_{\rm hc}$ $(G_{\rm vc})$.

If $x_p + w_p \le x_q \forall e(p,q) \in G_{\rm hc}$ and $y_p + (A_p/w_p) \le y_q \forall e(p,q) \in G_{\rm vc}$ are satisfied, the relations of the modules in M_p will not be violated. Fig. 11 illustrates how to construct $G_{\rm hc}$ and $G_{\rm vc}$, and what corresponding constraints must be satisfied. Fig. 11(a) shows a cluster module m_c with the cluster scheme $\{\{m_1, m_2\}, m_3\}$. Fig. 11(b) shows the corresponding constraint subgraphs $G_{\rm hc}$ and $G_{\rm vc}$ of m_c. Fig. 11(c) shows the constraints to ensure that no relation of modules is violated. Thus, it implies that $w_c \ge x_t$ and $h_c \ge y_t$.

At level i, let $M^i = \{m_1^i, m_2^i, \ldots, m_{n_i}^i\}$ denote the set of cluster modules. For each $m_j^i \in M^i$, (x_j^i, y_j^i) denote the coordinate of its bottom-left corner, and h_j^i and w_j^i denote the height and width of m_j^i, respectively. Note that x_j^i, y_j^i, h_j^i, and w_j^i are non-negative real numbers. For convenience, we additionally create two variables, namely: $x_{n_i+1}^i$ and $y_{n_i+1}^i$, which denote the estimated height and width of the chip at level i, respectively. Thus, the estimated area of the chip at level i equals $x_{n_i+1}^i y_{n_i+1}^i$. To estimate wirelength, we adopt the quadratic of the length of the complete graph of pins in a net and take the center of a module as the location of a pin, if the pins are not assigned during floorplanning. Let E^i denote the set of nets at level i. For a net $e_j^i \in E^i$, e_j^i can be represented as a set of the modules $\{m_k^i | e_j^i$ has a pin connecting to $m_k^i\}$. Thus, the estimated wirelength ϖ_j^i of a net $e_j^i \in E^i$ is defined by

$$\varpi_j^i = \sum_{m_p^i, m_q^i \in e_j^i} \Big(\big((x_p^i + w_p^i/2) - (x_q^i + w_q^i/2) \big)^2 + \big((y_p^i + h_p^i/2) - (y_q^i + h_q^i/2) \big)^2 \Big).$$

We use the cost function ϕ' to guide the clustering of soft modules as

$$\phi'(\vec{x}, \vec{y}) = \alpha x_{n_i+1}^i \, y_{n_i+1}^i + \beta \sum_{e_j^i \in E^i} \varpi_j^i \qquad (5)$$

where $\vec{x}$ and $\vec{y}$ denote the respective vectors of the x-coordinate and y-coordinate of a module, α and β are nonnegative user-defined parameters, and ϖ_j^i denotes the estimated wirelength of a net e_j^i. In the formulation of clustering for soft modules, we have the constraints that all modules are not overlapped and must be laid in the chip (i.e., $x_j^i + w_j^i \le x_{n_i+1}^i$ and $y_j^i + h_j^i \le y_{n_i+1}^i$). Therefore, we can formulate the problem of clustering for soft modules, called *CS*, as follows:

$$\begin{aligned}
\text{Minimize} \quad & \alpha x_{n_i+1}^i y_{n_i+1}^i + \beta \sum_{e_j^i \in E^i} \varpi_j^i \\
\text{subject to} \quad & x_j^i + w_j^i \le x_{n_i+1}^i \quad \forall 1 \le j \le n_i, \\
& y_j^i + h_j^i \le y_{n_i+1}^i \quad \forall 1 \le j \le n_i, \\
& x_{t_j} \le w_j^i, \; y_{t_j} \le h_j^i \quad \forall 1 \le j \le n_i, \\
& x_p + w_p \le x_q \quad \forall e(p,q) \in G_{hj} \quad \forall 1 \le j \le n_i, \\
& y_p + \frac{A_p}{w_p} \le y_q \quad \forall e(p,q) \in G_{vj} \quad \forall 1 \le j \le n_i, \\
& L_i \le w_i \le U_i \quad \forall 1 \le i \le n
\end{aligned}$$

where α and β are nonnegative user-defined parameters.

C. Lagrangian Relaxation

Then, the Lagrangian relaxation subproblem associated with the multiplier $\vec{\mathcal{P}} = (\vec{\kappa}, \vec{\eta}, \vec{\lambda}, \vec{\mu}, \vec{r}, \vec{s})$, denoted by $LRS/(\vec{\mathcal{P}})$, can be defined as follows:

$$\begin{aligned}
\text{Minimize} \quad & \alpha x_{n_i+1}^i y_{n_i+1}^i + \beta \sum_{e_j^i \in E^i} \varpi_j^i \\
& + \sum_{j=1}^{n_i} \kappa_j \left(x_j^i + w_j^i - x_{n_i+1}^i \right) \\
& + \sum_{j=1}^{n_i} \eta_j \left(y_j^i + h_j^i - y_{n_i+1}^i \right) \\
& + \sum_{j=1}^{n_i} \sum_{e(p,q) \in G_{hj}} \lambda_{jpq} (x_p + w_p - x_q) \\
& + \sum_{j=1}^{n_i} \sum_{e(p,q) \in G_{vj}} \mu_{jpq} \left(y_p + \frac{A_p}{w_p} - y_q \right) \\
& + \sum_{j=1}^{n_i} r_j \left(x_{t_j} - w_j^i \right) + s_j \left(y_{t_j} - h_j^i \right) \\
\text{subject to} \quad & L_i \le w_i \le U_i \quad \forall 1 \le i \le n.
\end{aligned}$$

Let $Q(\vec{\mathcal{P}})$ denote the optimal value of $LRS/(\vec{\mathcal{P}})$. The Lagrangian dual problem (LDP) of *CS* can be defined as follows:

$$\begin{aligned}
\text{Maximize} \quad & Q(\vec{\mathcal{P}}) \\
\text{subject to} \quad & \vec{\mathcal{P}} \ge 0.
\end{aligned}$$

Since *CS* can be transformed into a convex problem, we can apply the theorem in [5, Th. 6.2.4]. This implies that if $\vec{\mathcal{P}}$ is an

optimal solution to LDP, the optimal solution of $LRS/(\vec{\mathcal{P}})$ will also optimize *CS*.

Consider the Lagrangian ζ of *CS* defined as follows:

$$\begin{aligned}\zeta =& \alpha x^i_{n_i+1} y^i_{n_i+1} + \beta \sum_{e^i_j \in E^i} \varpi^i_j \\ &+ \sum_{j=1}^{n_i} \kappa_j \left(x^i_j + w^i_j - x^i_{n_i+1}\right) \\ &+ \sum_{j=1}^{n_i} \eta_j \left(y^i_j + h^i_j - y^i_{n_i+1}\right) \\ &+ \sum_{j=1}^{n_i} \sum_{e(p,q)\in G_{hj}} \lambda_{jpq}(x_p + w_p - x_q) \\ &+ \sum_{j=1}^{n_i} \sum_{e(p,q)\in G_{vj}} \mu_{jpq}\left(y_p + \frac{A_p}{w_p} - y_q\right) \\ &+ \sum_{j=1}^{n_i} \left(r_j\left(x_{t_j} - w^i_j\right) + s_j\left(y_{t_j} - h^i_j\right)\right) \\ &+ \sum_{i=1}^{n} u_i(L_i - w_i) + \sum_{i=1}^{n} v_i(w_i - U_i).\end{aligned}$$

The Kuhn–Tucker conditions imply that the optimal solution of *CS* must be at $\partial\zeta/\partial x_p = 0$ and $\partial\zeta/\partial y_p = 0$. Thus, we only need to consider the multipliers $\vec{\mathcal{P}}$ that satisfy these conditions. Therefore, for $1 \le p \le n$, we have

$$\partial\zeta/\partial x_p = \sum_{j=1}^{n_i}\left(\sum_{e(p,q)\in G_{hj}} \lambda_{jpq} - \sum_{e(q,p)\in G_{hj}} \lambda_{jqp}\right) = 0 \quad (6)$$

$$\partial\zeta/\partial y_p = \sum_{j=1}^{n_i}\left(\sum_{e(p,q)\in G_{hj}} \mu_{jpq} - \sum_{e(q,p)\in G_{hj}} \mu_{jqp}\right) = 0. \quad (7)$$

D. Solving $LRS/(\vec{\mathcal{P}})$ and LDP

Let Ω denote the set of multipliers $\vec{\mathcal{P}}$ satisfying (6) and (7). We now consider solving the Lagrangian relaxation subproblem $LRS/(\vec{\mathcal{P}})$ for a given $\vec{\mathcal{P}} \in \Omega$, i.e., computing the dimension and coordinate of each module. First, we partially differentiate ζ with respect to w_i to get an optimal value of w_i such that ζ is minimized, i.e.,

$$\begin{aligned}\partial\zeta/\partial w_i = (v_p - u_p) &+ \sum_{j=1}^{n_i}\left(\sum_{q\in \mathrm{out}_{G_{hj}}(v_p)} \lambda_{jpq}\right) \\ &- \sum_{j=1}^{n_i}\left(\sum_{q\in \mathrm{out}_{G_{vj}}(v_p)} \mu_{jpq}\frac{A_p}{w_p^2}\right) = 0.\end{aligned}$$

Thus, we have

$$w_p = \sqrt{\frac{\sum_{j=1}^{n_i}\sum_{q\in \mathrm{out}_{G_{vj}}(v_p)} \mu_{jpq} A_p}{(v_p - u_p) + \sum_{j=1}^{n_i}\sum_{q\in \mathrm{out}_{G_{hj}}(v_p)} \lambda_{jpq}}}$$

where $\mathrm{out}_G(v) = \{u | e(v,u) \in E(G)\}$. Recall that $L_p \le w_p \le U_p$, $1 \le p \le n$. Thus, the optimal w_p^* can be computed by $w_p^* = \min\{U_p, \max\{L_p, w_p\}\}$.

Since the dimension of each primitive module (w_p and h_p) has been determined, the dimension of each cluster module (w^i_j and h^i_j) can be computed by applying a longest path algorithm in G_{hj} and G_{vj}. Then, we consider partial differentiation of ζ with respect to x^i_j and y^i_j, giving the optimality conditions of *CS*. Therefore, for $1 \le j \le n_i$, we have

$$\begin{aligned}\partial\zeta/\partial x^i_j = \beta\Bigg(&\sum_{e^i_k \supset \{m^i_j\}} 2\left(|e^i_k| - 1\right) x^i_j - \sum_{e^i_k \supset \{m^i_j\}} \sum_{m^i_l \in e^i_k \setminus \{m^i_j\}} x^i_l \\ &+ \sum_{e^i_k \supset \{m^i_j\}} \sum_{m^i_l \in e^i_k \setminus \{m^i_j\}} \left(w^i_j - w^i_l\right)\Bigg) + \sum_{j=1}^{n_i} \kappa_j = 0\end{aligned} \quad (8)$$

$$\begin{aligned}\partial\zeta/\partial y^i_j = \beta\Bigg(&\sum_{e^i_k \supset \{m^i_j\}} 2\left(|e^i_k| - 1\right) y^i_j - \sum_{e^i_k \supset \{m^i_j\}} \sum_{m^i_l \in e^i_k \setminus \{m^i_j\}} y^i_l \\ &+ \sum_{e^i_k \supset \{m^i_j\}} \sum_{m^i_l \in e^i_k \setminus \{m^i_j\}} \left(h^i_j - h^i_l\right)\Bigg) + \sum_{j=1}^{n_i} \eta_j = 0\end{aligned} \quad (9)$$

where $|e^i_k|$ denotes the number of pins of e^i_k.

In (8), there are n_i equations with n_i variables. Thus, we can apply Gaussian elimination to solve these n_i equations with n_i variables to get the optimal value of x^i_j. In these n_i equations, all the coefficients of variables depend only on the net information (i.e., e^i_k). Since the net information is the same through the entire process, each variable can be solved by the same process. Hence, we can record the process of solving each variable during the first iteration (which takes cubic time), and then each subsequent computation will take only quadratic time by applying the same process. Similarly, we can compute the optimal value of y^i_j. After the dimensions and coordinates of all modules are computed, then we can get the dimension of the chip, $x^i_{n_i+1}$ and $y^i_{n_i+1}$, since all modules are within the chip, i.e., $x^i_{n_i+1} = \max\{x^i_j + w^i_j\}$, and $y^i_{n_i+1} = \max\{y^i_j + h^i_j\}$ for all cluster modules m^i_j.

Next, we use a subgradient optimization method to search for the optimal $\vec{\mathcal{P}}$. Let $\vec{\mathcal{P}}$ be a multiplier at step k. We move $\vec{\mathcal{P}}$ to a new multiplier $\vec{\mathcal{P}}'$ based on the subgradient direction

$$\begin{aligned}\kappa'_j &= \left[\kappa_j + \rho_k\left(x^i_j + w^i_j - x^i_{n_i+1}\right)\right]^+ \\ \eta'_j &= \left[\eta_j + \rho_k\left(y^i_j + h^i_j - y^i_{n_i+1}\right)\right]^+ \\ \lambda'_{jpq} &= \left[\lambda_{jpq} + \rho_k\left(x_p + w_p - x_q\right)\right]^+ \\ \mu'_{jpq} &= \left[\mu_{jpq} + \rho_k\left(y_p + \frac{A_p}{w_p} - y_q\right)\right]^+\end{aligned}$$

where $[x]^+ = \max\{x, 0\}$, and ρ_k is a step size such that $\lim_{k\to\infty}\rho_k = 0$ and $\sum_{k=1}^{\infty}\rho_k = \infty$.

After updating $\vec{\mathcal{P}}$, we need to project $\vec{\mathcal{P}}'$ to $\vec{\mathcal{P}}* \in \Omega$ and then solve the Lagrangian relaxation subproblem $LRS/(\vec{\mathcal{P}}*)$ by the above algorithm until the solution converges.

E. Projecting Lagrange Multipliers

We present a network-flow-based algorithm to check whether $\vec{\mathcal{P}}$ belongs to Ω and to project $\vec{\mathcal{P}}$ to $\vec{\mathcal{P}}* \in \Omega$, if $\vec{\mathcal{P}} \notin \Omega$. Further, an increamental update technique is employed to make the maximum flow computation more efficient. For each cluster module m_c, we first create two networks N_{hc} (for $G_{\rm hc}$) and N_{vc} (for $G_{\rm vc}$) as follows.

1) For each $v_i \in V(G_{\rm hc})(V(G_{\rm vc}))$, create a vertex v_i' in $N_{hc}(N_{vc})$ and make v_s' and v_t' as the source and sink, respectively.
2) For each $e(p,q) \in E(G_{\rm hc})(E(G_{\rm vc}))$, create a corresponding edge $e(p',q')$ with capacity $\lambda_{cpq}(\mu_{cpq})$ in $N_{hc}(N_{vc})$.

We apply the maximum flow computation on the networks to check whether $\vec{\mathcal{P}}$ belongs to Ω. The maximum flow computation finds an augmenting path from v_s' to v_t' and then pushes flow on it until no argument path can be found. Let cap(v,u) and flow(v,u) denote the capacity and flow on the edge $e(v,u)$. An edge $e(v,u)$ is saturated if its capacity equals the flow (i.e., cap$(v,u) =$ flow(v,u)).

Theorem 2: If all edges in the networks are saturated, $\vec{\mathcal{P}} \in \Omega$.

Proof: After the maximum flow computation, for each v_p' in a network except the source and sink, the sum of the flows of v_p' incoming edges equals the sum of its outgoing ones (i.e., $\sum_{e(p',q') \in N_{hc}} \text{flow}(p',q') = \sum_{e(q',p') \in N_{hc}} \text{flow}(q',p')$ for each N_{hc} and $\sum_{e(p',q') \in N_{vc}} \text{flow}(p',q') = \sum_{e(q',p') \in N_{vc}} \text{flow}(q',p')$ for each N_{vc}). Besides, cap$(p',q') =$ flow(p',q') for all edges $e(p',q')$ (all edges are saturated), and cap(p',q') of each edge $e(p',q')$ in $N_{hc}(N_{vc})$ equals $\lambda_{cpq}(\mu_{cpq})$. Hence, $\sum_{e(p,q) \in G_{\rm hc}} \lambda_{cpq} = \sum_{e(q,p) \in N_{hc}} \lambda_{cqp}$ and $\sum_{e(p,q) \in G_{\rm vc}} \mu_{cpq} = \sum_{e(q,p) \in G_{\rm vc}} \mu_{cqp}$ for each cluster module m_c. $\vec{\mathcal{P}}$ belongs to Ω. ■

If $\vec{\mathcal{P}}$ does not belong to Ω, we project $\vec{\mathcal{P}}$ to $\vec{\mathcal{P}}*$ by restoring the flow flow(p',q') of each edge $e(p',q')$ in N_{hc} (N_{vc}) to λ_{cpq} (μ_{cpq}) for each m_c.

Theorem 3: $\vec{\mathcal{P}}* \in \Omega$.

The projection process greatly affects the efficiency of the entire optimization, since there may be $O(n^2)$ edges in the worst case. Thus, we employ an incremental flow update technique to speed up the max–flow computation after updating $\vec{\mathcal{P}}$ and its corresponding capacity. Fig. 12 shows an algorithm for the incremental network update. Lines 1–2 check whether each edge $e(p',q')$ violates the capacity constraint (i.e., $0 \le$ flow$(p',q') \le$ cap(p',q')). Lines 3–9 fix the overflow on $e(p',q')$ if an edge $e(p',q')$ violates its capacity constraint. Finally, Line 10 computes a maximum flow again.

Note that, for efficiency consideration, we may perform Lagrangian relaxation only at the higher levels of the agglomeratively multilevel framework (when the number of modules become small enough for Lagrangian relaxation). To do so, however, we still need to pass the information of the aspect ratio for each soft module level by level.

Algorithm: IncreamentalUpate(N,s,t)
Input: N—the flow network;
s—the source of N;
t—the sink of N;
1 **for** each edge $e(p',q') \in E(N)$;
2 **if** $flow(p',q') > cap(p',q')$ **then**
3 $f_{over} \leftarrow flow(p',q') - cap(p',q')$;
4 **while** $f_{over} > 0$ **do**
5 find a path p from s to t, passing through $e(p',q')$, and $\min\{flow(u,v) | e(u,v) \in p\} > 0$.
6 $f_{reduced} \leftarrow \min\{\min\{flow(u,v) | e(u,v) \in p\}, f_{over}\}$;
7 **for** each edge $e(u,v) \in p$
8 $flow(u,v) \leftarrow flow(u,v) - f_{reduced}$;
9 $f_{over} \leftarrow f_{over} - f_{reduced}$;
10 compute maximum flow on N;

Fig. 12. Incremental update algorithm.

TABLE I
BENCHMARK CIRCUITS USED IN OUR EXPERIMENT

Circuit	#modules	#net
ami49_1	49	408
ami49_2	98	865
ami49_4	196	1779
ami49_10	490	4521
ami49_20	980	9091
ami49_40	1960	18231
ami49_60	2940	27371
ami49_80	3920	36511
ami49_100	4900	45651
ami49_150	7350	68501
ami49_200	9800	91351
industry	189	9777

VI. EXPERIMENTAL RESULTS

We implemented the MB*-tree algorithm in C++ language on a 450-MHz SUN Ultra 60 workstation with 2-GB memory. The package is available at http://eda.ee.ntu.edu.tw/research.htm.

Columns 1, 2, and 3 of Table I list the names of the benchmark circuits, the number of modules, and the number of nets, respectively. ami49 is the largest Microelectronics Center of North Carolina benchmark circuit used in the previous works [9], [15] for comparative study. To test the scalability of existing methods, we created ten synthetic circuits, named ami49_x, by duplicating the modules and nets of ami49 by x times. The largest circuit ami49_200 contains 9800 modules and 91 351 nets (specified by pin-to-pin interconnections). Note that the work in [22] simply duplicates all modules and nets of the circuit ami49. However, these kinds of synthetic circuits are not general since there is no interconnection between the duplicated copies of circuits. To avoid possible biases, we also added interconnections among different copies of duplicated circuits. For the circuit ami49_x, we duplicated each module/net x times. For each module m_i, we duplicated it as $m_{i,1}, m_{i,2}, \ldots, m_{i,x}$ and added $x-1$ nets between $(m_{i,1}, m_{i,2})$, $(m_{i,1}, m_{i,3}), \ldots, (m_{i,1}, m_{i,x})$. Also, we divide

TABLE II
COMPARISONS FOR AREA, DEAD SPACE, RUNTIME, AND MEMORY AMONG MB*-TREE, SP, O-TREE, AND B*-TREE. NR: NO RESULT OBTAINED WITHIN 5-h CPU TIME ON SUN SPARC ULTRA 60. NOTE THAT MB*-TREE, SP, AND B*-TREE FINISHED THEIR MEMORY ALLOCATION IN THE VERY EARLY STAGE OF EXECUTION. THEREFORE, THEIR MEMORY CONSUMPTION FOR THE LISTED CIRCUIT SIZES CAN BE MEASURED. O-TREE PERFORMS MEMORY ALLOCATION AND DEALLOCATION DURING EXECUTION; THEREFORE, ONLY THE MEMORY REQUIREMENTS FOR THE SMALL CASES THAT FINISHED EXECUTION ARE AVAILABLE

Circuit	Total	MB*-tree				Sequence Pair [31]			
		Area	Dead space	Time	Mem	Area	Dead space	Time	Mem
	(mm^2)	(mm^2)	(%)	(min)	(MB)	(mm^2)	(%)	(min)	(MB)
ami49_1	35.445	36.46	2.78	0.4	1.3	38.89	8.87	6.86	2.0
ami49_2	70.890	72.72	2.51	2.5	1.5	80.27	11.69	45.51	2.1
ami49_4	141.780	145.73	2.70	2.6	2.1	162.97	13.00	309.00	2.9
ami49_10	354.454	364.14	2.66	5.4	4.6	NR	NR	NR	8.6
ami49_20	708.908	730.95	3.02	15.6	12.2	NR	NR	NR	27.1
ami49_40	1417.816	1472.62	3.72	24.8	37.6	NR	NR	NR	101.0
ami49_60	2126.724	2205.86	3.58	42.2	77.2	NR	NR	NR	222.1
ami49_80	2835.632	2943.72	3.67	57.0	131.4	NR	NR	NR	382.3
ami49_100	3544.540	3671.42	3.45	51.6	200.3	NR	NR	NR	609.2
ami49_150	5316.750	5505.34	3.42	142.2	437.4	NR	NR	NR	1361.0
ami49_200	7089.080	7341.91	3.44	256.2	767.2	NR	NR	NR	NR

Circuit	Total	O-tree [15]				B*-tree			
		Area	Dead space	Time	Mem	Area	Dead space	Time	Mem
	(mm^2)	(mm^2)	(%)	(min)	(MB)	(mm^2)	(%)	(min)	(MB)
ami49_1	35.445	36.77	3.61	10.46	3.6	36.74	3.53	0.25	3.2
ami49_2	70.890	80.82	12.29	70.56	4.6	73.11	3.03	1.26	4.8
ami49_4	141.780	155.76	9.86	179.23	10.0	151.31	6.60	4.73	5.4
ami49_10	354.454	NR	NR	NR	NR	407.32	12.98	19.25	5.6
ami49_20	708.908	NR	NR	NR	NR	870.45	18.55	23.48	7.3
ami49_40	1417.816	NR	NR	NR	NR	1951.04	27.33	53.21	19.0
ami49_60	2126.724	NR	NR	NR	NR	NR	NR	NR	38.1
ami49_80	2835.632	NR	NR	NR	NR	NR	NR	NR	65.2
ami49_100	3544.540	NR	NR	NR	NR	NR	NR	NR	100.1
ami49_150	5316.750	NR	NR	NR	NR	NR	NR	NR	217.4
ami49_200	7089.080	NR	NR	NR	NR	NR	NR	NR	381.2

the block widths/heights by 5 for the benchmarks to avoid overflows in computing the wirelength for ami49_x.

Table II shows the results for ami49_x by optimizing area alone ($\gamma = 1.0$ and $\delta = 0.0$). Columns 2, 3, 4, 5, and 6 give the total area of modules in the circuit, the resulting area, the dead space, the runtime, and the memory requirement for our MB*-tree, respectively. The remaining columns list the results for the well-known previous works, SP [31], O-tree [15], and B*-tree [9]. Note that the B*-tree package we used here is the September 2000 version, *B*-tree-v1.0*, available also at http://eda.ee.ntu.edu.tw/research.htm. It runs 50–100× faster and achieves better area utilization than the B*-tree package reported in [9]. We shall also note that the tools we compared here are all variable-die floorplanners. The well-known floorplanner Parquet-3.1/-4.0 [1], [34] and the floorplacer Capo 9 [2] both target on fixed-die floorplanning, a different floorplanning problem from what we have solved in this paper. We have also tested publicly available mixed-size placers on the floorplan benchmarks, including Feng Shui 2.6/5.0 [14] and mGP [7], [30]. Feng Shui generated the floorplanning results directly using its legalizer without performing global placement. Thus, its results are far from optimal. For mGP, it results in many overlaps and places some modules outside the chip boundary. So we shall not compare our results with those mixed-size placers.

As shown in Table II, our MB*-tree algorithm obtained a dead space of only 2.78% for ami49 in only 0.4-min runtime and 1.3-MB memory, while B*-tree-v1.0 reported a dead space of 3.53% using 0.25-min runtime and 3.2-MB memory. Further, the experimental results for larger circuits show that the MB*-tree scales very well as the circuit size increases while the previous works, i.e., SP, O-tree, and B*-tree, do not. For circuit sizes ranging from 49 to 9800 modules and from 408 to 81 600 nets, the MB*-tree consistently obtains high-quality floorplans with dead spaces of less than 3.72% in empirically linear runtime, while SP, O-tree, and B*-tree can handle only up to 196, 98, and 1960 modules in the same amount of time and result in dead spaces of as large as 13.00% (at 196 modules), 12.29% (at 98 modules), and 27.33% (at 1960 modules), respectively. In Fig. 13, the resulting dead space and runtime are plotted as functions of the circuit size (in the number of modules), respectively. As shown in Table II and Fig. 13(a), the resulting dead spaces for the MB*-tree are almost independent of circuit sizes, which proves the high scalability of the MB*-tree. In contrast, the dead spaces for the nonhierarchical previous works all grow dramatically as the circuit size increases. Fig. 13(b) shows the empirical runtime for the four algorithms. This figure reveals that the empirical runtime of the MB*-tree is the best. In particular, the empirical runtime of the MB*-tree approaches linear in circuit size while the other previous works cannot handle large-scale designs. Fig. 14 shows the layout for the largest circuit ami49_200 obtained by MB*-tree in 256-min CPU time. It has a dead space of only 3.44%. Note that this circuit is not feasible to the previous works [9], [15], [31].

Table III shows the comparisons on area, dead space, and runtime between running the complete MB*-tree scheme and

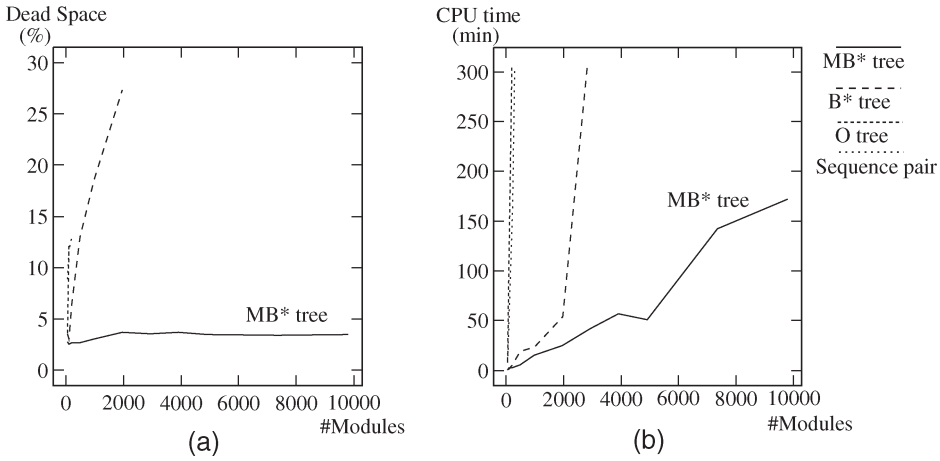


Fig. 13. (a) Comparison for the dead space versus circuit size (number of modules). (b) Comparison for the CPU time versus circuit size (number of modules).

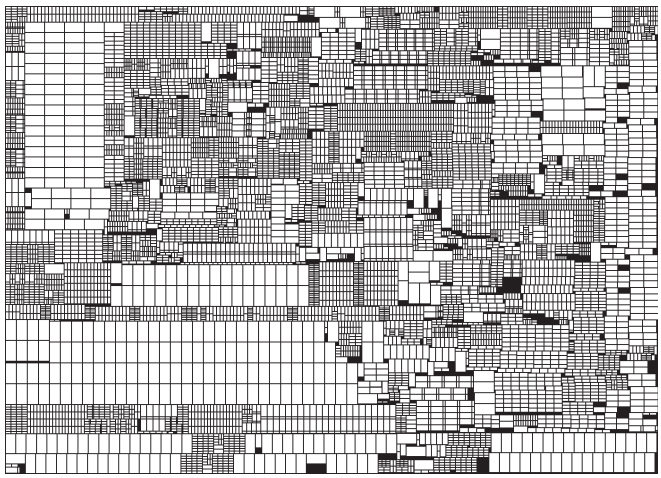


Fig. 14. Layout of ami49_200 (9800 modules, 81 600 nets). Dead space = 3.44%.

only clustering based on the ami49 family circuits for area optimization. The resulting areas right after clustering are on average about 1.78× of the final areas, and the runtimes for clustering range from less than 1% to about 13% of the total runtime. As shown in the table, although several initial placements have around 60% dead spaces, our declustering can consistently reduce the dead spaces to around 3%. The results show the effectiveness of the declustering scheme.

We also tested the scalability of B*-tree and MB*-tree for wirelength optimization. (We shall omit the comparisons with SP and O-tree since they cannot handle this wirelength optimization problem with more than 100 modules well.) Table IV shows the results on half-perimeter wirelength (HPWL), dead space, and CPU time for B*-tree and MB*-tree based on the ami49_x circuits and within 24-h CPU time. As shown in the table, MB*-tree results in significantly smaller wirelength and average dead space than B*-tree for wirelength optimization. In particular, the MB-tree scales very well while the B*-tree does not.

Table V shows the comparisons for area optimization alone ($\gamma = 1.0,\ \delta = 0.0$), wirelength optimization alone ($\gamma = 0.0,\ \delta = 1.0$), and simultaneous area and wirelength optimization ($\gamma = 0.5,\ \delta = 0.5$) among SP, B*-tree, and MB*-tree based on the circuit industry (whose total area = 658.04 mm^2). The circuit industry is a 0.18-μm 1-GHz industrial design with 189 modules, 20 million gates, and 9777 center-to-center interconnections. It is a large chip design and consists of three "tough" modules with aspect ratios greater than 19 (and as large as 36). (Note that we do not have the results for O-tree for this experiment because the data industry cannot be fed into the O-tree package.) In each entry of the table, we list the best/average values obtained in ten runs of simulated annealing using a random seed for each run. For the column "Time," we report the runtime for obtaining the best value and the average runtime of the ten runs. As shown in the table, our MB*-tree algorithm obtained significantly better silicon area and wirelength than SP and B*-tree in all tests. For area optimization, MB*-tree can obtain a dead space of only 2.11% while SP (B*-tree) results in a dead space of at least 28.1% (12.9%). For wirelength optimization, MB*-tree can obtain a total wirelength of only 56 631 mm while SP (B*-tree) requires a total wirelength of at least 81 344 mm (113 216 mm). For simultaneous area and wirelength optimization, MB*-tree also obtains the best area and wirelength. The results show the effectiveness of our MB*-tree algorithm. For the runtimes, MB*-tree is larger than B*-tree and SP for wirelength optimization. (For area optimization, MB*-tree runs faster than SP.) This is reasonable because it took much longer to obtain significantly better results, and the multilevel process incurred some overhead. Nevertheless, as shown in Table II, both SP and B*-tree do not scale well to instances with a large number of modules (and thus their runtimes increase dramatically when the number of modules grows into hundreds). The resulting layout of industry for simultaneous area and wirelength optimization using MB*-tree is shown in Fig. 15.

VII. Concluding Remarks

We have presented the MB*-tree-based agglomeratively multilevel framework to handle the floorplanning and packing for large-scale modules. Experimental results have shown that the MB*-tree scales very well as the circuit size increases. The capability of the MB*-tree shows promise in handling large-scale designs with complex constraints. We propose to explore the

TABLE III
COMPARISONS FOR AREA, DEAD SPACE, AND RUNTIME BETWEEN RUNNING THE COMPLETE MB*-TREE SCHEME AND ONLY CLUSTERING

Circuit	Total	MB*-tree results			MB*-tree results after clustering		
		Area	Dead space	Time	Area	Dead space	Time
	(mm^2)	(mm^2)	(%)	(min)	(mm^2)	(%)	(min)
ami49_1	35.445	36.46	2.78	0.4	42.75	17.09	< 0.01
ami49_2	70.890	72.72	2.51	2.5	134.04	47.11	< 0.01
ami49_4	141.780	145.73	2.70	2.6	167.51	15.36	< 0.01
ami49_10	354.454	364.14	2.66	5.4	585.22	39.43	0.02
ami49_20	708.908	730.95	3.02	15.6	985.52	28.07	0.07
ami49_40	1417.816	1472.62	3.72	24.8	3013.23	52.94	0.38
ami49_60	2126.724	2205.86	3.58	42.2	2707.78	21.46	1.01
ami49_80	2835.632	2943.72	3.67	57.0	7460.29	61.99	2.12
ami49_100	3544.540	3671.42	3.45	51.6	5262.02	32.64	4.13
ami49_150	5316.750	5505.34	3.42	142.2	14692.56	63.81	16.53
ami49_200	7089.080	7341.91	3.44	256.2	18972.45	62.63	32.08

TABLE IV
COMPARISONS FOR HPWL, DEAD SPACE, AND CPU TIME BETWEEN B*-TREE AND MB*-TREE FOR THE ami49_x CIRCUITS. NR: NO RESULT OBTAINED WITHIN 24-h CPU TIME

Circuit	#Module	B*-tree			MB*-tree		
		HPWL	Dead Space	Time	HPWL	Dead Space	Time
	/#Net	($\times$ e6)	(%)	(min)	($\times$ e6)	(%)	(min)
ami49_1	49/408	0.16	32.43	0.5	0.15	30.85	0.5
ami49_2	98/865	0.46	37.44	2.1	0.39	28.75	2.2
ami49_4	196/1779	1.16	43.65	10.0	1.06	31.20	4.0
ami49_10	490/4521	4.28	36.66	84.5	3.79	39.22	10.5
ami49_20	980/9091	1.18	39.88	510.5	1.02	44.73	24.0
ami49_40	1960/18231	NR	NR	NR	2.67	50.24	63.5
ami49_60	2940/27371	NR	NR	NR	4.99	48.64	128.5
ami49_80	3920/36511	NR	NR	NR	7.48	55.28	183.5
ami49_100	4900/45651	NR	NR	NR	10.29	54.18	331.0
ami49_150	7350/68501	NR	NR	NR	18.44	56.08	537.5
ami49_200	9800/91351	NR	NR	NR	33.39	67.80	1124.1

TABLE V
COMPARISONS FOR AREA OPTIMIZATION ALONE, WIRELENGTH OPTIMIZATION ALONE, AND SIMULTANEOUS AREA AND WIRELENGTH OPTIMIZATION AMONG SP, B*-TREE, AND MB*-TREE BASED ON THE CIRCUIT INDUSTRY. IN EACH ENTRY, BOTH BEST/AVERAGE VALUES OBTAINED IN TEN RUNS OF SIMULATED ANNEALING ARE REPORTED. THE LAST TWO ROWS GIVE THE RATIOS OF THE RESULTS (SP TO MB*-TREE AND B*-TREE TO MB*-TREE)

Tool	Area optimization ($\gamma = 1.0$, $\delta = 0.0$)			
	Area (mm^2)	Dead space (%)	Wirelength (mm)	Time (min)
SP	914.5/988.0	28.1/33.2	263289/283632	31.3/37.3
B*-tree	755.7/876.6	12.9/24.7	232624/245676	12.6/9.1
MB*-tree	671.9/740.8	2.1/3.1	188637/186582	346.1/204.7
SP : MB*-tree	1.36/1.34	13.38/10.71	1.40/1.52	0.09/0.18
B*-tree : MB*-tree	1.13/1.18	6.14/7.97	1.23/1.32	0.04/0.05
Tool	Wirelength optimization ($\gamma = 0.0$, $\delta = 1.0$)			
	Area (mm^2)	Dead space (%)	Wirelength (mm)	Time (min)
SP	952.9/1031.2	44.8/56.7	113216/122609	31.3/37.3
B*-tree	831.2/899.0	26.3/36.6	81344/91169	12.6/9.1
MB*-tree	722.6/751.6	9.8/14.2	56631/61698	346.1/204.7
SP : MB*-tree	1.32/1.37	4.57/3.99	2.00/1.98	0.09/0.18
B*-tree : MB*-tree	1.15/1.20	2.68/2.58	1.44/1.48	0.04/0.05
Tool	Simultaneous area and wirelength optimization ($\gamma = 0.5$, $\delta = 0.5$)			
	Area (mm^2)	Dead space (%)	Wirelength (mm)	Time (min)
SP	1104.2/1182.5	40.4/44.3	136001/159340	46.4/37.3
B*-tree	834.7/982.8	21.2/32.5	104558/112200	14.4/9.11
MB*-tree	716.3/740.8	8.1/11.1	67786/67541	221.3/220.2
SP : MB*-tree	1.54/1.60	4.99/3.99	2.01/2.36	0.21/0.17
B*-tree : MB*-tree	1.17/1.33	2.62/2.93	1.54/1.66	0.07/0.04

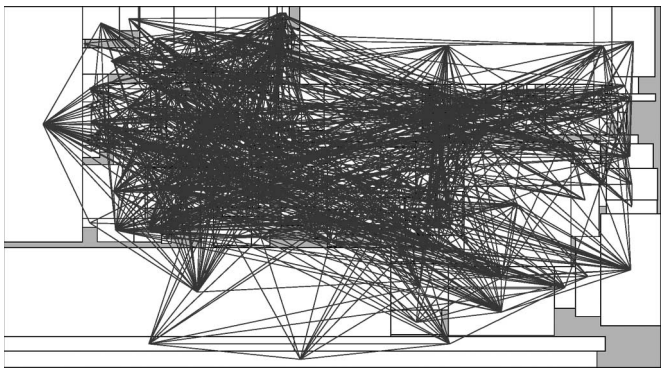


Fig. 15. Layout of industry by simultaneously optimizing area and wirelength ($\gamma = 0.5, \delta = 0.5$). CPU time $= 5234$ s, Area $= 716\,263\,680$ μm^2, Total wirelength $= 67\,786\,296$ μm, Dead space $= 8.14\%$.

floorplanning/placement problem with large-scale rectilinear and mixed-sized modules/cells as well as buffer-block planning for interconnect-driven floorplanning in the future.

REFERENCES

[1] S. Adya and I. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.

[2] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov, "Unification of partitioning, placement and floorplanning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2004, pp. 550–557.

[3] C. J. Alpert, J. H. Hu, S. S. Sapatnekar, and P. G. Villarrubia, "A practical methodology for early buffer and wire resource allocation," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2001, pp. 189–194.

[4] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 8, pp. 655–667, Aug. 1998.

[5] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming Theory and Algorithms*. Hoboken, NJ: Wiley, 1993.

[6] T. F. Chan, J. Cong, T. Kong, and J. R. Shinnerl, "Multilevel optimization for large-scale circuit placement," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2000, pp. 171–176.

[7] C.-C. Chang, J. Cong, and X. Yuan, "Multi-level placement for large-scale mixed-size ic designs," in *Proc. ACM/IEEE Asia and South Pac. Des. Autom.*, Jan. 2003, pp. 325–330.

[8] Y.-W. Chang and S.-P. Lin, "MR: A new framework for multilevel full-chip routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 793–800, May 2004.

[9] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2000, pp. 458–463.

[10] J. Cong, J. Fang, and Y. Zhang, "Multilevel approach to full-chip gridless routing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2001, pp. 396–403.

[11] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 1999, pp. 358–363.

[12] J. Cong, M. Xie, and Y. Zhang, "An enhanced multilevel routing system," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2002, pp. 51–58.

[13] W. W. Dai, B. Eschermann, E. Kuh, and M. Pedram, "Hierarchical placement and floorplanning in bear," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 8, no. 12, pp. 1335–1349, Dec. 1989.

[14] FengShui Standard Cell/Mixed Block/Structure Placer. [Online]. Available: http://vlsicad.cs.binghamton.edu/software.html

[15] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 1999, pp. 268–273.

[16] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graph," in *Proc. Supercomputing*, 1995, pp. 1–24.

[17] T.-Y. Ho, Y.-W. Chang, S.-J. Chen, and D. T. Lee, "A fast crosstalk- and performance-driven multilevel routing system," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, San Jose, CA, Nov. 2003, pp. 382–387.

[18] T.-Y. Ho, Y.-W. Chang, S.-J. Chen, and D. T. Lee, "Multilevel full-chip routing considering crosstalk and performance," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 6, pp. 869–878, Jun. 2005.

[19] H.-R. Jiang, Y.-W. Chang, J.-Y. Jou, and K.-Y. Chao, "Simultaneous floorplanning and buffer block planning," in *Proc. IEEE/ACM Asia and South Pac. Des. Autom. Conf.*, Jan. 2003, pp. 431–434.

[20] H.-R. Jiang, Y.-W. Chang, J.-Y. Jou, and K.-Y. Chao, "Simultaneous floorplanning and buffer block planning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 694–703, May 2004.

[21] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 1999, pp. 343–348.

[22] H.-C. Lee, Y.-W. Chang, J.-M. Hsu, and H. Yang, "Multilevel floorplanning/placement for large-scale modules using B*-trees," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2003, pp. 812–817.

[23] S.-M. Li, Y.-H. Cherng, and Y.-W. Chang, "Noise-aware buffer planning for interconnect-driven floorplanning," in *Proc. IEEE/ACM Asia and South Pac. Des. Autom. Conf.*, Jan. 2003, pp. 423–426.

[24] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closure graph based representation for non-slicing floorplans," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2001, pp. 764–769.

[25] J.-M. Lin and Y.-W. Chang, "TCG-S: Orthogonal coupling of P*-admissible representations for general floorplans," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2002, pp. 842–847.

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

[27] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closure graph based representation for general floorplans," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 2, pp. 288–292, Feb. 2005.

[28] J.-M. Lin, Y.-W. Chang, and S.-P. Lin, "Corner sequence: A P-admissible floorplan representation with a worst-case linear-time packing scheme," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 4, pp. 679–686, Aug. 2003.

[29] S.-P. Lin and Y.-W. Chang, "A novel framework for multilevel routing considering routability and performance," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 2002, pp. 44–50.

[30] MGP: Multilevel Global Placer for large-scale standard-cell placement, mixed-sized (standard-cells mixed with macros) placement for wirelength minimization and routability optimization. [Online]. Available: http://ballade.cs.ucla.edu/mGP/

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

[32] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, Nov. 1996, pp. 484–491.

[33] R. H. J. M. Otten, "Automatic floorplan design," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 1982, pp. 261–267.

[34] PARQUET: Floorplanner for fixed-outline floorplanning and classical min-area block packing. [Online]. Available: http://vlsicad.eecs.umich.edu/BK/parquet/

[35] P. Sarkar, V. Sundararaman, and C. K. Koh, "Routability-driven repeater block planning for interconnect-centric floorplanning," in *Proc. ACM Int. Symp. Phys. Des.*, Apr. 2000, pp. 186–191.

[36] T. C. Wang and D. F. Wong, "An optimal algorithm for floorplan and are optimization," in *Proc. ACM/IEEE Des. Autom. Conf.*, 1990, pp. 180–186.

[37] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 1986, pp. 101–107.

[38] F. Y. Young, C. C. N. Chu, W. S. Luk, and Y. C. Wong, "Floorplan area minimization using Lagrangian relaxation," in *Proc. ACM Int. Symp. Phys. Des.*, Apr. 2000, pp. 174–179.

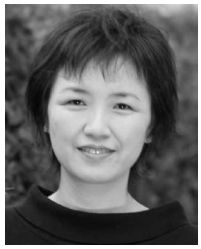
Hsun-Cheng Lee received the B.S. degree in information computer engineering from Chung Yuan Christian University, Chungli, Taiwan, R.O.C., in 1999, and the M.S. degree in computer information science from the National Chiao Tung University, Hsinchu, Taiwan, in 2001.

He is currently a Software Engineer with Synopsys, Taipei, Taiwan. His research interests include physical design and DFM-related topics.

Yao-Wen Chang (S'94–M'96) received the B.S. degree from National Taiwan University, Taipei, Taiwan, in 1988, and the M.S. and Ph.D. degrees from the University of Texas at Austin in 1993 and 1996, respectively, all in computer science.

He is a Professor in the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University. He is currently also a Visiting Professor at Waseda University, Japan. He was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, in the summer of 1994. From 1996 to 2001, he was on the faculty of National Chiao Tung University, Taiwan. His current research interests lie in VLSI physical design, design for manufacturing, and FPGA. He has been working closely with industry on projects in these areas. He has coauthored one book on routing and over 120 ACM/IEEE conference/journal papers in these areas.

Dr. Chang received an award at the 2006 ACM ISPD Placement Contest, Best Paper Award at ICCD-1995, and nine Best Paper Award Nominations from DAC-2007, ISPD-2007 (two), DAC-2005, 2004 ACM TODAES, ASP-DAC-2003, ICCAD-2002, ICCD-2001, and DAC-2000. He has received many awards for research performance, such as the inaugural First-Class Principal Investigator Awards and the 2004 Mr. Wu Ta You Memorial Award from the National Science Council of Taiwan, the 2004 MXIC Young Chair Professorship from the MXIC Corp, and for excellent teaching from National Taiwan University and National Chiao Tung University. He is an editor of the *Journal of Computer and Information Science*. He has served on the ACM/SIGDA Physical Design Technical Committee and the technical program committees of ASP-DAC (topic chair), DAC, DATE, FPT (program co-chair), GLSVLSI, ICCAD, ICCD, IECON (topic chair), ISPD, SOCC (topic chair), TENCON, and VLSI-DAT (topic chair). He is currently an independent board member of Genesys Logic, Inc, the chair of the Design Automation and Testing (DAT) Consortium of the Ministry of Education, Taiwan, a member of the board of governors of the Taiwan IC Design Society, and a member of the IEEE Circuits and Systems Society, ACM, and ACM/SIGDA.



Hannah Honghua Yang received the B.S. degree from Peking University, Beijing, China, in 1988, and the M.S. and Ph.D. degrees from the University of Texas, Austin, in 1991 and 1995, respectively, all in computer science.

She is currently a Senior Staff CAD Researcher with Strategic CAD Laboratory, Intel Corporation, Hillsboro, OR. She is a technical lead on very large scale integration (VLSI) design automation for physical design and microarchitecture planning and exploration. She has published over 30 technical papers in the semiconductor design automation field in premium international conferences and journals.