47.4

Multilevel Floorplanning/Placement for Large-Scale Modules Using B*-trees *

Hsun-Cheng Lee[1], Yao-Wen Chang[2], Jer-Ming Hsu[3], and Hannah H. Yang[4]
[1]Synopsys Inc., Taipei, Taiwan
[2]Dept. of Electrical Engineering & Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
[3]National Center for High-Performance Computing, Hsinchu 300, Taiwan
[4]Strategic CAD Labs, Intel Corporation, Hillsboro, OR 97124, USA

Abstract

We present in this paper a 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, 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, MB*-tree scales very well as the circuit size increases.

Categories and Subject Descriptors: B. 7.2 [**Integrated Circuits**]: Design Aids—Placement and Routing; J.6 [**Computer Applications**]: Computer-Aided Engineering—Computer-Aided Design
General Terms: Algorithms, Design, Performance, Theory
Keywords: Floorplanning, Multilevel Framework, Lagrangian Relaxation

1 Introduction

Design complexities are growing at a breathtaking speed with the continued improvement of the nanometer IC technologies. On one hand, designs with hundreds of million transistors are already in production, IP modules are widely reused, and a large number of buffer blocks are used for delay optimization as well as noise reduction in very deep-submicron interconnect-driven floorplanning [1, 7, 11, 13, 21], which all 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 [5, 9, 14, 15, 16, 18, 19, 20, 22] in the literature. However, traditional floorplanning/placement algorithms do not scale well as the design size, complexity, and constraints increase, mainly due to their inflexibility in handling non-slicing floorplans, and/or intrinsically non-hierarchical data structures (representations). The B*-tree, in contrast, has been shown an efficient, effective, and flexible data structure for non-slicing floorplans [5]. It is particularly suitable for representing a non-slicing floorplan with large-scale modules and for creating or incrementally updating a floorplan. What is more important, 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 in this paper a 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 multilevel framework in graph/circuit partitioning such as Chaco [10], hMetis [12], and ML [2], placement such as MPL [4], and routing such as MRS [6], MR [17], and MARS [8]. 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 inter-module communication and reduce critical path length.

The MB*-tree algorithm adopts a two-stage technique, clustering followed by declustering. 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, 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 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 apply 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.

Experimental results show that the MB*-tree scales very well as the circuit size increases while the famous previous works, sequence pair, O-tree, and B*-tree alone, do not. For circuit sizes ranging from 49 to 9,800 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 sequence pair, O-tree, and B*-tree can handle only up to 196, 196, and 1,960 modules in the same amount of runtime and result in the dead spaces of as large as 13.00% (@ 196 modules), 9.86% (@ 196 modules), and 27.33% (@ 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 2 formulates the module floorplanning/placement problem. Section 3 gives a brief overview on the B*-tree representation. Section 4 presents our two-stage algorithm, clustering followed by declustering. Section 5 presents our approach for handling soft modules. Section 6 gives the experimental results.

2 Problem Formulation

Let $M = \{m_1, m_2, ..., m_n\}$ 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's with the coordinates of their bottom-left corners being assigned to (x_i, y_i)'s so that no two modules overlap (and $h_i/w_i \in [r_{i,min}, r_{i,max}], \forall i$). We consider in this paper 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's, where A_{tot} is measured by the final enclosing rectangle of P and W_{tot} the summation of half the bounding box of pins for each net.

3 Review of the B*-tree Representation

Given a compacted placement P that can neither move down nor move left (called an *admissible placement* [9]), we can represent it by a unique B*-tree T [5]. (See Figure 1(b) for the B*-tree representing the placement of

*This work was supported in part by the National Science Council of Taiwan by Grant No. NSC-91-2215-E-002-038. E-mails: hclee@synopsys.com; ywchang@cc.ee.ntu.edu.tw; gis89530@cis.nctu.edu.tw; hyang@ichips.intel.com.

DAC 2003, June 2–6, 2003, Anaheim, California, USA.

Figure 1(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 equal to that of m_i.

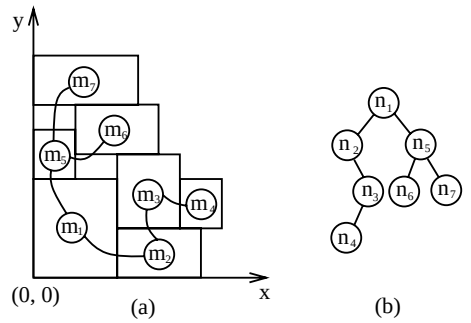

Figure 1: An admissible placement and its corresponding B*-tree.

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 $(x_{root}, y_{root}) = (0, 0)$.

Inheriting from the nice properties of ordered binary trees, the B*-tree is simple, efficient, effective, and flexible for handling non-slicing floorplans. It is particularly suitable for representing a non-slicing floorplan with various types of modules and for creating or incrementally updating a floorplan. What is more important, 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.

4 The 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, clustering followed by declustering, by using the B*-tree representation.

The clustering operation results in two types of modules, namely *primitive modules* and *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.

In the following subsections, we detail the clustering and declustering algorithms for hard modules.

4.1 Clustering

In this stage, we iteratively group a set of (primitive or cluster) modules until a single cluster is formed (or until the number of cluster modules is smaller than a threshold) based on a cost metric of area and connectivity. We shall first consider the clustering metric.

The clustering metric is defined by the two criteria: area utilization (*dead space*) and the *connectivity density* among modules.

- 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 $s_{tot} = A_{tot} - \sum_{m_i \in M} A_i$, where A_i denotes the area of module m_i and A_{tot} 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}.
- Connectivity density: Let the connectivity c_{ij} denote the number of nets between two modules m_i and m_j. The *connectivity denisty* d_{ij} between two (primitive or cluster) modules m_i and m_j is given by

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

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.

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

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

where $\hat{s}_{ij}$ and $K/\hat{d}_{ij}$ are respective normalized costs for s_{ij} and K/d_{ij}, α, β and K are user-specified parameters/constants.

Based on ϕ, we cluster a set of modules into one at each iteration by applying the aforementioned methods 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 the classical program). During clustering, we shall record how two modules m_i and m_j are clustered into a new cluster module m_k. If m_i is placed left to (below) m_j, then m_i is *horizontally (vertically) related* to m_j, 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.

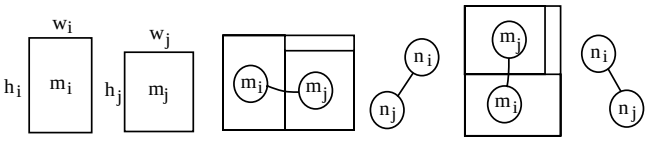

Figure 2: The relation of two modules and their clustering. (a) Two candidate modules m_i and m_j. (b) The clustering and the corresponding B*-subtree for the case where m_i is horizontally related to m_j. (c) The clustering and the corresponding B*-subtree for the case where m_i is vertically related to m_j.

4.2 Declustering

We shall first introduce the metric used in simulated annealing for refining floorplan/placement solutions. The declustering metric is defined by the two creiteria: area utilization (*dead space*) and the *wirelength* among modules.

- Dead space: Same as that defined in Section 4.1.
- Wire length: 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, w_{ij}, is measured by the total wirelength interconnecting the two modules. The total wirelength in the final floorplan P, 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, $\psi_{ij} : \{m_i, m_j\} \rightarrow \Re^+ \cup \{0\}$, is then given by

$$\psi_{ij} \quad = \quad \gamma \hat{s}_{ij} + \delta \hat{w}_{ij}, \tag{3}$$

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

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 $left(n_i) \leftarrow n_j; parent(n_j) \leftarrow n_i; right(n_j) \leftarrow NIL$;
8 $right(n_i) \leftarrow right(n_k)$;
9 **if** ($right(n_k) \neq NIL$) **then**
10 $parent(right(n_k)) \leftarrow n_i$;
11 $left(n_j) \leftarrow left(n_k)$;
12 **if** ($left(n_k) \neq NIL$) **then**
13 $parent(left(n_k)) \leftarrow n_j$;
14 **if** ($m_i \uparrow m_j$) **then**
15 $right(n_i) \leftarrow n_j; parent(n_j) \leftarrow n_i$;
16 $right(n_j) \leftarrow right(n_k)$;
17 **if** ($right(n_k) \neq NIL$) **then**
18 $parent(right(n_k)) \leftarrow n_j$;
19 let $n_a \in \{m_i, m_j\}$ and $a \neq b$ such that $h_a \geq h_b$;
20 $left(n_a) \leftarrow left(n_k)$;
21 **if** ($left(n_k) \neq NIL$) **then**
22 $parent(left(n_k)) \leftarrow n_a$;
23 $left(n_b) \leftarrow NIL$;

Figure 3: The declustering algorithm.

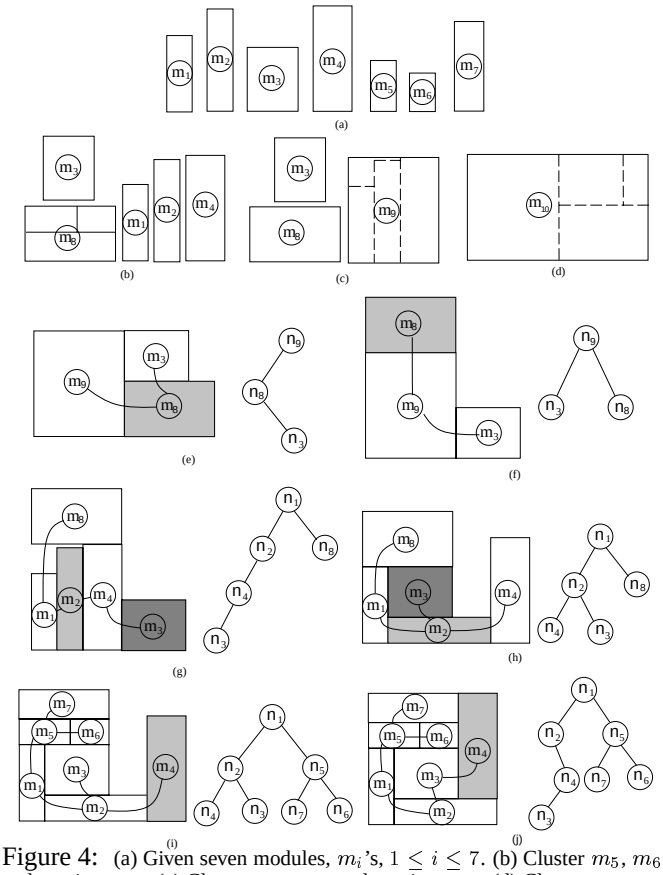

Figure 4: (a) Given seven modules, m_i's, $1 \leq i \leq 7$. (b) Cluster m_5, m_6, and m_7 into m_8. (c) Cluster m_1, m_2, and m_4 into m_9. (d) Cluster m_3, m_8, and m_9 into m_{10}. (e) Decluster m_{10} to m_3, m_8, and m_9. (f) Perform Op2 for m_8. (g) Decluster m_9 to m_1, m_2, and m_4. (h) Perform 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.

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.

Figure 3 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 Figure 3), $parent(n_i)$, $right(n_i)$, and $left(n_i)$ denote the parent, the right child, and the 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 the right (left) child of n_k. Similarly, lines 14–23 cope with the case where m_i is vertically related to m_j.

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.*

4.3 Simulated Annealing

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.

- Op1: Rotate a module.
- Op2: Move a module to another place.
- 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 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.

4.4 The Overall MB*-tree Algorithm

The MB*-tree algorithm integrates the aforementioned three algorithms. We first perform clustering to reduce the problem size level by level 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_SA.

Figure 4 illustrates an execution of the MB*-tree algorithm. For explanation, we cluster three modules each time in Figure 4. Figure 4(a) lists seven modules to be packed, m_i's, $1 \leq i \leq 7$. Figures 4(b)–(d) illustrate the execution of the clustering algorithm. Figures 4(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 Figure 4(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 Figure 4(e)). We then move m_8 to the top of m_9 (perform Op2 for m_8) during simulated annealing (see Figure 4(f)). As shown in Figure 4(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 Figure 4(h). Finally, we decluster m_8 shown in Figure 4(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 Figure 4(j).

5 Handling Soft Modules

In this section, we present our approach for handling soft modules. We first apply Lagrangian relaxation [23] 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.

5.1 Formulation

Let $M = \{m_1, m_2, ..., m_n\}$ 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 the 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 a horizontal and a vertical constraint subgraphs of m_c, denoted by G_{hc} and G_{vc}, respectively. G_{hc} and G_{vc} are constructed as follows:

- For m_s and m_t, create two vertices v_s and v_t in both G_{hc} and G_{vc}.
- For each $m_p \in M_p$, create a vertex v_p in G_{hc} and G_{vc}.
- 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_{hc} (G_{vc}).
- For each m_p which is placed at the left boundary (bottom boundary), create an edge $e(v_s, v_p)$ from v_s to v_p in G_{hc} (G_{vc}).
- For each m_p which is placed at the right boundary (top boundary), create an edge $e(v_p, v_t)$ from v_p to v_t in G_{hc} (G_{vc}).

If $x_p + w_p \leq x_q, \forall e(p,q) \in G_{hc}$ and $y_p + \frac{A_p}{w_p} \leq y_q, \forall e(p,q) \in G_{vc}$ are satisfied, the relations of the modules in M_p will not be violated. Figure 5 illustrates how to construct G_{hi} and G_{vi} and what corresponding constraints must be satisfied. Figure 5(a) gives a cluster modules m_c with the cluster scheme $\{\{m_1, m_2\}, m_3\}$. Figure 5(b) shows the corresponding constraint subgraphs G_{hc} and G_{vc} of m_c. Figure 5(c) shows the constraints to ensure that no relation of modules is violated. Thus, it impies that $w_c \geq x_t$, and $h_c \geq y_t$.

At level i, Let $M^i = \{m_1^i, m_2^i, ..., m_{n_i}^i\}$ 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. For convenience, we additionally create two variables, $x_{n_i+1}^i$ 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 wirelengh ϖ_j^i of a net $e_j^i \in E^i$ is defined by

$$\varpi_j^i = \sum_{m_p^i, m_q^i \in e_j^i} (((x_p^i + w_p^i/2) - (x_q^i + w_q^i/2))^2$$

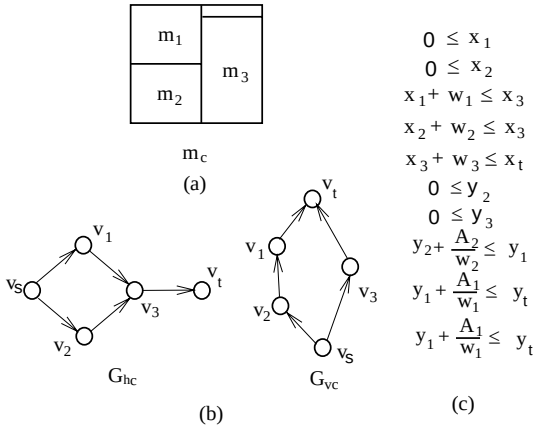

Figure 5: (a) A cluster module m_c with the cluster scheme $\{\{m_1, m_2\}, m_3\}$. (b) m_c's corresponding constraint subgraphs G_{hc} and G_{vc}. (c) The constraints to ensure that no relation of modules is violated.

$$+((y_p^i + h_p^i/2) - (y_q^i + h_q^i/2))^2). \quad (4)$$

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

$$\phi'(x,y) = \alpha x^i_{n_i+1} y^i_{n_i+1} + \beta \sum_{e^i_j \in E^i} \varpi^i_j, \quad (5)$$

where α and β are nonnegative user-defined parameters, and ϖ^i_j denotes the estimated wirelength of a net e^i_j. 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. $x^i_j + w^i_j \le x^i_{n_i+1}$ and $y^i_j + h^i_j \le y^i_{n_i+1}$). Therefore, we can formulate the problem of clustering for soft modules, called *CS*, as follows:

$$\begin{aligned}
\text{Minimize} \quad & \alpha x^i_{n_i+1} y^i_{n_i+1} + \beta \textstyle\sum_{e^i_j \in E^i} \varpi^i_j \\
\text{subject to} \quad & x^i_j + w^i_j \le x^i_{n_i+1},\ y^i_j + h^i_j \le y^i_{n_i+1},\ \forall 1 \le j \le n_i, \\
& x_p + w_p \le x_q,\ \forall e(p,q) \in G_{hj}\ \forall 1 \le j \le n_i, \\
& y_p + \tfrac{A_p}{w_p} \le y_q,\ \forall e(p,q) \in G_{vj}\ \forall 1 \le j \le n_i, \\
& x_{t_j} \le w^i_j,\ y_{t_j} \le h^i_j,\ \forall 1 \le j \le n_i \\
& L_i \le w_i \le U_i,\ \forall 1 \le i \le n,
\end{aligned}$$

where α and β are nonnegative user-defined parameters.

5.2 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:

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

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 Theorem 6.2.4 of [3]. 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:

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

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

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

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

and

$$\partial\zeta/\partial x^i_{n_i+1} = \alpha y^i_{n_i+1} - \textstyle\sum_{j=1}^{n_i} \kappa_j = 0,$$
$$\partial\zeta/\partial y^i_{n_i+1} = \alpha x^i_{n_i+1} - \textstyle\sum_{j=1}^{n_i} \eta_j = 0.$$

Thus, we have $y^i_{n_i+1} = \frac{1}{\alpha} \sum_{j=1}^{n_i} \kappa_j$, and $x^i_{n_i+1} = \frac{1}{\alpha} \sum_{j=1}^{n_i} \eta_j$.

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

Let Ω denote the set of multipliers $\vec{\mathcal{P}}$ satisfying Equations (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.

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

Thus, we have

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

where $out_G(v) = \{u | e(v,u) \in E(G)\}$. 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 $1 \le j \le n_i$,

$$\partial\zeta/\partial x^i_j = \beta \left(\sum_{e^i_k \supset \{m^i_j\}} 2\left(|e^i_k| - 1\right) x^i_j - \sum_{e^i_k \supset \{m^i_j\}} \sum_{m^i_l \in e^i_k \setminus \{m^i_j\}} x^i_l \right.$$

$$+ \sum_{e_k^i \supset \{m_j^i\}} \sum_{m_l^i \in e_k^i \setminus \{m_j^i\}} \left(w_j^i - w_l^i \right) \Bigg) = 0 \quad (8)$$

$$\partial\zeta/\partial y_j^i = \beta \left(\sum_{e_k^i \supset \{m_j^i\}} 2\left(|e_k^i| - 1\right) y_j^i - \sum_{e_k^i \supset \{m_j^i\}} \sum_{m_l^i \in e_k^i \setminus \{m_j^i\}} y_l^i \right.$$

$$\left. + \sum_{e_k^i \supset \{m_j^i\}} \sum_{m_l^i \in e_k^i \setminus \{m_j^i\}} \left(h_j^i - h_l^i \right) \right) = 0, \quad (9)$$

where $|e_k^i|$ denotes the number of the pins of e_k^i.

In Equation (8), there are n_i equations with n_i variables. Thus, we can apply the Gaussian elimination to solve these n_i equations with n_i variables to get the optimal value of x_j^i. In these n_i equations, all coefficients of variables depend only on the net information (i.e., e_k^i). 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_j^i.

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:

$$\begin{aligned}
\kappa_j' &= [\kappa_j + \rho_k(x_j^i + w_j^i - x_{n_i+1}^i)]^+ \\
\eta_j' &= [\eta_j + \rho_k(y_j^i + h_j^i - y_{n_i+1}^i)]^+ \\
\lambda_{jpq}' &= [\lambda_{jpq} + \rho_k(x_p + w_p - x_q)]^+ \\
\mu_{jpq}' &= [\mu_{jpq} + \rho_k(y_p + \frac{A_p}{w_p} - y_q)]^+,
\end{aligned}$$

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.

5.4 Projecting Lagrange Multipiers

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_{hc}) and N_{vc} (for G_{vc}) as follows:

- For each $v_i \in V(G_{hc})$ $(V(G_{vc}))$, create a vertex v_i' in N_{hc} (N_{vc}), and make v_s' and v_t' as the source and sink, respectively.
- For each $e(p,q) \in E(G_{hc})$ $(E(G_{vc}))$, create a corresponding edge $e(p',q')$ with capacity λ_{cpq} (μ_{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,* $\vec{\mathcal{P}} \in \Omega$.

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. Figure 6 gives 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 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') ∈ E(N);
2     if flow(p', q') > cap(p', q') then
3       f_over ← 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) ∈ p} > 0.
6         f_reduced ← min{min{flow(u, v)|e(u, v) ∈ p}, f_over};
7         for each edge e(u, v) ∈ p
8           flow(u, v) ← flow(u, v) − f_reduced;
9         f_over ← f_over − f_reduced;
10  compute maximum flow on N;
```

Figure 6: The increamental update algorithm.

6 Experimental Results

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

Columns 1, 2, 3, and 4 of Table 1 lists the names of the benchmark circuits, the number of modules, the number of nets, and the total area of modules in the circuits, respectively. ami49 is the largest MCNC benchmark circuit used in the previous works [5, 9] for comparative study. To test the capability of existing methods, we created ten synthetic circuits, named ex_ami49_x, by duplicating the modules and nets of ami49 by x times. The largest circuit ex_ami49_200 contains 9,800 modules and 81,600 nets.

Table 1 also shows the results for ex_ami49_x by optimizing area alone ($\gamma = 1.0$ and $\delta = 0.0$). Columns 5, 6, and 7 give the resulting area, the dead space, and the runtime for our MB*-tree, respectively. The remaining columns list the results for the famous previous works, sequence pair [18], O-tree [9], and B*-tree [5]. Note that the B*-tree package we used here is the September 2000 version, *B*-tree-v1.0*, available also at http://cc.ee.ntu.edu.tw/∼ywchang/research.html. It runs 50X–100X faster and achieves better area utilization than the B*-tree package reported in [5]. As shown in the table, our MB*-tree algorithm obtained a dead space of only 2.78% for ami49 in only 0.4 min runtime while B*-tree-v1.0 reported a dead space of 3.53% using 0.25 min runtime. Further, the experimental results for larger circuits show that the MB*-tree scales very well as the circuit size increases while the previous works, sequence pair, O-tree, and B*-tree, do not. For circuit sizes ranging from 49 to 9,800 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 sequence pair, O-tree, and B*-tree can handle only up to 196, 98, and 1,960 modules in the same amount of time and result in dead spaces of as large as 13.00% (@ 196 modules), 12.29% (@ 98 modules), and 27.33% (@ 1960 modules), respectively. As shown in Table 1, the resulting dead spaces for the MB*-tree is almost independent of the circuit sizes, which proves the high scalability of the MB*-tree. In contrast, the dead spaces for the non-hierarchical previous works all grow dramatically as the circuit size increases. In particular, the empirical runtime of the MB*-tree approaches linear in the circuit size while the other previous works cannot handle large-scale designs. Figure 7 shows the layout for the largest circuit ex_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 [5, 9, 18].

Table 2 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 sequence pair (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 9,777 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 sequence pair and B*-tree in all tests. For area optimization, MB*-tree can obtain a dead space of only 2.11% while sequence pair (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 56631 mm while sequence pair (B*-tree) requires a total wirelength of at least 81344 mm (113216 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 multulevel process incurred some overhead. Nevertheless, as shown in Table 1, both SP and B*-tree do not scale well to the 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

Circuit	# mod.	# nets	Total area (mm^2)	MB*-tree			Sequence Pair [18]			O-tree [9]			B*-tree		
				Area (mm^2)	Dead space (%)	Time (min)	Area (mm^2)	Dead space (%)	Time (min)	Area (mm^2)	Dead space (%)	Time (min)	Area (mm^2)	Dead space (%)	Time (min)
ami49	49	408	35.445	36.46	2.78	0.4	38.89	8.87	6.9	36.77	3.61	10.5	36.74	3.53	0.3
ex_ami49_2	98	816	70.890	72.72	2.51	2.5	80.27	11.69	45.5	80.82	12.29	70.6	73.11	3.03	1.3
ex_ami49_4	196	1632	141.78	145.73	2.70	2.6	162.97	13.00	309.0	155.76	9.86	179.3	151.31	6.60	4.7
ex_ami49_10	490	4080	354.45	364.14	2.66	5.4	NR	NR	NR	NR	NR	NR	407.32	12.98	19.3
ex_ami49_20	980	9160	708.90	730.95	3.02	15.6	NR	NR	NR	NR	NR	NR	870.45	18.55	23.5
ex_ami49_40	1960	16320	1417.8	1472.62	3.72	24.8	NR	NR	NR	NR	NR	NR	1951.04	27.33	53.2
ex_ami49_60	2940	24480	2126.7	2205.86	3.58	42.2	NR	NR	NR	NR	NR	NR	NR	NR	NR
ex_ami49_80	3920	32640	2835.6	2943.72	3.67	57.0	NR	NR	NR	NR	NR	NR	NR	NR	NR
ex_ami49_100	4900	40800	3544.5	3671.42	3.45	51.6	NR	NR	NR	NR	NR	NR	NR	NR	NR
ex_ami49_150	7350	61200	5316.8	5505.34	3.42	142.2	NR	NR	NR	NR	NR	NR	NR	NR	NR
ex_ami49_200	9800	81600	7089.0	7341.91	3.44	256.2	NR	NR	NR	NR	NR	NR	NR	NR	NR

Table 1: Comparisons for area, dead space, and runtime among MB*-tree, Sequence pair, O-tree, and B*-tree. **NR:** No result obtained within 5-hr CPU time on SUN Sparc Ultra 60.

Package	Area optimization ($\gamma = 1.0, \delta = 0.0$)			Wirelength optimization ($\gamma = 0.0, \delta = 1.0$)		Simultaneous area and wirelength optimization ($\gamma = 0.5, \delta = 0.5$)			
	Area (mm^2)	Dead space (%)	Time (min)	Wirelength (mm)	Time (min)	Area (mm^2)	Dead space (%)	Wirelength (mm)	Time (min)
SP	914.5/988.0	28.1/33.2	46.8/35.31	113216/122609	31.3/37.3	1104.2/1182.5	40.4/44.3	136001/159340	46.4/37.3
B*-tree	755.7/876.6	12.9/24.7	0.2/0.1	81344/91169	12.6/9.1	834.7/982.8	21.2/32.5	104558/112200	14.4/9.11
MB*-tree	671.9/740.8	2.1/3.1	11.5/6.6	56631/61698	346.1/204.7	716.3/740.8	8.1/11.1	67786/67541	221.3/220.2
SP : MB*-tree	1.36/1.34	13.38/10.71	4.07/5.35	2.00/1.98	0.09/0.18	1.54/1.60	4.99/3.99	2.01/2.36	0.21/0.17
B*-tree : MB*-tree	1.13/1.18	6.14/7.97	0.02/0.02	1.44/1.48	0.04/0.05	1.17/1.33	2.62/2.93	1.54/1.66	0.07/0.04

Table 2: Comparisons for area optimization alone, wirelength optimization alone, and simultaneous area and wirelength optimization among sequence pair (SP), B*-tree, and MB*-tree based on the circuit industry. In each entry, both the **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).

Figure 8.

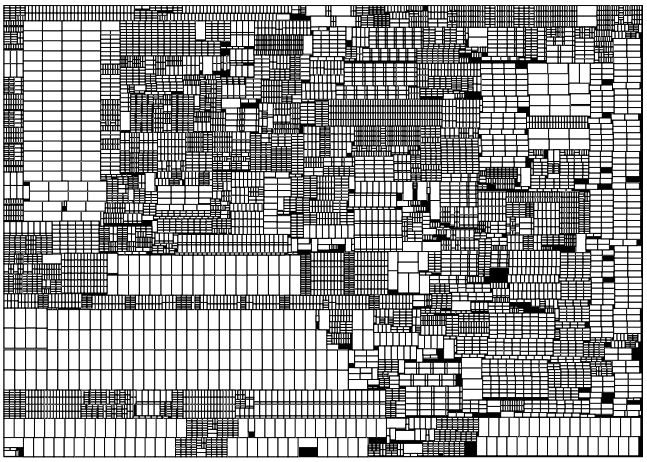

Figure 7: ex_ami49_200 layout (9,800 modules). Dead space = 3.44%.

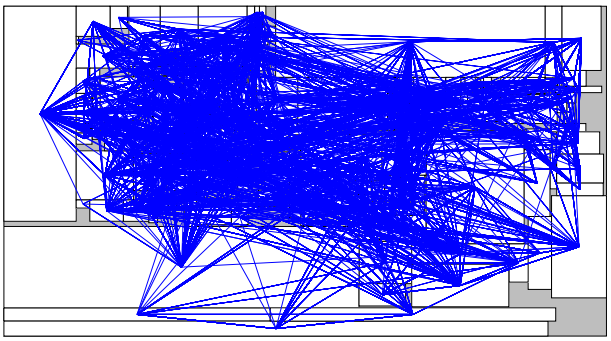

Figure 8: The layout of industry by simultaneously optimizing area and wirelength ($\gamma = 0.5, \delta = 0.5$). CPU time = 5234 sec, Area = 716263680 μm^2, Total wirelength = 67786296 μm, Dead space = 8.14%.

7 Concluding Remarks

We have presented the MB*-tree based 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 its promise in handling large-scale designs with complex constraints. We propose to explore the 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] C. J. Alpert, J. H. Hu, S. S. Sapatnekar, and P. G. Villarrubia, "A practical methodology for early buffer and wire resource allocation," *Proc. of DAC*, pp. 189–194, 2001.

[2] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," *IEEE Trans. CAD*, vol. 17, no. 8, pp. 655–667, August 1998.

[3] M. S. Bazaraa, H. D. Sherali, C. M. Shetty, *Nonlinear Programming Theory and Algorithms*, 1993.

[4] T. F. Chan, J. Cong, T. Kong, J. R. Shinnerl, "Multilevel optimization for large-scale circuit placement," *Proc. of ICCAD*, pp. 171–176, 2000.

[5] Y.-C. Chang, Y.-W. Chang, G.-M. Wu and S.-W. Wu, "B*-Trees: A new representation for non-slicing floorplans," *Proc. of DAC*, pp. 458–463, 2000.

[6] J. Cong, J. Fang, and Y. Zhang, "Multilevel approach to full-chip gridless routing," *Proc. of ICCAD*, pp. 396-403, 2001.

[7] J. Cong, T. Kong and D. Z. Pan, "Buffer Block Planning for Interconnect-Driven Floorplanning," *Proc. of ICCAD*, pp. 358–363, 1999.

[8] J. Cong, M. Xie, and Y. Zhang, "An enhanced multilevel routing system," *Proc. of ICCAD*, pp. 51–58, 2002.

[9] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," *Proc. of DAC*, pp. 268–273, 1999.

[10] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graph," *Proc. of Supercomputing*, 1995.

[11] H.-R. Jiang, Y.-W. Chang, J.-Y. Jou, and K.-Y. Chao, "Simultaneous floorplanning and buffer block planning," *Proc. of ASP-DAC*, pp. 431–434, 2003.

[12] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," *Proc. of DAC*, pp. 343–348, 1999.

[13] S.-M. Li, Y.-H. Cherng, and Y.-W. Chang, "Noise-aware buffer planning for interconnect-driven floorplanning," *Proc. of ASP-DAC*, pp. 423–426, 2003.

[14] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closuer graph based representation for non-slicing floorplans," *Proc. of DAC*, pp. 764–769, 2001.

[15] J.-M. Lin and Y.-W. Chang, "TCG-S: Orthogonal coupling of P*-admissible representations for general floorplans," *Proc. of DAC*, pp. 842–847, 2002.

[16] 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. VLSI Systems*, 2003.

[17] S.-P. Lin and Y.-W. Chang, "A novel framework for multilevel routing considering routability and performance," *Proc. of ICCAD*, pp. 44–50, 2002.

[18] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing based module placement," *Proc. of ICCAD*, pp. 472–479, 1995.

[19] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," *Proc. of ICCAD*, pp. 484–491, 1996.

[20] R. H. J. M. Otten, "Automatic floorplan design," *Proc. of DAC*, pp. 261–267, 1982.

[21] P. Sarkar, V. Sundararaman and C.K. Koh, "Routability-driven repeater block planning for interconnect-centric floorplanning," *Proc. of ISPD*, 2000.

[22] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," *Proc. of DAC*, pp. 101–107, 1986.

[23] F.Y. Young, C. C.N. Chu, W.S. Luk, and Y.C. Wong, "Floorplan area minimization using Lagrangian relaxation," *Proc. of ISPD.*, pp. 174–179, 2000.