An Effective Floorplan-Guided Placement Algorithm for Large-Scale Mixed-Size Designs

JACKEY Z. YAN, Cadence Design Systems
NATARAJAN VISWANATHAN, IBM
CHRIS CHU, Iowa State University

In this article we propose an effective algorithm flow to handle modern large-scale mixed-size placement, both with and without geometry constraints. The basic idea is to use floorplanning to guide the placement of objects at the global level. The flow consists of four steps: (1) The objects in the original netlist are clustered into blocks; (2) floorplanning is performed on the blocks; (3) the blocks are shifted within the chip region to further optimize the wirelength; (4) with large macro-locations fixed, incremental placement is applied to place the remaining objects. There are several advantages to handling placement at the global level with a floorplanning technique. First, the problem size can be significantly reduced. Second, exact Half-Perimeter WireLength (HPWL) can be minimized. Third, better object distribution can be achieved so that legalization only needs to handle minor overlaps among small objects in a block. Fourth, macro-rotation and various geometry constraints can be handled. To demonstrate the effectiveness of this new flow, we implement a high-quality and efficient floorplan-guided placer called *FLOP*. We also construct the Modern Mixed-Size (MMS) placement benchmarks that can effectively represent the complexities of modern mixed-size designs and the challenges faced by modern mixed-size placers. Compared with most state-of-the-art mixed-size placers and leading macroplacers, experimental results show that FLOP achieves the best HPWL and easily obtains legal solutions on all circuits with all geometry constraints satisfied.

Categories and Subject Descriptors: B.7.2 [**Hardware, Integrated Circuits, Design Aids**]: Placement and Routing

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Floorplanning, placement, geometry constraint, mixed-size design

ACM Reference Format:
Jackey Z. Yan, Natarajan Viswanathan, and Chris Chu. 2014. An effective floorplan-guided placement algorithm for large-scale mixed-size designs. ACM Trans. Des. Autom. Electron. Syst. 19, 3, Article 29 (June 2014), 25 pages.
DOI: http://dx.doi.org/10.1145/2611761

1. INTRODUCTION

In the nanometer-scale era, placement has become an extremely challenging stage in modern Very-Large-Scale Integration (VLSI) designs. Millions of objects need to be placed legally within a chip region, while both the interconnection and object distribution have to be optimized simultaneously. As an early step of VLSI physical design flow, placement significantly impacts on both routing and manufacturing. In modern System-on-Chip (SoC) designs, the usage of Intellectual Property (IP) and embedded memory blocks becomes more and more popular. As a result, a design usually

This work was partially supported by IBM Faculty Award and NSF under grant CCF-0540998.
Authors' addresses: J.-Z. Yan (corresponding author), Placement Technology Group, Cadence Design Systems, San Jose, CA 95134; email: jackey.yan@gmail.com; N. Viswanathan, Systems and Technology Group, IBM, Austin, TX 78758; C. Chu, Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010.

DOI: http://dx.doi.org/10.1145/2611761

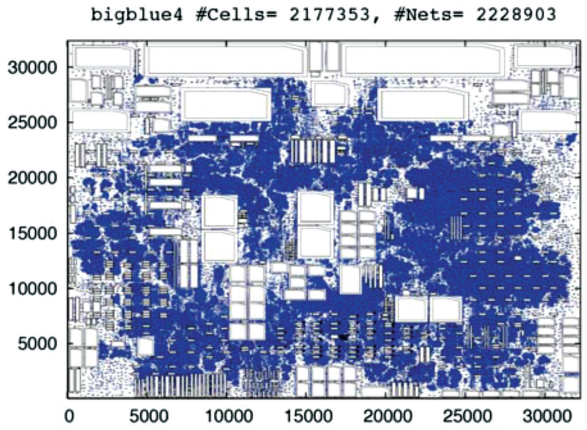

Fig. 1. Example of modern mixed-size circuit which contains 2177353 objects and 2228903 nets. The blue dots represent standard cells, and the white rectangular regions represent macros.

contains tens or even hundreds of large macroblocks (i.e., large macros). A design with large movable macros and numerous standard cells is known as mixed-size design. An example of modern mixed-size design is shown in Figure 1.

For mixed-size designs, the placement of large macros plays a key role. Due to the big size difference between large macros and standard cells, the placement of mixed-size designs is much more difficult than the standard-cell placement. Existing placement algorithms perform very poorly on mixed-size designs. They usually cannot generate a legal solution by themselves, and have to rely on a postplacement legalization process. However, legalizing large macros with wirelength minimization has been considered very hard to solve for a long time. Moreover, sometimes the large macros have various placement geometry constraints, such as preplaced, boundary, distance constraints. This makes the problem of mixed-size placement even harder. As existing placement algorithms simply cannot handle such geometry constraints, the designer has to place these macros manually beforehand.

1.1. Previous Work

Most mixed-size placement algorithms place both the macros and standard cells simultaneously. Examples are the annealing-based placer Dragon [Taghavi et al. 2006], the partitioning-based placer Capo [Roy et al. 2006], and the analytical placers FastPlace3 [Viswanathan et al. 2007], APlace2 [Kahng and Wang 2006], Kraftwerk [Spindler and Johannes 2006], mPL6 [Chan et al. 2006], and NTUplace3 [Chen et al. 2006]. The analytical placers are the state-of-the-art placement algorithms. They can produce the best result in the best runtime. But, the analytical approach has three problems. First, only an approximation (e.g., by log-sum-exp or quadratic function) of the HPWL is minimized. Second, the distribution of objects is also approximated and this usually results in a large amount of overlaps. They have to rely on a legalization step to resolve the overlaps. For mixed-size designs, such a legalization process is very difficult and is likely to significantly increase the wirelength, because a small movement of one big macro can potentially affect the locations of thousands of small standard cells. Third, traditional analytical placers cannot optimize macro-orientations and handle geometry constraints. Most recently, Hsu and Chang [2010] improved the traditional analytical placement algorithm by introducing a rotation force to optimize the macro-orientations. But their algorithm still cannot handle geometry constraints.

Fig. 2. Previous two-stage apnproach.

Other researchers apply a two-stage approach as shown in Figure 2 to handle the mixed-size placement. An initial wirelength-driven placement is first generated. Then a macroplacement or legalization algorithm is used to place only the macros, without considering the standard cells. After this, the macros are fixed and the standard cells are replaced in the remaining whitespace from scratch. As the macroplacement is a crucial stage in this flow, people propose different techniques to improve the quality of result (QoR). Based on the multi-packing tree (MPT) representation, Chen et al. [2007] used a packing-based algorithm to place the macros around the four corners of the chip region. In Chen et al. [2008], a transitive closure graph (TCG) based technique was applied to enhance the quality of macroplacement. In Cong and Xie [2006], the authors applied a combination of constraint graph and linear programming approaches to obtain a legalized placement on the macros. One main problem with the preceding three approaches is that the initial placement is produced with a large amount of overlaps. Thus, the initial solution may not provide good indications on the locations of objects. However, the following macroplacement stage still determines the macro-locations by minimizing the displacement from the low-quality initial placement.

Alternatively, after the initial placement is generated by a standard-cell placer, Adya and Markov [2005] used an annealing-based fixed-outline floorplanner to remove the overlap between the macros and clustered standard cells at the macroplacement stage. But, they still have to rely on the illegal placement to determine the initial locations of macros and clusters. For all of the previous two-stage approaches, after fixing the macros, the initial positions of standard cells have to be discarded to reduce the overlaps.

1.2. Our Contributions

In this work, an efficient and high-quality placement tool is presented to effectively handle the complexities of modern large-scale mixed-size placement. Such a tool is developed based on a new placement flow that integrates floorplanning and incremental placement algorithms. The main idea of this flow is to use the fixed-outline floorplanner to guide the state-of-the-art analytical placer. As floorplanners have a good capability of handling a small number of objects [Roy et al. 2006, 2009] and various geometry constraints [Young et al. 2004], we apply a floorplanning algorithm on the clustered circuit to generate a global overlap-free layout, and use it to guide the subsequent placement algorithm.

This proposed new algorithm flow for mixed-size placement is as follows (see Figure 3).

(1) *Block Formation*. The first step is to cut down the problem size. We define "small objects" as small macros and standard cells. The small objects are clustered into soft blocks, while each large macro is treated as a single hard block.
(2) *Floorplanning*. In this step, a floorplanner is applied on the blocks to directly minimize the *exact* HPWL at floorplan level. Simultaneously, the objects are better distributed across the chip region to guarantee an overlap-free layout.
(3) *Wirelength-Driven Shifting*. In order to further optimize the HPWL, the blocks are shifted at the floorplan level. After shifting, large macros are fixed. The remaining movable objects are assumed to be at the center of the corresponding soft block.

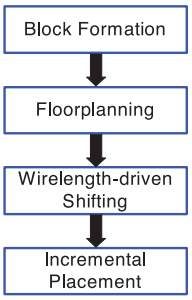

Fig. 3. New algorithm flow for mixed-size placement.

(4) *Incremental Placement*. Lastly, the placement algorithm will place the remaining objects. The initial positions of such objects provided by the previous step are used to guide the incremental placement.

Generally, there are several advantages to handling mixed-size placement at global level with a floorplanning technique. First, at the floorplanning stage the problem size has been significantly reduced, so that the algorithm performs more efficiently and effectively. Second, the exact HPWL among the blocks can be minimized at floorplan level. Third, better object distribution can be achieved at the floorplanning stage, so that the legalization in the placement stage only needs to handle minor overlaps among small objects. Last but not least, macro-rotation and various geometry constraints on the macros can be handled at floorplanning stage. Comparing this new methodology with the state-of-the-art analytical placers, we can see that it is superior in several aspects: (1) The exact HPWL is optimized in steps 1–3; (2) the objects are better distributed in step 2; (3) geometry constraints and macro-orientation optimization can be handled in step 2. Compared with the previous two-stage approaches, instead of starting from an illegal initial placement, we use the floorplanner to directly generate a global overlap-free layout among the large macros, as well as between large macros and small objects. In addition, the problem size has been significantly reduced by clustering. A good floorplanner should be able to produce a high-quality global layout for the subsequent incremental placer. Furthermore, the initial positions of the small objects are not discarded. We keep such information as a starting point of incremental placement. Since the large macros have already been fixed, the placer avoids the difficulty of legalizing the large macros.

Based on the new algorithm flow, we implement a robust, efficient, and high-quality floorplan-guided placement tool called *FLOP*[1]. FLOP has two operation modes. In the default mode, FLOP optimizes mixed-size placement with all movable objects (including both macros and standard cells) without geometry constraints. FLOP also determines the macro-orientation with respect to packing and wirelength optimization. In order to handle the design with geometry constraints, we developed the *FLOP-C* mode. To implement such a mode, we proposed several new techniques, of which the main focus is to enable an enhanced annealing-based floorplanning framework to handle various geometry constraints.

To show the effectiveness of FLOP, based on ISPD05/06 placement benchmarks we derive the Modern Mixed-Size (MMS) placement benchmarks and MMS-C benchmarks (a version of MMS benchmarks with geometry constraints). These new circuits can represent the challenges of modern large-scale mixed-size placement.

[1]A preliminary version of FLOP was presented in Yan et al. [2009a].

The rest of this article is organized as follows. Section 2 describes the overview of FLOP. Section 3 presents the block formation step in FLOP. Section 4 introduces the floorplanning algorithm applied in the default mode. Section 5 describes the annealing-based floorplanner in the FLOP-C mode. Section 6 presents the wirelength-driven shifting technique. Section 7 describes the incremental placement algorithm. Section 8 describes the construction of MMS and MMS-C benchmarks. Section 9 presents the experimental results. Finally, this article ends with the conclusion.

2. OVERVIEW OF FLOP

FLOP follows the algorithm flow in Figure 3. It has two operation modes: the default and FLOP-C modes. The only difference between the two modes is in the floorplanning step.

In FLOP, the block formation is done by a combination of the clustering and recursive partitioning. We first apply a high-quality clustering algorithm *SafeChoice* [Yan et al. 2011] to initially generate some small clusters, and then we perform partitioning using *hMetis2.0* [Karypis and Kumar 1999] on the clustered netlist to further cut down the problem size. After partitioning, small objects in each partition are grouped into a soft block and each large macro becomes a single hard block.

In the default mode, the subsequent floorplanning step adopts a min-cut-based fixed-outline floorplanner *DeFer* [Yan and Chu 2010]. In *DeFer*, a hierarchy of the blocks needs to be derived using recursive partitioning. Because such a hierarchy has already been generated during the block formation step, it will be passed down and will not be generated again. Another way to look at this is that the block formation step in the default mode is merged into the floorplanning step as the first stage of *DeFer*.

Different from the floorplanning algorithm used in the default mode, an enhanced annealing-based floorplanner is proposed to handle the geometry constraints in FLOP-C mode. Due to the inherent slowness of annealing, we are not completely relying on the annealing process to produce a high-quality floorplan. Instead, we first apply *DeFer* to generate a floorplan containing only the blocks without geometry constraints, namely, nonconstraint blocks. Second, we physically insert the constraint blocks into this floorplan and obtain an initial overlap-free floorplan containing all blocks in the design. After such insertion, the constraint blocks are close to their locations specified in the geometry constraints, and most nonconstraint blocks also maintain their previous locations generated by *DeFer*. Thus, this initial floorplan gives the annealing-based floorplanner a good start point. At the end, we apply the annealing process to further optimize the complete block-level netlist with geometry constraint awareness, and output a final legal floorplan.

In FLOP, the problem of wirelength-driven shifting is formulated as a min-cost flow problem. So, we can efficiently find the *optimal* block position in terms of the HPWL minimization among the blocks.

Because analytical placers have the best capability in placing a large number of small objects, we use an analytical placer as the engine in the incremental placement step.

Comparing FLOP with the previous work in Adya and Markov [2005], both algorithms use floorplanning-, clustering-, and annealing-based methods for mixed-size placement. But besides the flow-wise differences as mentioned previously, they are different in the following aspects: (1) The floorplanner in FLOP does not rely on any initial placement generated by the standard-cell placer. (2) The clustering method in FLOP is used to generate the initial small clusters, and the final soft blocks (i.e., clustered small objects) are generated by partitioning. (3) FLOP uses an annealing-based floorplanner to handle various geometry constraints. For the design without such constraints, FLOP uses the nonstochastic floorplanner *DeFer*.

3. BLOCK FORMATION

A high-quality hypergraph clustering algorithm called *SafeChoice* was proposed in Yan et al. [2011]. *SafeChoice* maintains a global priority queue based on the safeness and area of potential clusters. Iteratively, a new cluster at the top of the queue is formed. The objective of *SafeChoice* is to cut down the problem size by clustering small objects without loss of placement quality. It has been shown that, compared with other clustering algorithms, *SafeChoice* generates the best clusters for wirelength-driven placement.

In the block formation step of FLOP, we first apply *SafeChoice* on the original netlist to generate a clustered netlist. The clusters generated by *SafeChoice* are usually small in size. As the cluster becomes bigger, the quality of each cluster degrades. In order to further reduce the problem size, we perform recursive bipartitioning on the clustered netlist. At the end, a block-level netlist is generated for the subsequent floorplanning step. Note that the stand-alone clustering algorithm applied before partitioning not only cuts down the runtime of partitioning, but also improves its QoR.

As mentioned earlier, the main purpose of the block formation step is to cut down the problem size. For a typical placement problem with millions of objects in the original netlist, we need to cut down to thousands of blocks for the block-level netlist. So we propose a stopping criterion for the partitioning process. Let A_o be the total area of all objects in the design. In one partition there are N_p objects of which the total area is A_p, and α is the area bound ($\alpha = 0.15\%$ by default). We will stop cutting this partition if either one of the following conditions is satisfied: (1) $\frac{A_p}{A_o} \leq \alpha$; (2) $N_p \leq 10$.

At the end of the partitioning process, inside each partition a large macro is treated as a hard block and all small objects are grouped into a soft block.

4. FLOORPLANNING ALGORITHM IN DEFAULT MODE

A high-quality and nonstochastic fixed-outline floorplanner *DeFer* was presented in Yan and Chu [2010]. It has been shown that, compared with other fixed-outline floorplanners, *DeFer* achieves the best success rate, the best wirelength, and the best runtime on average.

Here is a brief description of the algorithm flow of *DeFer*: First the original circuit is partitioned into several subcircuits. After that, a high-level slicing tree structure is built up. Second, for each subcircuit an associated shape curve is generated to represent all possible slicing layouts within the subcircuit. Third, the shape curves are combined from bottom-up following the high-level slicing tree. In the final shape curve at the root, the points within the fixed outline are chosen for further HPWL optimization. At the end *DeFer* outputs a final layout.

In the default mode of FLOP, we use *DeFer* in the floorplanning step. To make it more robust and efficient for mixed-size placement, we propose some new techniques and strategies that are described in the following two sections.

4.1. Usage of Exact Net Model

We use the exact net model in Chen et al. [2005] to improve the HPWL in partitioning. By applying this net model in partitioning, the cut value becomes exactly the same as the placed HPWL, so that the partitioner can directly minimize the HPWL instead of interconnections between two partitions. Note that, in the exact net model, the HPWL is calculated based on the assumption that the objects inside each subpartition are located at the corresponding center of that subpartition. In FLOP at the first β levels of the high-level slicing tree, we apply two cuts on the original partition. One is a horizontal cut, and another is a vertical cut. We compare these two cuts and pick the one with less cost, namely, HPWL.

However, for a vertical/horizontal cut, the cut value returned by the net model is only equal the horizontal/vertical component of HPWL. So for two cuts with different directions, it is incorrect to decide a better cut direction based on the two cut values generated by these two cuts. The authors in Chen et al. [2005] avoided such comparison by fixing the cut direction based on the dimension of the partition region. Nevertheless, this may potentially lose the better cut direction. Here we propose a simple heuristic to solve the cut value comparison between the cuts from two different directions.

Suppose K is the total number of nets in one partition that we are going to cut. For the horizontal cut (H-cut), $L^x_{H_i}$ and $L^y_{H_i}$ are the horizontal and vertical components of the HPWL of net i, respectively. Similarly, $L^x_{V_i}$ and $L^y_{V_i}$ are the horizontal and vertical components of the HPWL of net i for the vertical cut (V-cut), respectively. So the total HPWL of the K nets in this partition are as follows.

$$\text{for H-cut}: \ L_H = \sum_{i=1}^{K} L^x_{H_i} + \sum_{i=1}^{K} L^y_{H_i}$$
$$\text{for V-cut}: \ L_V = \sum_{i=1}^{K} L^x_{V_i} + \sum_{i=1}^{K} L^y_{V_i}$$

Thus, the correct way to make the comparison between H-cut and V-cut should be

$$\text{if } L_H \geq L_V \Rightarrow \text{V-cut is better}$$
$$\text{if } L_H < L_V \Rightarrow \text{H-cut is better.}$$

As the net model only returns $\sum_{i=1}^{K} L^y_{H_i}$ for the H-cut and $\sum_{i=1}^{K} L^x_{V_i}$ for V-cut, we need find a way to estimate $\sum_{i=1}^{K} L^x_{H_i}$ and $\sum_{i=1}^{K} L^y_{V_i}$. Let the aspect ratio[2] of the partition region be γ. When K is very big, based on statistics we can have

$$\frac{\sum_{i=1}^{K} L^y_{H_i}}{\sum_{i=1}^{K} L^x_{H_i}} \approx \frac{\sum_{i=1}^{K} L^y_{V_i}}{\sum_{i=1}^{K} L^x_{V_i}} \approx \gamma.$$

Thus,

$$\text{if } L^y_H \geq L^x_V \cdot \gamma \Rightarrow \text{V-cut is better}$$
$$\text{if } L^y_H < L^x_V \cdot \gamma \Rightarrow \text{H-cut is better.}$$

Two reasons prevent us from applying the net model in lower levels (i.e., when it is more than β levels): (1) As partitioning goes on, K becomes smaller and smaller, which makes the approximation of $\sum_{i=1}^{K} L^x_{H_i}$ and $\sum_{i=1}^{K} L^y_{V_i}$ inaccurate; (2) using the net model, we restrict the combine direction in the generalized slicing tree [Yan and Chu 2010], which hurts the packing quality. To make a trade-off we only apply the net model in the first β levels ($\beta = 3$ by default). After the first β levels, we use the traditional min-cut cost, and let *DeFer* decide the best cut direction for packing. In the experiments, we found that $\beta = 3$ gives us the best result. If β is set smaller than 3, the HPWL degrades. If β is set bigger than 3, some designs with large movable macros cannot fit into the fixed outline.

[2]In this article, *aspect ratio* is defined as the ratio of height to width.

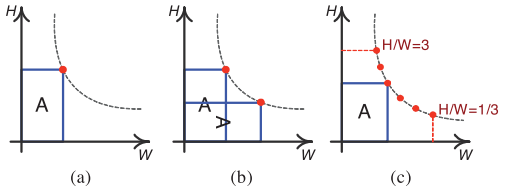

Fig. 4. Generation of shape curves for blocks.

4.2. Generation of Shape Curve for Blocks

To capture the shape of the blocks, we generate an associated shape curve for each block. For the hard block if a macro cannot be rotated, only one point representing the user-specified orientation is generated (see Figure 4(a)). Otherwise two points representing two different rotations are generated (see Figure 4(b)). For the soft block we bound its aspect ratio from 1/3 to 3, and sample multiple points on the shape curve to represent its shape (see Figure 4(c)). The details of the sampling process are described in Yan and Chu [2010]. Considering the target density constraint in the placement, we add extra whitespace in each soft block. In some sense, we "inflate" the soft block based on the target density.

$$A'_{s_i} = \frac{A_{s_i}}{TD} \times (max((TD - 0.93), 0) \times 0.5 + 1) \quad (1)$$

In Eq. (1), for soft block i, A'_{s_i} is the "inflated area", A_{s_i} is the total area of objects within soft block i, and TD is the target density. Based on this formula, if the target density is more than 93%, we add extra whitespace into the soft block. The purpose is to leave more space for the analytical placer to place the small objects.

5. FLOORPLANNING ALGORITHM IN FLOP-C MODE

This section presents the enhanced annealing-based floorplanning algorithm applied in the floorplanning step in the FLOP-C mode.

In Kahng and Wang [2005], the authors extended the analytic placer to handle geometry constraints by adding the penalty terms into the objective function. In Kim and Markov [2012], the authors applied the technique of global feasibility projection to handle various constraints. However, for both of these two approaches, the geometry constraints were only added on the standard cells. Several previous works [Young et al. 2004; Tam et al. 2006; Ma et al. 2011] have demonstrated the idea of using the annealing-based framework to handle various geometry constraints on the macros. But there are two common drawbacks in their annealing processes: (1) They all start with a random initial floorplan so it takes a long time to search for a high-quality solution; (2) at each annealing iteration, the shapes of the soft blocks are randomly determined by annealing to improve the packing. Such a shaping strategy has no optimality guarantee. Due to these two reasons, most of the moves in annealing may not be made effectively towards the global optimum, and consequently the annealing takes a long time to converge. This degrades both the effectiveness and efficiency of the annealing-based floorplanners. As shown in Young et al. [2004], Tam et al. [2006], and Ma et al. [2011], their floorplanners were only applied on the small-scale circuits (e.g., at most 300 blocks). But for modern SoC designs the circuit usually contains more than 1000 mixed of hard and soft blocks. The complexity would even increase dramatically if some geometry constraints are involved. Obviously, completely relying on the traditional annealing process to obtain a good solution is not practical

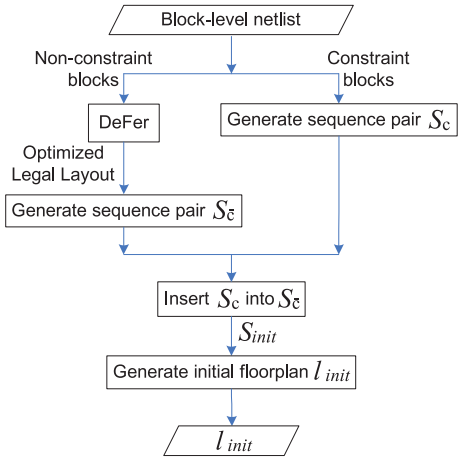

Fig. 5. Flow of generating initial floorplan l_{init}.

anymore. The focus of this section is to present two main techniques used to generate a high-quality initial floorplan as a starting point for the subsequent annealing process. Furthermore, in the FLOP-C mode we adopt an optimal and fast shaping algorithm called *SDS* [Yan and Chu 2012] to shape the soft blocks in each annealing iteration.

First of all, we introduce notation and definitions. The sequence pair [Murata et al. 1996] $S(S^+, S^-)$ is used to represent a floorplan layout in the annealing-based framework, where S^+ is the positive sequence and S^- is the negative sequence. To model the geometry relationship among the blocks in a floorplan, the horizontal and vertical constraint graphs are derived from a given sequence pair. In the constraint graph, the vertex represents the block and the edge between two vertices represents the nonoverlapping constraints between the two corresponding blocks. Using the longest path algorithm, the block locations can be calculated from the given constraint graphs.

In FLOP-C mode, we consider the following three geometry constraints.

—*Preplaced Constraint.* This constraint is imposed when some block has to be preplaced and fixed at some location in the chip region.
—*Range Constraint.* This constraint specifies that some block has to be within a certain coordinates range in either the horizontal or vertical direction. If both horizontal and vertical range constraints are imposed on the same block, then this constraint is the same as the *region constraint*.
—*Boundary Constraint.* This constraint specifies that some block has to be placed along either one of the four boundaries of the chip region.

The preceding three geometry constraints are the most commonly considered ones in modern mixed-size placement. Based on the framework proposed in Young et al. [2004], we can extend our tool to handle other constraints defined in Young et al. [2004].

Because of the slowness of the annealing process, we cannot completely rely on it to generate a high-quality floorplan from scratch, with all geometry constraints satisfied. Therefore, rather than starting from a random initial floorplan, which the traditional approach does, we start the annealing process based on an initial floorplan l_{init} generated from a high-quality sequence pair, namely, S_{init}. As shown in Figure 5, S_{init} is produced by inserting the sequence pair containing only constraint blocks (i.e., S_c)

into the sequence pair containing only nonconstraint blocks (i.e., $S_{\bar{c}}$). It is a high-quality sequence pair in the following two aspects.

—In l_{init}, the constraint blocks are placed close to the locations specified in the geometry constraints. Note that S_c is generated from a legal floorplan where the locations of the constraint blocks are set accordingly to satisfy their geometry constraints.
—The initial locations of nonconstraint blocks are generated by the high-quality floorplanner *DeFer* with both wirelength and packing awareness. In l_{init}, most nonconstraint blocks can hold their initial positions.

Compared with previous approaches, the initial floorplan l_{init} gives the annealing process a much better solution to start with. This significantly improves both the efficiency and quality of the annealing-based floorplanner. To obtain S_{init}, we propose the following two key techniques.

(1) A simple sequence pair generation algorithm produces a sequence pair from a given legal layout. In Figure 5 this algorithm is applied to generate S_c and $S_{\bar{c}}$.
(2) A location-aware sequence pair insertion algorithm inserts one sequence pair into another while maintaining the block physical locations as much as possible. In Figure 5 this algorithm is applied to insert S_c into $S_{\bar{c}}$.

The aforesaid two techniques are described in the following two sections.

5.1. Sequence Pair Generation from Given Layout

Since the sequence pair representation was introduced in Murata et al. [1996], many works have focused on efficiently generating a legal layout from a given sequence pair. But, only a few algorithms [Murata et al. 1996; Egeblad 2003; Kodama et al. 2004] were proposed to solve the reversed problem, that is, how to generate a sequence pair from a given legal layout. Murata et al. [1996] used the gridding method via drawing the up-right and down-left step-lines to derive a sequence pair from a layout. However, this method is not intuitive and quite complicated for implementation. Egeblad [2003] extended the step-line idea, so that the sequence pair can be derived from an illegal layout. In Kodama et al. [2004], using a more sophisticated data structure, namely, Q-sequence [Sakanushi et al. 2003], the authors proposed a linear-time algorithm to generate the sequence pair from a legal layout. In this section, we propose a very simple $\mathcal{O}(n^2)$ algorithm to solve this problem.

To generate a sequence pair $S(S^+, S^-)$ from a given legal layout, we apply the following consecutive three steps.

(1) The relative orders of every two blocks in S^+ and S^- are determined based on the block locations in the layout.
(2) Two directed acyclic graphs G^+ and G^- are built based on the relative orders determined in step 1.
(3) S^+ and S^- are generated by applying topological sort on G^+ and G^-, respectively.

In step 1, we first determine the location relations between every two blocks in a layout. Then, based on the location relations, we can determine their relative orders in S^+ and S^-. For example, given any legal layout and a pair of blocks a and b, we can always divide the chip region that is not occupied by b into eight subregions: left (L), right (R), upper (U), bottom (B), upper-left (UL), upper-right (UR), bottom-left (BL), and bottom-right (BR) subregions (see Figure 6). Depending on the location of block a,

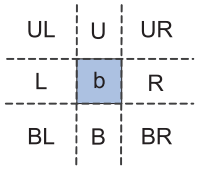

Fig. 6. Divided eight sub-regions based on block b.

there are only eight[3] possible location relations between a and b, and they can be determined as follows.

(1) a is on the *left* of b, iff a overlaps with L subregion.
(2) a is on the *right* of b, iff a overlaps with R subregion.
(3) a is *above* b, iff a overlaps with U subregion.
(4) a is *below* b, iff a overlaps with B subregion.
(5) a is on the *left* of and *above* b, iff a only overlaps with UL subregion.
(6) a is on the *right* of and *above* b, iff a only overlaps with UR subregion.
(7) a is on the *left* of and *below* b, iff a only overlaps with BL subregion.
(8) a is on the *right* of and *below* b, iff a only overlaps with BR subregion.

Once the location relation between a and b is available, based on the horizontal and vertical constraints [Murata et al. 1996] imposed by the sequence pair, the relative order between a and b in S^+ and S^- is determined as follows.

(1) If a is on the *left* of b, a is *before* b in S^+ and *before* b in S^-.
(2) If a is on the *right* of b, a is *after* b in S^+ and *after* b in S^-.
(3) If a is *above* b, a is *before* b in S^+ and *after* b in S^-.
(4) If a is *below* b, a is *after* b in S^+ and *before* b in S^-.
(5) If a is on the *left* of and *above* b, a is *before* b in S^+.
(6) If a is on the *right* of and *above* b, a is *after* b in S^-.
(7) If a is on the *left* of and *below* b, a is *before* b in S^-.
(8) If a is on the *right* of and *below* b, a is *after* b in S^+.

The preceding cases (5)–(8) are derived based on the cases (1)–(4). For example, if both cases (1) and (3) are satisfied, we know that a is definitely before b in S^+, which is case (5). But their relative order in S^- is undetermined.

In step 2, two directed acyclic graphs G^+ and G^- are built to capture the relative orders determined in step 1. For example, G^+ is built up in the following manner. Each vertex in G^+ represents a block. Given any two blocks a and b:

—if a is before b in S^+, then a direct edge from a to b is added;
—if a is after b in S^+, then a direct edge from b to a is added;
—otherwise, no edge is added between a and b.

In step 3, topological sort is applied on G^+ and G^-. After sorting, the resulting two sequences from G^+ and G^- are S^+ and S^-, respectively.

Note that, because the set of all sequence pairs is P-admissible [Murata et al. 1996], there could be multiple sequence pairs mapped to the same layout. Our proposed algorithm will only generate one sequence pair solution. The whole process of sequence pair generation takes $\mathcal{O}(n^2)$ time, as we traverse every pair of blocks in the design. If a more sophisticated data structure is used, the algorithm complexity can be improved

[3]Since the given layout is legal, a and b would not overlap with each other.

Begin
1. Initialize L_i^+ as the head of $S_{\bar{c}}^+$, $1 \leq i \leq n$,
2. $k = n$.
3. **For** $(j = m; j >= 1; j = j - 1)$
4. $b = S_{\bar{c}}^+[j]$,
5. Pack block b to C_{br}.
6. **For** $(i = k; i >= 1; i = i - 1)$
7. $b_c = S_c^+[i]$,
8. If b is neither on the right of nor below b_c,
9. Set $L_{b_c}^+$ after b in $S_{\bar{c}}^+$,
10. $k = k - 1$,
11. $j = j + 1$,
12. Break loop.
13. **End of loop**
14. **End of loop**

End

(a) Left-most insertion point calculation

Begin
1. Initialize R_i^+ as the rear of $S_{\bar{c}}^+$, $1 \leq i \leq n$,
2. $k = 1$.
3. **For** $(j = 1; j <= m; j = j + 1)$
4. $b = S_{\bar{c}}^+[j]$,
5. Pack block b to C_{ul}.
6. **For** $(i = k; i <= n; i = i + 1)$
7. $b_c = S_c^+[i]$,
8. If b is neither on the left of nor above b_c,
9. Set $R_{b_c}^+$ before b in $S_{\bar{c}}^+$,
10. $k = k + 1$,
11. $j = j - 1$,
12. Break loop.
13. **End of loop**
14. **End of loop**

End

(b) Right-most insertion point calculation

Fig. 7. Calculation of insertion range in $S_{\bar{c}}^+$.

further. In the algorithm flow shown in Figure 5, we apply the sequence pair generation technique when generating both S_c and $S_{\bar{c}}$.

5.2. Sequence Pair Insertion with Location Awareness

In this section, we propose a location-aware sequence pair insertion technique that inserts $S_c(S_c^+, S_c^-)$ into $S_{\bar{c}}(S_{\bar{c}}^+, S_{\bar{c}}^-)$. The resulting sequence pair after the insertion is S_{init}.

To maintain the original block locations, after insertion the relative orders of blocks in both S_c and $S_{\bar{c}}$ should not be changed. For each constraint block to be inserted, we define an *insertion range* that specifies the leftmost and rightmost possible insertion points in $S_{\bar{c}}(S_{\bar{c}}^+, S_{\bar{c}}^-)$. The main idea of the insertion technique is to use block packing to determine the insertion range for each constraint block. For example, given a constraint block i, we want to find its rightmost possible insertion point in $S_{\bar{c}}^-$. Based on the horizontal and vertical relations imposed by the sequence pair, any block before i in $S_{\bar{c}}^-$ should be either on the left of or below i in the layout. Thus, we traverse the blocks from the head to the rear in $S_{\bar{c}}^-$, and pack them one by one to the bottom-left corner of the chip region, until we find any block in $S_{\bar{c}}^-$, say block j, is neither on the left nor below i. This means block j has to be after block i in $S_{\bar{c}}^-$. Thus, we set the rightmost possible insertion point for block i as before block j in $S_{\bar{c}}^-$.

Before we present the detailed algorithm of calculating the insertion range in $S_{\bar{c}}^+$ and $S_{\bar{c}}^-$, we first introduce notation. Let C_{ul}, C_{ur}, C_{bl}, and C_{br} denote the upper-left, upper-right, bottom-left, and bottom-right corners of the chip region, respectively. Assuming there are m nonconstraint blocks and n constraint blocks, the position index in a sequence is ordered from the head to the rear. Let $S^+[i]$ denote the block at the ith position in S^+; similarly we can define $S^-[i]$. We define (L_i^+, R_i^+) as the insertion range in $S_{\bar{c}}^+$ for block i, where L_i^+ and R_i^+ denote the leftmost and rightmost possible insertion points, respectively. Similarly, the insertion range in $S_{\bar{c}}^-$ for block i is defined as (L_i^-, R_i^-).

The pseudocodes of calculating the insertion range in $S_{\bar{c}}^+$ and $S_{\bar{c}}^-$ are shown in Figures 7 and 8, respectively. The insertion range for each constraint block is first initialized as spanning from the head to the rear of $S_{\bar{c}}$. This initial range is considered to be loose, which implies that the constraint block can be inserted at any point in the sequence. As the algorithm moves forward, a tighter range will be determined based on

Begin
1. Initialize L_i^- as the head of $S_{\overline{c}}^-$, $1 \leq i \leq n$,
2. $k = n$.
3. **For** $(j = m; j >= 1; j = j - 1)$
4. $b = S_{\overline{c}}^-[j]$,
5. Pack block b to C_{ur}.
6. **For** $(i = k; i >= 1; i = i - 1)$
7. $b_c = S_c^-[i]$,
8. If b is neither on the right of nor above b_c,
9. Set $L_{b_c}^-$ after b in $S_{\overline{c}}^-$,
10. $k = k - 1$,
11. $j = j + 1$,
12. Break loop.
13. **End of loop**
14. **End of loop**

End

(a) Left-most insertion point calculation

Begin
1. Initialize R_i^- as the rear of $S_{\overline{c}}^-$, $1 \leq i \leq n$,
2. $k = 1$.
3. **For** $(j = 1; j <= m; j = j + 1)$
4. $b = S_{\overline{c}}^-[j]$,
5. Pack block b to C_{bl}.
6. **For** $(i = k; i <= n; i = i + 1)$
7. $b_c = S_c^-[i]$,
8. If b is neither on the left of nor below b_c,
9. Set $R_{b_c}^-$ before b in $S_{\overline{c}}^-$,
10. $k = k + 1$,
11. $j = j - 1$,
12. Break loop.
13. **End of loop**
14. **End of loop**

End

(b) Right-most insertion point calculation

Fig. 8. Calculation of insertion range in $S_{\overline{c}}^-$.

the packing. Finally, after the insertion range is available, we choose the middle point in the range as the final insertion position for the corresponding constraint block. The runtime complexity of the insertion technique is $\mathcal{O}(mn)$.

5.3. Geometry Constraint Handling and Annealing Schedule

We adopt a similar annealing-based method as in Young et al. [2004] to handle the three types of geometry constraints, namely, preplaced, range, and boundary constraints. Using the initial floorplan l_{init} generated in Figure 5 as a start point, we apply the annealing process to optimize both the total HPWL and packing among the blocks at the floorplan step while satisfying the geometry constraints. The following cost function is used to evaluate a floorplan in annealing:

$$C = WL + \mu \times P_o + \nu \times P_c,$$

where WL is the term of average HPWL over all nets in the block-level netlist, P_o is the penalty term if the blocks are placed outside of the chip region, and P_c is the penalty term for the unsatisfied geometry constraints. μ and ν are the weight among the three terms (by default, $\mu = 2$ and $\nu = 4$). P_o is calculated as follows:

$$P_o = max(0, (w - W)) + max(0, (h - H)),$$

where w and h are the width and height of a given floorplan, respectively, and W and H are the width and height of the chip region, respectively. If one constraint is not satisfied, we will move the corresponding constraint block to the closest location in order to satisfy the geometry constraint. This may create some overlap in the resulting floorplan, therefore we use the total amount of such overlapping area as the penalty term P_c.

In each annealing iteration, we apply one of the following four moves to change the current floorplan:

—switch two random blocks in one sequence pair;
—rotate random hard blocks;
—switch two random blocks on the horizontal or vertical critical path;
—rotate random hard bocks on the horizontal or vertical critical path.

As mentioned previously, at the floorplan step the blocks contain both hard and soft ones, where the soft blocks are clusters of standard cells and small hard macros. The

shaping of such soft blocks is critical for improving the packing quality of a floorplan. A fast and optimal soft-block shaping algorithm called *SDS* was proposed in Yan and Chu [2012]. In *SDS*, soft blocks are shaped iteratively. During the whole shaping process, the layout height is monotonically reducing and eventually converges to an optimal solution. In our annealing process, after the move is made at each annealing iteration, we apply *SDS* to shape the soft blocks, however, to stop the shaping process, it is not necessary to wait until the layout height converges to the optimal solution. We stop the shaping process as soon as the floorplan layout is within the chip region.

6. WIRELENGTH-DRIVEN SHIFTING

In FLOP, for both default and FLOP-C modes, the wirelength-driven shifting process is formulated as a min-cost flow problem, which is the same as in Tang et al. [2006]. The authors in Tang et al. [2006] show that such a min-cost flow problem can be easily extended to consider various geometry constraints. We use the contour structure [Guo et al. 1999] to derive the horizontal and vertical nonoverlapping constraints among the blocks.

The wirelength-driven shifting is an essential step in FLOP. In terms of the HPWL minimization, it can find the optimal position for each block and provides a globally optimized layout for the analytical placer. Since the shifting step optimizes the HPWL at the floorplan level, it only ignores the local nets among the small objects within each soft block. The smaller the soft block, the fewer nets it ignores, and the better the HPWL we will get at last. However, if the soft blocks become too small, numerous nets will be considered in the shifting. This would slow down the whole algorithm. Because of this, in the partition stopping criteria we set an area bound α, so that the soft blocks would not become too small. On the other hand, we only need the shifting step to generate a globally good layout. Regarding the local nets within the soft blocks, the following analytical placer can handle them very efficiently and effectively.

7. INCREMENTAL PLACEMENT

As mentioned before, the output of the wirelength-driven shifting step is a layout with legal, nonoverlapping locations for the large macros. These large macros are then fixed in place to prevent further movement during any subsequent steps. But, there are multiple "soft blocks" in the layout, each containing numerous "small objects" (i.e., small macros and standard cells). The shifting step assigns these small objects to the center of the corresponding soft block. The incremental placement step uses the initial locations of the small objects as obtained by in previous step and spreads the small objects over the whole chip region to obtain a final overlap-free placement among all objects.

In FLOP, we adopt a similar analytical placement algorithm as *FastPlace* [Viswanathan et al. 2007]. The detailed algorithm flow for this step is presented in Algorithm 1.

8. MMS AND MMS-C BENCHMARKS

The only publicly available benchmarks for mixed-size designs are ISPD02 and ICCAD04 IBM-MS [Adya and Markov 2005; Adya et al. 2004] that are derived from ISPD98 placement benchmarks. As pointed out in Nam et al. [2005], these circuits can no longer be representative of modern VLSI physical design. To continue driving the progress of physical design for the academic community, two suites of placement benchmarks [Nam et al. 2005; Nam 2006] have been recently released. They are directly derived from modern industrial ASICs design. However, in the original circuits most macros have been fixed due to the difficulty of handling movable macros for the existing placers. The authors in Chen et al. [2007, 2008] freed all fixed objects in ISPD06 benchmarks and created new mixed-size placement circuits. But seven out of eight

ALGORITHM 1: Analytical Incremental Placement

```
1:  Phase 0: Physical and Netlist based clustering
2:    initial_objects ← number_of_small_objects
3:    set locations of small objects to center of their soft blocks
4:    while number_of_clusters > target_number_of_clusters do
5:        cluster netlist using Best-choice clustering [Nam et al. 2006]
6:        use physical locations of small objects in clustering score
7:        set cluster_location ← center of gravity of the objects within cluster
8:    end while
9:  end
10: Phase 1: Coarse global placement
11:   generate "fixing forces" for clusters based on their initial locations
12:   solve initial quadratic program (QP)
13:   repeat
14:       perform Cell Shifting [Viswanathan et al. 2007] on coarse-grain clusters
15:       add spreading forces to QP formulation
16:       solve the quadratic program
17:   until placement is roughly even
18:   repeat
19:       perform Iterative Local Refinement [Viswanathan et al. 2007] on coarse-grain clusters
20:   until placement is quite even
21:   uncluster movable macro-blocks
22:   legalize and fix movable macro-blocks
23: end
24: Phase 2: Refinement of fine-grain clusters
25:   while number_of_clusters < 0.5*number_of_small_objects do
26:       uncluster netlist
27:   end while
28:   perform Iterative Local Refinement on fine-grain clusters
29: end
30: Phase 3: Refinement of flat netlist
31:   while number_of_clusters < number_of_small_objects do
32:       uncluster netlist
33:   end while
34:   perform Iterative Local Refinement on flat netlist
35: end
36: Phase 4: Legalization and detailed placement
37:   Legalize the standard cells in the presence of fixed macros
38:   Perform detailed placement [Pan et al. 2005] to further improve wirelength
39: end
```

circuits do not have any fixed I/O objects, which is not realistic in the real designs. In order to recover the complexities of modern mixed-size designs, we modify the original ISPD05/06 benchmarks and derive the Modern Mixed-Size (MMS) placement benchmarks (see Table I). Essentially, we make the following changes on the original circuits.

(i) All macros are freed from the original positions. In the GSRC bookshelf format that the original benchmarks use, both fixed macros and fixed I/O objects are treated as fixed objects; there is no extra specification to differentiate them. So we have to distinguish them only based on the size differences. Basically, if the area of one fixed object is more than $\lambda\times$ the average area of the whole circuit, we will recognize it as a macro. Otherwise, it is a fixed I/O object. Because for each circuit the average area is different, we need to use a different λ (see the last column in Table I) to decide a reasonable number and suitable threshold size for the macros. There is one exception:

Table I. Statistics of the Modern Mixed-Size (MMS) Placement Benchmarks

Circuit	#.Objects	#.Movable Objects	#.Standard Cells	#.Macros	#.Fixed I/O Objects	#.Net	#.Net Pins	Target Density%	λ
adaptec1	211447	210967	210904	63	480	221142	944053	100	70
adaptec2	255023	254584	254457	127	439	266009	1069482	100	160
adaptec3	451650	450985	450927	58	665	466758	1875039	100	650
adaptec4	496054	494785	494716	69	1260	515951	1912420	100	460
bigblue1	278164	277636	277604	32	528	284479	1144691	100	120
bigblue2	557866	535741	534782	959	22125	577235	2122282	100	30
bigblue3	1096812	1095583	1093034	2549	1229	1123170	3833218	100	470
bigblue4	2177353	2169382	2169183	199	7970	2229886	8900078	100	550
adaptec5	843128	842558	842482	76	570	867798	3493147	50	440
newblue1	330474	330137	330073	64	337	338901	1244342	80	2000
newblue2	441516	440264	436516	3748	1252	465219	1773855	90	190
newblue3	494011	482884	482833	51	11127	552199	1929892	80	170
newblue4	646139	642798	642717	81	3341	637051	2499178	50	400
newblue5	1233058	1228268	1228177	91	4790	1284251	4957843	50	570
newblue6	1255039	1248224	1248150	74	6815	1288443	5307594	80	650
newblue7	2507954	2481533	2481372	161	26421	2636820	10104920	80	650

in both circuits *bigblue2* and *bigblue4*, there is one macro that does not connect with any other objects. If this macro is freed, it may cause some trouble for quadratic-based analytical placers, so we keep it fixed. Since this macro is also very small compared with other macros, it would not affect the circuit property.

(ii) The sizes of all I/O objects are set to zero. In MMS benchmarks there are two types of I/Os: *perimeter I/Os* around the chip boundary and *area-array I/Os* spreading across the chip region. Generally, the area-array I/Os are allowed to be overlapped with other movable objects in the design. But existing placers treat all fixed I/Os as fixed objects, so that their algorithms *internally* do not allow such overlaps during the legalization. Since the macros have already been freed in MMS benchmarks, the placers should ignore the overlaps between fixed I/O objects and movable objects, and concentrate on the legalization of movable objects. As we cannot change the code of other placers, one simple way to enforce this is to set the sizes of all I/O objects to zero.

The target density constraints are the same as the original circuits. The same scoring function[4] is used to calculate the scaled HPWL. However, since the macros are movable in the MMS circuits, we need to modify the script used in Nam [2006] to get the correct "scaled_overflow_factor". The modification is that any movable macro that has a width or height greater than the bin dimension used for scaled overflow calculation is now treated as a fixed macro during scaled overflow calculation. Note that this was the method employed by the original script on *newblue1*, which is the only design that has large movable macros in the original circuits. It is required to treat large movable macros as fixed, otherwise we will get an incorrect picture of the placement density.

We have discussed the MMS benchmarks setup with the authors in Nam et al. [2005] and Nam [2006]. To retain the original circuit properties as much as possible, the preceding changes are the best we can do without accessing the original industrial data of the circuits. The MMS benchmarks are publicly available at Yan et al. [2009b].

In MMS benchmarks, all macros are movable and there is no geometry constraint in any circuit. In order to test FLOP in FLOP-C mode, we add three types of geometry constraints, namely, preplaced, range, and boundary constraints, into existing MMS

[4]scaled_HPWL = HPWL * (1 + scaled_overflow_factor).

Table II. List of Geometry Constraints in MMS-C Benchmarks

Circuit	Geometry Constraints				#. non-constraint macros
	#.Preplaced Constraints	#.Range Constraints	#.Boundary Constraints	Total #. Constraints	
adaptec1	4	-	-	4	59
adaptec2	2	1	-	3	134
adaptec3	6	-	-	6	52
adaptec4	1	1	1	3	66
bigblue1	2	5	3	10	22
bigblue2	4	-	-	4	955
bigblue3	2	6	-	8	2541
bigblue4	4	-	-	4	195
adaptec5	2	-	-	2	74
newblue1	1	2	-	3	61
newblue2	-	-	8	8	3740
newblue3	-	-	18	18	33
newblue4	2	4	4	10	71
newblue5	4	-	-	4	87
newblue6	-	8	-	8	66
newblue7	-	8	-	8	153

benchmarks. This new set of benchmarks with geometry constraints is called MMS-C benchmarks. Basically, in each circuit we add the constraints on the larger macros, which is similar to what the designers intend to do in real designs. Still, we keep the nonconstraint macros movable. The detailed constraints information and number of nonconstraint macros in MMS-C benchmarks is shown in Table II. MMS-C benchmarks is the first set of large-scale mixed-size placement benchmarks with various geometry constraints. Compared with both ISPD05/06 and MMS benchmarks, MMS-C benchmarks is much harder for placers to handle, not only because the macros in the circuits are movable, but also because some of them have various geometry constraints.

9. EXPERIMENTAL RESULTS

All experiments were performed on a Linux machine with AMD Opteron 2.6GHz CPU and 8GB memory. The seed of *hMetis2.0* is set to 5.

9.1. Experiments on Benchmarks without Geometry Constraints

This section presents the experimental results of FLOP in default mode on the circuits without any geometry constraint. We set up three sets of experiments.

(i) To test the capability of handling the large-scale mixed-size placement, we compare FLOP with five state-of-the-art mixed-size placers *APlace2*, *NTUplace3*, *mPL6*, *Capo10.5*, and *Kraftwerk* on MMS benchmarks. Before the experiments, we have contacted the authors of each placer mentioned and they provided us their best-available binary for MMS circuits. In Table III, for the ISPD06 circuits (*adaptec5* – *newblue7*) the reported HPWL is the scaled HPWL. *APlace2* crashed on every circuit, so we do not report its results. For the default mode, FLOP generates 10%, 4%, 47%, and 29% better HPWL compared with *NTUplace3*, *mPL6*, *Capo10.5*, and *Kraftwerk*, respectively. Regarding the runtime, FLOP is 11×, 5×, and 16% faster than *Capo10.5*, *mPL6*, and *NTUplace3*. Also FLOP achieves a legal solution on all circuits. We also compare FLOP with the most recent mixed-size placer *MRT* [Hsu and Chang 2010]. We did not run the binary of *MRT* on the same local machine; instead, we directly cited their experimental results on MMS benchmarks from Hsu and Chang [2010]. On average,

Table III. Comparison with Mix-Size Placers on MMS Benchmarks

Circuit	NTUplace3 [Chen et al. 2006]		mPL6 [Chan et al. 2006]		Capo10.5 [Roy et al. 2006]		Kraftwerk [Spindler et al. 2006]		MRT [Hsu et al. 2010]	FLOP	
	HPWL	Time(s)	HPWL	Time(s)	HPWL	Time(s)	HPWL	Time(s)	HPWL	HPWL	Time(s)
adaptec1	80.45	630	77.84	2404	84.77	5567	86.73	285	79.05	73.09	470
adaptec2	136.46	1960	88.40	2870	92.61	7373	108.61	408	84.26	82.54	708
adaptec3	*illegal*	–	180.64	5983	202.37	16973	272.76	773	168.11	170.51	1172
adaptec4	177.23	1896	162.02	5971	202.38	17469	260.96	962	156.33	166.18	1385
bigblue1	95.11	955	99.36	2829	112.58	8458	130.78	348	99.78	93.53	1492
bigblue2	144.42	1661	144.37	12202	149.54	17647	*aborted*	–	149.96	146.41	1956
bigblue3	*illegal*	–	319.63	9548	583.37	69921	417.73	2369	392.72	343.85	2869
bigblue4	764.72	8182	804.00	23868	915.73	109785	969.03	6000	809.41	768.54	11562
adaptec5*	*aborted*	–	376.30	22636	565.88	23957	402.21	1653	306.12	333.72	2849
newblue1*	60.73	875	66.93	3171	110.54	3133	76.22	481	61.75	66.01	831
newblue2*	*aborted*	–	179.18	6044	303.25	8156	272.67	615	170.80	185.01	1572
newblue3*	*aborted*	–	415.86	17623	1282.19	73339	374.54	578	223.38	343.34	2548
newblue4*	*aborted*	–	277.69	9732	300.69	6589	291.45	1266	253.39	237.42	2272
newblue5*	*aborted*	–	515.49	24806	570.32	16548	503.13	2639	450.07	452.04	5728
newblue6*	*aborted*	–	482.44	13112	609.16	18076	651.34	2726	485.73	492.52	4966
newblue7*	*aborted*	–	1038.66	31680	1481.45	43386	*illegal*	–	1125.84	1057.15	12567
Norm	**1.10**	**1.15**	**1.04**	**4.61**	**1.47**	**10.58**	**1.29**	**0.52**	**0.98**	**1**	**1**

(* comparison of scaled HPWL), HPWL($\times 10e6$).

Table IV. Comparison among Various Versions of FLOP

Circuit	FLOP-NR-NC		FLOP-NI-NC		FLOP-NC		FLOP	
	HPWL	Time(s)	HPWL	Time(s)	HPWL	Time(s)	HPWL	Time(s)
adaptec1	77.18	568	85.27	741	76.83	634	73.09	470
adaptec2	87.17	976	86.72	1153	84.14	974	82.54	708
adaptec3	182.21	1635	173.07	2004	175.99	1767	170.51	1172
adaptec4	166.55	1791	175.67	2148	161.68	1886	166.18	1385
bigblue1	95.45	1179	98.91	1385	94.92	1218	93.53	1492
bigblue2	150.66	1713	162.40	2341	153.02	1828	146.41	1956
bigblue3	372.79	5973	394.75	5377	346.24	4283	343.85	2869
bigblue4	807.53	10450	839.53	12773	777.84	10516	768.54	11562
adaptec5*	381.83	4621	385.07	3522	357.83	2933	333.72	2849
newblue1*	73.36	1214	71.69	1156	67.97	889	66.01	831
newblue2*	231.94	2153	190.50	1957	187.40	1541	185.01	1572
newblue3*	344.71	1427	355.07	1998	345.99	1518	343.34	2548
newblue4*	256.91	1852	268.46	2614	256.54	1786	237.42	2272
newblue5*	516.71	5147	536.38	6669	510.83	5287	452.04	5728
newblue6*	502.24	6211	506.99	7524	493.64	6348	492.52	4966
newblue7*	1113.07	10168	1101.07	13053	1078.18	11298	1057.15	12567
Norm	**1.07**	**1.17**	**1.09**	**1.32**	**1.02**	**1.08**	**1**	**1**

(* comparison of scaled HPWL), HPWL($\times 10e6$).

under the default mode FLOP generates 2% worse wirelength than *MRT*. Note that *MRT* was developed after the preliminary version of FLOP was published in Yan et al. [2009a]. Among the placers listed in Table III, only *MRT* and FLOP can optimize the macro-orientations.

(*ii*) In order to show the individual contribution of the main techniques in FLOP, we turn on/off three features in FLOP, namely, clustering process, macro-rotation, and incremental placement. The experimental results are presented in Table IV. FLOP-NC turns off the stand-alone clustering process in the block formation step. Based on FLOP-NC, FLOP-NR-NC also restricts the rotation on all macros, and FLOP-NI-NC also discards the initial positions of small objects in the incremental placement step. Compared with FLOP-NR-NC, FLOP-NC generates 5% better HPWL by rotating the macros. By doing nonincremental placement, FLOP-NI-NC produces 7% worse HPWL and is 24% slower than FLOP. By employing *SafeChoice* clustering in the block formation step, on average FLOP generates 2% better HPWL with 8% faster runtime than FLOP-NC.

(*iii*) We compare FLOP with previous mixed-size placement algorithms that use the two-stage approach. In this experiment, three leading macroplacers *CG*, *MPT*, and *XDP* are used to place the macros and optimize their orientations, followed by *NTUplace3* to place the remaining standard cells. Due to the IP issues, their binaries are not available, but the authors sent us the benchmarks used in Chen et al. [2008]. So in Table V the other placers' results are cited from Chen et al. [2008]. These benchmarks allow the rotation of macros and do not consider the target density. As shown in Table V, FLOP achieves 1%, 12%, and 7% better HPWL compared with *CG+NTUplace3*, *MPT+NTUplace3*, and *XDP+NTUplace3*, respectively. In the table, as a reference we also list the stand-alone *NTUplace3*'s results.

9.2. Experiments on Benchmarks with Geometry Constraints

This section presents the experimental results of FLOP in the FLOP-C mode on MMS-C benchmarks. In the current implementation, none of the state-of-the-art mixed-size

Table V. Comparison with Macroplacers on Modified ISPD06 Benchmarks [Chen et al. 2008] with Default Chip Utilization

Circuit	CG [Chen et al. 2008] +NTUplace3 HPWL(×10e7)	MPT [Chen et al. 2007] +NTUplace3 HPWL(×10e7)	XDP [Cong and Xie 2006] +NTUplace3 HPWL(×10e7)	NTUplace3 HPWL(×10e7)	FLOP HPWL(×10e7)
adaptec5	29.46	31.01	31.08	29.03	27.36
newblue1	6.23	6.50	6.32	6.06	7.32
newblue2	18.89	22.60	18.90	28.09	23.79
newblue3	30.18	37.57	37.64	53.48	33.61
newblue4	21.38	23.77	22.01	22.83	19.72
newblue5	42.92	43.71	45.41	39.91	36.66
newblue6	44.93	50.50	46.43	44.24	41.87
newblue7	99.03	108.06	102.21	100.06	86.96
Norm	**1.01**	**1.12**	**1.07**	**1.14**	**1**

Table VI. Comparison with Mixed-Size Placers on MMS-C Benchmarks with Geometry Constraints

Circuit	NTUplace3 [Chen et al. 2006] HPWL (×10e6)	Time (s)	mPL6 [Chan et al. 2006] HPWL (×10e6)	Time (s)	Capo10.5 [Roy et al. 2006] HPWL (×10e6)	Time (s)	FLOP-C HPWL (×10e6)	Time (s)
adaptec1†	98.36	804	99.28	2410	100.13	4988	98.11	2267
adaptec2	×	–	×	–	×	–	159.38	1021
adaptec3†	187.90	1268	*illegal*	–	252.64	15322	207.56	2093
adaptec4	×	–	×	–	×	–	195.14	1572
bigblue1	×	–	×	–	×	–	103.37	1209
bigblue2†	180.33	1760	*illegal*	–	*illegal*	–	167.80	4360
bigblue3	×	–	×	–	×	–	540.14	8101
bigblue4†	936.34	13348	*illegal*	–	912.72	102460	901.64	16519
adaptec5†	*illegal*	–	*illegal*	–	385.86	34277	355.05	4170
newblue1	×	–	×	–	×	–	80.75	1862
newblue2	×	–	×	–	×	–	342.25	5887
newblue3	×	–	×	–	×	–	552.90	3482
newblue4	×	–	×	–	×	–	250.33	2501
newblue5†	*illegal*	–	*illegal*	–	550.69	66594	546.80	7454
newblue6	×	–	×	–	×	–	531.02	13177
newblue7	×	–	×	–	×	–	2178.60	9075
Norm	**1.01**	**0.54**	**1.01**	**1.06**	**1.07**	**6.58**	**1**	**1**

† circuit has only preplaced constraints; × circuit has constraints that cannot be handled.

placement algorithms supports the geometry constraints. But, they can handle the preplaced constraint indirectly by simply treating those constraint macros as placement blockages. So we can only compare FLOP-C with other mixed-size placers on the six circuits that only have preplaced constraints in MMS-C benchmarks.

The experimental results on MMS-C benchmarks are shown in Table VI. We compare FLOP-C with *NTUplace3*, *mPL6*, and *Capo10.5*. It is clear that FLOP-C is able to successfully satisfy all geometry constraints on all 16 circuits. Even for the six circuits with only preplaced constraints, it is still very difficult for *NTUplace3*, *mPL6*, and *Capo10.5* to produce a legal solution. For example, *mPL6* failed to generate a legal solution on five circuits. On average, FLOP-C generates 1%, 1%, and 7% of better HPWL than *NTUplace3*, *mPL6*, and *Capo10.5*, respectively. Regarding the runtime, FLOP-C is 7× and 6% faster than *Capo10.5* and *mPL6*, respectively. Figure 9(a)–9(p) show the layouts with only constraint macros that are generated by FLOP-C for each circuit.

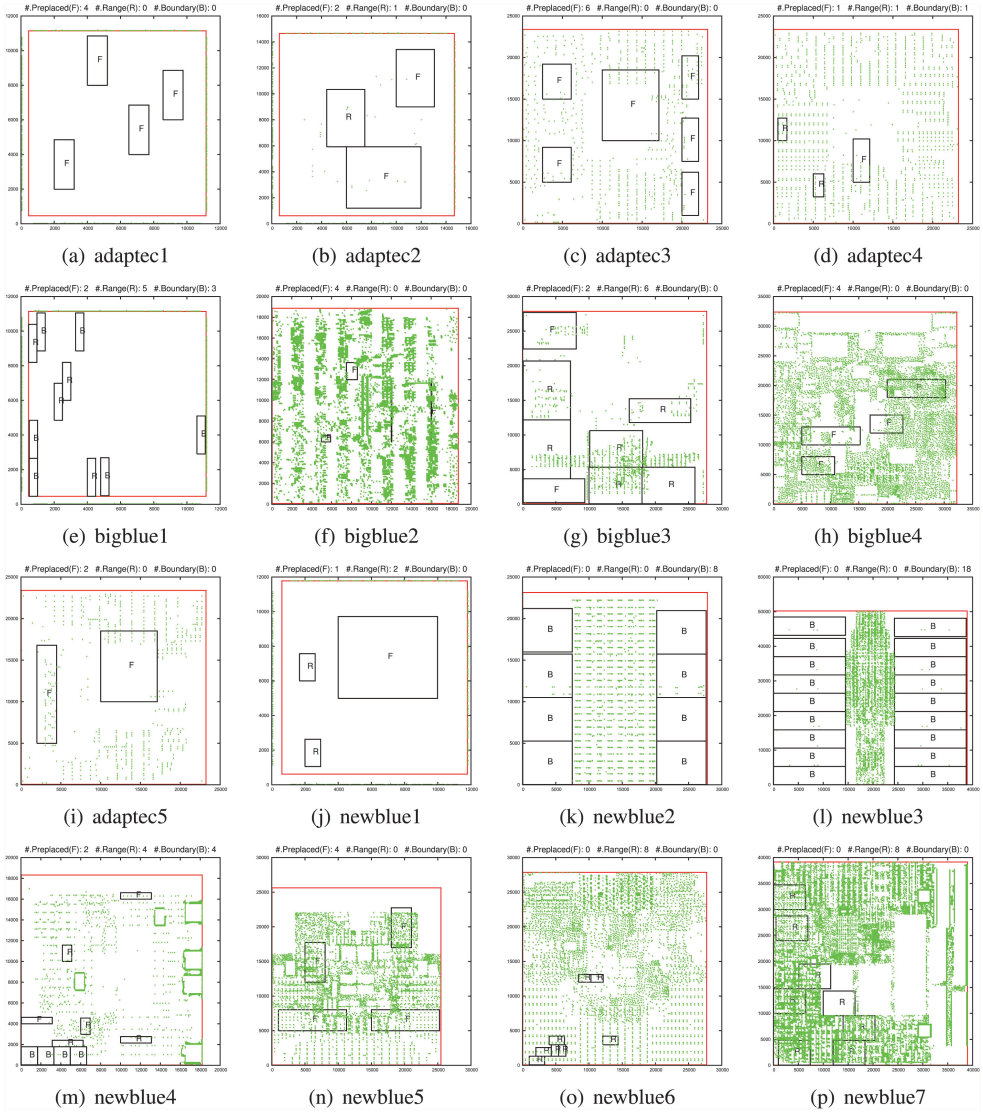

Fig. 9. Layouts containing only constraint macros generated by FLOP-C in MMS-C benchmarks. (Preplaced, range, and boundary constraint macros are marked with F, R, and B, respectively. Green dot denotes I/O object and red line denotes chip boundary.)

Note that in MMS-C benchmarks the nonconstraint macros are all movable. The problem of simultaneously handling both movable and constraint macros is inherently much harder than either the one of handling all preplaced macros (e.g., in ISPD05/06 benchmarks) or the one of handling all movable macros (e.g., in MMC benchmarks). For example, for the design with prefixed constraint in MMS-C benchmarks, the placer needs to place the nonconstraint large movable macros into the remaining whitespace that is not occupied by the preplaced macros, which is not trivial. This is the main reason why some placers can find a legal placement in ISPD05/06 or MMS benchmarks, but not in MMS-C benchmarks.

Table VII. Circuit Data at Floorplan Level after Block Formation Step in FLOP

Circuit	#.Hard Block	#.Soft Block	#.Net	#.Net Pins
adaptec1	63	632	19997	61638
adaptec2	127	324	19611	60572
adaptec3	58	372	21956	65285
adaptec4	69	681	27283	77157
bigblue1	32	987	32261	110710
bigblue2	959	612	35650	120656
bigblue3	2549	348	41020	140266
bigblue4	199	683	75676	282380
adaptec5	76	485	30213	89283
newblue1	64	685	19771	57853
newblue2	3748	429	37200	144564
newblue3	51	101	33291	115417
newblue4	81	823	39645	118958
newblue5	91	713	47208	188379
newblue6	74	1078	59705	211887
newblue7	161	578	96103	387367

Table VIII. Comparison of HPWL at Floorplan Level before and after Wirelength-Driven Shifting Step in FLOP

Circuit	Before shifting	After shifting	Relative Change
adaptec1	85488288	83179008	−2.7%
adaptec2	93557648	88376584	−5.5%
adaptec3	213671392	202221584	−5.4%
adaptec4	186671440	180954848	−3.1%
bigblue1	114924416	113103560	−1.6%
bigblue2	222146784	209748384	−5.6%
bigblue3	361922528	350726656	−3.1%
bigblue4	918914752	891681408	−3.0%
adaptec5	423175552	410239008	−3.1%
newblue1	71667856	69801384	−2.6%
newblue2	266253680	244399568	−8.2%
newblue3	616148800	573167808	−7.0%
newblue4	295049824	289260544	−2.0%
newblue5	576941952	560006976	−3.0%
newblue6	644556608	599088960	−7.0%
newblue7	1441669888	1353312128	−6.1%

9.3. Data Analysis of FLOP

In FLOP, the circuit size will be reduced for floorplanning at the block formation step. Table VII shows the detailed circuit data information at the floorplan level for FLOP. Comparing Table I with Table VII, one can see that the circuit size has been cut down significantly from millions to thousands of objects.

In order to show the effectiveness of the wirelength-driven shifting step, in Table VIII we compare the floorplan-level HPWL before and after applying the shifting process. The data shows that wirelength-driven shifting significantly reduces the wirelength. For example, for *newblue2*, the HPWL has been improved 8.2% after the shifting process.

Table IX. Runtime Breakdown of FLOP in Two Operation Modes

Mode	Block Formation	Floorplanning	Wirelength-driven Shifting	Incremental Placement
Default Mode	42.44%	8.97%	0.50%	48.09%
FLOP-C Mode	39.06%	27.79%	0.67%	32.48%

Table IX shows the runtime breakdown of FLOP in the default and FLOP-C modes. For both operation modes, FLOP's runtime is dominated by two steps, namely, block formation and incremental placement, which together contribute 70% to 90% of the total runtime, while the wirelength-driven shifting step only takes a very small portion (i.e., less than 1%). Because we add the annealing process into the floorplanning step in the FLOP-C mode, the portion of the floorplanning step's runtime increases from 8.97% to 27.79%. Even so, the bottleneck of the total runtime in the FLOP-C mode is not the annealing-based floorplanner. This demonstrates the efficiency of the enhanced annealing process we propose. The runtime of FLOP can be further improved by refining the clustering strategy in the block formation step.

10. CONCLUSION AND FUTURE WORK

This article presents a new algorithm flow for modern large-scale mixed-size placement, both with and without geometry constraints. To show the effectiveness of such flow, we implemented a high-quality and efficient mixed-size placer FLOP, and constructed two new sets of benchmarks, MMS and MMS-C. Compared with most state-of-the-art mixed-size placers and leading macroplacers, FLOP achieves the best HPWL and easily produces the legal layout for every circuit with all geometry constraints satisfied.

Due to the ever-increasing complexity of ICs, sometimes the real design could be much more complex than the one addressed in this work. For example, hundreds or even thousands of movable macros may be subject to multiple constraints. In this case, the annealing-based framework may no longer be sufficient. So one future work will still be based on the proposed new floorplan-guided placement flow, but focusing on developing a more scalable and robust nonstochastic floorplanning algorithm to handle a very great number of geometry constraints. Moreover, as the routability becomes more and more critical as the design goes to 20nm and below, another future work is to model the routing cost as early as the floorplanning step in the new flow.

ACKNOWLEDGMENTS

The authors would like to thank Prof. George Karypis, Dr. Gi-Joon Nam, Prof. Guojie Luo, Dr. Tung-Chieh Chen and Dr. Peter Spindler for the help with *hMetis 2.0*, ISPD05/06 placement benchmarks, *mPL6*, *NTUplace3* and *Kraftwerk*, respectively. Also thanks goes to Dr. Yi-Lin Chuang and Prof. Yao-Wen Chang for providing us their benchmarks.

REFERENCES

S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov. 2004. Unification of partitioning, placement and floorplanning. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'04)*. 550–557.

S. N. Adya and I Markov. 2005. Combinatorial techniques for mixed-size placement. *ACM Trans. Des. Autom. Electron. Syst.* 10, 1, 58–90.

T. Chan, J. Cong, J. Shinnerl, K. Sze, and M. Xie. 2006. mPL6: Enhanced multilevel mixed-sized placement. In *Proceedings of the International Symposium on Physical Design (ISPD'06)*. 212–214.

H.-C. Chen, Y.-L. Chuang, Y.-W. Chang, and Y.-C. Chang. 2008. Constraint graph-based macro placement for modern mixed-size circuit designs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'08)*. 218–223.

T.-C. Chen, Y.-W. Chang, and S.-C. Lin. 2005. IMF: Interconnect-driven multilevel floorplanning for large-scale building-module designs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (*ICCAD'05)*. 159–164.

T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. 2006. A high-quality mixed-size analytical placer considering preplaced blocks and density constraints. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'06)*. 187–192.

T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and D. Liu. 2007. MP-tree: A packing-based macro placement algorithm for mixed-size designs. In *Proceedings of the 44th Annual Design Automation Conference (DAC'07)*. 447–452.

J. Cong and M. Xie. 2006. A robust detailed placement for mixed-size IC designs. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'06)*. 188–194.

J. Egeblad. 2003. Placement techniques for VLSI layout using sequence-pair legalization. Ph.D. dissertation. http://www.diku.dk/∼jegeblad/thesis.pdf.

P.-N. Guo, C.-K. Cheng, and T. Yoshimura. 1999. An o-tree representation of non-slicing floorplan and its applications. In *Proceedings of the 36th Design Automation Conference (DAC'99)*. 268–273.

M.-K. Hsu and Y.-W. Chang. 2010. Unified analytical global placement for large-scale mixed-size circuit designs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'10)*. 657–662.

A. B. Kahng and Q. Wang. 2005. Implementation and extensibility of an analytical placer. *IEEE Trans. Comput.-Aid. Des.* 24, 5, 734–747.

A. B. Kahng and Q. Wang. 2006. A faster implementation of aplace. In *Proceedings of the International Symposium on Physical Design (ISPD'06)*. 218–220.

G. Karypis and V. Kumar. 1999. hMetis2.0. http://glaros.dtc.umn.edu/gkhome/.

M.-C. Kim and I. Markov. 2012. ComPLx: A competitive primal-dual lagrange optimization for global placement. In *Proceedings of the 49th Annual Design Automation Conference (DAC'12)*. 747–752.

C. Kodama, K. Fujiyoshi, and T. Koga. 2004. A novel encoding method into sequence-pair. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS'04)*. 329–332.

Q. Ma, L. Xiao, Y.-C. Tam, and E. F. Y. Young. 2011. Simultaneous handling of symmetry, common centroid, and general placement constraints. *IEEE Trans. Comput.-Aid. Des.* 30, 1, 85–95.

H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. 1996. VLSI module placement based on rectangle-packing by the sequence-pair. *IEEE Trans. Comput.-Aid. Des.* 15, 12, 1518–1524.

G.-J. Nam. 2006. ISPD 2006 placement contest: Benchmark suite and results. In *Proceedings of the International Symposium on Physical Design (ISPD'06)*. 167–167.

G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz. 2005. The ISPD 2005 placement contest and benchmarks suite. In *Proceedings of the International Symposium on Physical Design (ISPD'05)*. 216–220.

G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, and A. B. Kahng. 2006. A fast hierarchical quadratic placement algorithm. *IEEE Trans. Comput.-Aid. Des.* 25, 4, 678–691.

M. Pan, N. Viswanathan, and C. Chu. 2005. An efficient and effective detailed placement algorithm. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'05)*. 48–55.

J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov. 2006. Min-cut floorplacement. *IEEE Trans. Comput.-Aid. Des.* 25, 7, 1313–1326.

J. A. Roy, A. N. Ng, R. Aggarwal, V. Ramachandran, and I. L. Markov. 2009. Solving modern mixed-size placement instances. *Integr.* 42, 2, 262–275.

K. Sakanushi, Y. Kajitani, and D. P. Mehta. 2003. The quarter-state-sequence floorplan representation. *IEEE Trans. Circ. Syst. I Fundam. Theory Appl.* 50, 3, 376–386.

P. Spindler and F. M. Johannes. 2006. Fast and robust quadratic placement combined with an exact linear net model. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'06)*. 179–186.

T. Taghavi, X. Yang, B.-K. Choi, M. Yang, and M. Sarrafzadeh. 2006. Dragon2006: Blockage-aware congestion-controlling mixed-size placer. In *Proceedings of the International Symposium on Physical Design (ISPD'06)*. 209–211.

Y.-C. Tam, E. F. Y. Young, and C. C. N. Chu. 2006. Analog placement with symmetry and other placement constraints. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'06)*. 349–354.

X. Tang, R. Tian, and M. D. F. Wong. 2006. Minimizing wire length in floorplanning. *IEEE Trans. Comput.-Aid. Des.* 25, 9, 1744–1753.

N. Viswanathan, M. Pan, and C. Chu. 2007. FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'07)*. 135–140.

J. Z. Yan and C. Chu. 2010. DeFer: Deferred decision making enabled fixed-outline floorplanning algorithm. *IEEE Trans. Comput.-Aid. Des.* 43, 3, 367–381.

J. Z. Yan and C. Chu. 2012. Optimal slack-driven block shaping algorithm in fixed-outline floorplanning. In *Proceedings of the ACM International Symposium on Physical Design (ISPD'12)*. 179–186.

J. Z. Yan, C. Chu, and W. K. Mak. 2011. SafeChoice: A novel approach to hypergraph clustering for wirelength-driven placement. *IEEE Trans. Comput.-Aid. Des.* 30, 7, 1020–1033.

J. Z. Yan, N. Viswanathan, and C. Chu. 2009a. Handling complexities in modern large-scale mixed-size placement. In *Proceedings of the 46th Annual Design Automation Conference (DAC'09)*. 436–441.

J. Z. Yan, N. Viswanathan, and C. Chu. 2009b. MMS placement benchmarks. http://www.public.iastate.edu/zijunyan/.

E. F. Y. Young, C. C. N. Chu, and M. L. Ho. 2004. Placement constraints in floorplan design. *IEEE Trans. VLSI Syst.* 12, 7, 735–745.

Received August 2013; revised November 2013; accepted February 2013