Constraint Graph-Based Macro Placement for Modern Mixed-Size Circuit Designs

Hsin-Chen Chen[1], Yi-Lin Chuang[2], Yao-Wen Chang[1,2], and Yung-Chung Chang[3]
[1]Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
[2]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
[3]Genesys Logic, Inc., Shindian City, Taipei County 231, Taiwan
{indark, nicky}@eda.ee.ntu.edu.tw; ywchang@cc.ee.ntu.edu.tw; Bennice.Chang@genesyslogic.com.tw

Abstract— **In this paper, we propose a constraint graph-based macro placement algorithm that removes macro overlaps and optimizes macro positions for modern mixed-size circuit designs. Improving over the constraint graph by working only on its essential edges without loss of the solution quality, our algorithm can search for high-quality macro placement solutions effectively and efficiently. Instead of packing macros along chip boundaries like most recent previous work, our placer can determine a non-compacted macro placement by linear programming and placement region cost evaluation and handle various placement constraints/objectives. Compared with various leading academic macro placers, our algorithm can consistently and significantly reduce the wirelengths for designs with different utilization rates, implying that our macro placer is robust and has very high quality.**

I. Introduction

Due to the extensive use of intellectual property (IP) modules and pre-designed macro blocks (such as embedded memories, analog blocks, pre-designed datapaths, etc.), the number of macros in a modern circuit design is increasing dramatically. It is reported in [3] that a modern system-on-a-chip (SoC) design may consist of hundreds of macros, and they may occupy more than 70% chip area. For such large-scale designs, it is impractical to place macros manually. Further, modern circuit designs usually contain large macros with areas more than 10,000 times greater than that of a standard cell. The size and aspect-ratio differences of the macros make this problem even more complicated. Consequently, many mixed-size placement flows/algorithms are proposed in the recent literature to tackle this problem.

A. *Previous Work*

According to the macro handling methods, we can classify the mixed-size placement algorithms into three types: (1) Simultaneous macro and standard-cell placement: Most existing mixed-size placers [4]–[8], [13], [17], [19] belong to this type. However, as the number of macros and the size difference between the macros and standard cells increase dramatically, this methodology incurs significant difficulties in legality and complexity. (2) Constructive macro placement: Most partitioning-based placers [11], [18] keep macro overlap free during the placement process by recursively partitioning the chip/macros into subregions. Due to the intrinsic limitation of the partitioning-based approach, its solution quality is usually limited, compared with those in the first type. (3) Two-stage mixed-size placement: The combinatorial technique [2], [9], [10] belong to this type. Methods of this type consist of macro placement followed by standard-cell placement. Given an initial placement that considers both macros and standard cells and optimizes a predefined cost metric (e.g., wirelength), legal macro positions are determined with the minimum displacement in the macro placement stage. Then, in the standard-cell placement stage, the macros are fixed and the standard cells are placed into the rest area. Compared with the previous two types of mixed-size placement approaches, this two-stage approach is more robust since it can guarantee a feasible solution as long as an overlap-free macro placement is obtained. Further, macro orientations and placement constraints, such as pre-placed macros and placement blockages, can be handled more easily. Thus, the two-stage approach is widely used in industry.

The MP-tree [9], [10] was proposed to solve the addressed macro placement problem. Given an initial placement, the MP-tree macro placer removes overlaps, minimizes displacement, and maximizes the area of the chip central regions for the standard-cell placement by a multi-packing-tree representation. Although the MP-tree is effective for higher-density designs, its solution quality for lower-density designs is limited. The leading macro legalizers, XDP [12] and Floorist [16], can also be applied to determine the legalized macro positions. For a given initial placement with macro overlaps, XDP constructs constraint graphs and iteratively swaps the edges between the constraint graphs by a min-cut-based edge-selection heuristic. In contrast, Floorist removes macro overlaps by an incremental construction of constraint graphs. Since these two macro legalizers optimize the macro positions by iterative methods, their search spaces are often limited, and thus they may not find a desired macro placement.

B. *Our Contributions*

We adopt the two-stage mixed-size placement approach because of its various advantages and popularity in industry. Our work focuses on the first stage, macro placement, which is crucial for modern mixed-size placement since macro positions significantly affect the final placement quality. For the addressed problem, we propose a constraint graph-based macro placement algorithm that removes macro overlaps and optimizes macro positions/orientations. In this paper, we choose the transitive closure graph (TCG) [14], [15] as our constraint graph due to its good properties. (Nevertheless, our algorithm is flexible and can be based on different constraint graphs as well.) Based on the TCG floorplan representation [14], [15], our algorithm searches for desired macro positions through a new *adaptive* simulated annealing (SA) scheme. For a feasible TCG, we further adopt a linear programming formulation to determine the legalized macro positions with displacement minimization. We summarize the advantages of our macro placement algorithm as follows:

- By linear programming and bin-based standard-cell placement region cost evaluation, our placer can determine a non-compacted macro placement and preserve a continuous region for standard-cell placement. Compared with the MP-tree, our macro placer can achieve 11% better average wirelength based on the ISPD'06 placement benchmarks.
- Our approach smoothly controls the macro distributions under different chip utilization rates. As a result, our macro placer significantly outperforms NTUplace3 alone, XDP, and MP-tree by 4%–14% under various pratical utilization rates. Even for the 90% utilization rate, our macro placer obtains much shorter wirelengths than NTUplace3 alone and XDP, and is comparable to MP-tree which is specially developed for high-density designs.
- Without loss of the solution quality, our algorithm only needs to work on a subset of the edges in TCG (specifically, *the*

reduction edges as defined in [14], [15]), significantly improving the efficiency without trading off the solution quality.

- We propose a new adaptive SA flow with a smoothing technique for cost evaluation to reduce the number of linear programs solved and minimize the gap between two different evaluation methods (namely corner packing and linear programming).
- Our macro placer is very robust. Combined with various leading academic placers (such as mPL6, Capo 10.2, and NTUplace3), it can consistently and significantly reduces the wirelength.

Table 1 summarizes the comparisons among our macro placer, MP-trees, and two constraint graph-based macro legalizers, XDP and Floorist.

The rest of this paper is organized as follows. Section II defines the macro placement problem. Section III details our macro placement algorithm. Experimental results are presented in Section IV. Finally, conclusions are given in Section V.

II. Problem Formulation

In this section, we first describe our mixed-size placement flow and then define the addressed macro placement problem.

A. The Mixed-Size Placement Flow

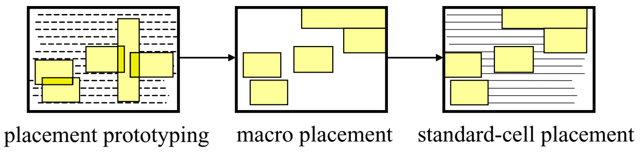

Fig. 1. Our mixed-size placement flow.

Figure 1 summarizes our mixed-size placement flow. Given the circuit information, a placement prototype is first generated by wirelength-driven mixed-size global placement that considers both the macros and standard cells. The placement prototype may contain macro overlaps. For the given macro positions, our macro placer not only legalizes the macros, but also optimizes the macro orientations. With the objective of displacement minimization and placement region optimization, our macro placer explicitly optimizes the wirelength to find a better final placement. Finally, all macros are fixed and a standard-cell placer is then applied to place all standard cells in the remaining area.

B. The Macro Placement Problem

In the mixed-size placement flow, macro placement plays an important role and significantly affects the final placement. Thus, we shall consider the macro placement problem formulated as follows:

- **Macro Placement Problem:** Given an initial placement which contains a set $M = \{m_1, m_2, \cdots, m_n\}$ of n macros, where the respective coordinates, width, height, and area of m_i are (x_i, y_i), W_i, H_i, and A_i, and a set of pre-placed macros $F \subset M$, the problem is to find legalized macro positions that minimizes some cost metric (e.g., macro displacement from their original positions, wirelength, placement region cost).

III. Our Macro Placement Algorithm

Our underlying strategy is to place macros inside the placement region and reserve a continuous standard-cell placement region considering displacement minimization. Since the macros in modern designs are usually very large and there are routing blockages above macros in real-world applications, the macros tend to block the routes if they are placed in the chip center. Further, by minimizing the macro displacement, we can keep the desired wirelength implicitly since the initial global placement has been optimized for wirelength.

We present a TCG-based macro placement algorithm that removes macro overlaps and optimizes macro positions and orientations. For a feasible TCG, we solve a linear program to determine legalized macro positions with displacement minimization and standard-cell placement region optimization. We define a macro placement evaluation metric which consists of wirelength, displacement, and placement region costs. Based on TCG and the cost metric, our algorithm searches for better macro configurations through a new *adaptive* simulated annealing (SA) scheme. We shall detail those techniques in the following sections.

A. Macro Placement Using TCG

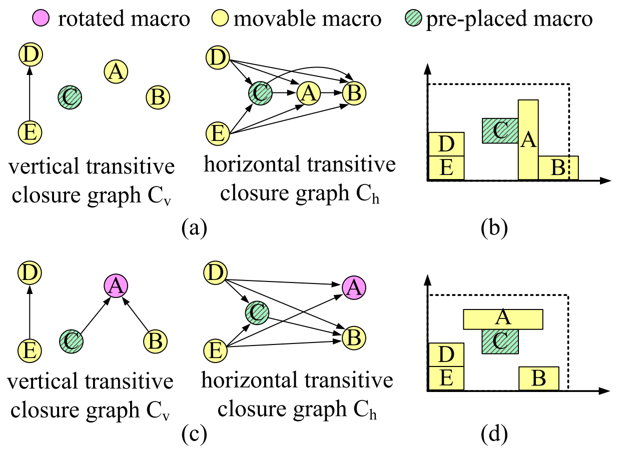
rotated macro
movable macro
pre-placed macro
D
A
C
B
E
vertical transitive closure graph C_v
horizontal transitive closure graph C_h
(a)
(b)
(c)
(d)

Fig. 2. (a) An infeasible TCG. (b) The corresponding placement exceeds the chip boundary. (c) A feasible and optimized TCG, (d) and its corresponding placement.

Due to the chip outline constraint and the existence of pre-placed macros, a macro geometric relation represented by a TCG may not produce legalized macro positions. Take the TCG shown in Figure 2(a) as an example. The corresponding placement (Figure 2(b)) cannot fit into the chip outline due to the pre-placed macro C. Thus, we define the feasibility of a TCG as follows:

Definition 1: A TCG is *feasible* if there exists a legalized macro placement that can fit into the chip outline under the geometric relations defined by the TCG.

We compute the slacks for each vertex to check if a constraint graph is feasible. The slack is the feasible position range for a macro without violating the TCG properties. For a given TCG, we first add two pseudo macros m_s and m_t representing the source v_s and the sink v_t in C_h. The widths and heights of the pseudo macros m_s and m_t are set to zero, and their initial positions are set to the corresponding chip boundaries, *i.e.*, $(x_s, y_s) = (L, B)$ and $(x_t, y_t) = (R, T)$, where L, R, T, and B represent the left, right, top, and bottom boundaries of the placement region, respectively. We then connect v_s to the vertices with zero in-degree and the vertices with zero out-degree to v_t. Let C'_h denote the modified graph, and l_{v_i} and u_{v_i} denote the leftmost and rightmost coordinates that a macro m_i can place. The slack s_{v_i} of each vertex v_i in C'_h can be calculated as follows:

$$\begin{aligned} l_{v_i} &= u_{v_i} = x_i, && \forall m_i \in F' \text{ and } v_i \in C'_h, \\ l_{v_j} &= \max(l_{v_i} + w_{ij}), && \forall e_{ij} \in C'_h, \\ u_{v_i} &= \min(u_{v_j} - w_{ij}), && \forall e_{ij} \in C'_h, \\ s_{v_i} &= u_{v_i} - l_{v_i}, && \forall v_i \in C'_h, \end{aligned}$$

where $F' = F \cup \{m_s, m_t\}$, and w_{ij} is the weight of the edge e_{ij} in C'_h, which equals $(W_i + W_j)/2$. C'_v and the corresponding values for the vertical dimension can be similarly defined.

This procedure, at first glance, is similar to the one proposed in [12] (XDP), but we extend it to consider pre-placed macros and add significant modifications to handle the addressed problem. (See Section IV-A for the different effects of XDP and our method.) We then define the infeasibility value κ of a TCG by

$$\kappa = -min(s_{v_i}), \quad \forall v_i \in C'_h, C'_v. \tag{1}$$

Lemma 1: A TCG is feasible if and only if $\kappa = 0$.

Unlike [12] that resorts to an iterative graph refinement process, we model the infeasibility into our cost function to guide the adaptive SA process. Figure 2(c) shows a feasible and optimized TCG, and (d) its corresponding placement.

TABLE I

COMPARISONS BETWEEN OUR MACRO PLACER, MP-TREE, XDP, AND FLOORIST.

	MP-tree	XDP	Floorist	Ours
Topology representation	Multi-packing tree	Constraint graph	Constraint graph	Transitive closure graph
Optimization method	Simulated annealing	Iterative refinement	Iterative refinement	Adaptive SA
Optimization objective	Displacement, wirelength, standard-cell placement region	Displacement minimization	Displacement minimization	Displacement, wirelength, standard-cell placement region, routability
Macro position determination	Packing procedure	Linear programming	Greedy heuristic	Linear programming and corner packing
Orientation optimization	Yes	No	No	Yes
Pre-placed macro consideration	Yes	No	No	Yes

B. Linear Programming Formulation

In order to minimize the macro displacement and preserve a better standard-cell placement region, we solve the following linear programming problem to determine the macro coordinates after a feasible TCG is obtained.

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}(\omega_{x_i} d_{x_i} + \omega_{y_i} d_{y_i}) & & \\ s.t. \quad & -d_{x_i} \leq x'_i - \hat{x}_i \leq d_{x_i} & & (2) \\ & -d_{y_i} \leq y'_i - \hat{y}_i \leq d_{y_i} & & (3) \\ & x'_j - x'_i \geq w^h_{ij}, & \forall e_{ij} \in E'_h, & \quad (4) \\ & y'_j - y'_i \geq w^v_{ij}, & \forall e_{ij} \in E'_v, & \quad (5) \\ & x'_i = x_i,\ y'_i = y_i, & \forall m_i \in F', & \end{aligned}$$

where $\hat{x}_i$ ($\hat{y}_i$) is the expanded displacement reference coordinate, d_{x_i} (d_{y_i}) denotes the displacement along the x (y) direction w.r.t. $\hat{x}_i$ ($\hat{y}_i$), E'_h (E'_v) is the set of all *reduction edges* in C'_h (C'_v), w^h_{ij} and w^v_{ij} are the edge weights defined in C'_h and C'_v, and Inequalities (4) and (5) are derived from the reduction edges of a TCG. Here, as defined in [14], [15], an edge e_{ij} is said to be a *reduction edge* if there does not exist another path from v_i to v_j, except the edge e_{ij} itself.

To reserve a continuous region for standard-cell placement, we expand the displacement reference positions to free a "central" region for the standard-cell placement. The expanded reference position is calculated by

$$\hat{x}_i = (1-\rho)c_x + \rho x_i, \qquad \hat{y}_i = (1-\rho)c_y + \rho y_i,$$

where c_x and c_y give the coordinate of the gravity center of standard cells calculated by the average x and y coordinates for all standard cells. The idea is to expand the displacement reference position linearly w.r.t. the gravity center of standard cells. Here, ρ is a user-specified parameter, called *the expansion ratio*, which can be used to control the degree of expansion. Figure 3 shows example expansion ratios for placement with different utilization rates. For a design with a low utilization rate, a smaller expansion ratio is more suitable to free the space for the standard cells while the displacement minimization can still be preserved. For a design with a high utilization rate, in contrast, a larger expansion ratio will try to minimize the dead spaces among macros and ensure that the central region be maximized for the standard-cell placement. In our implementation, the expansion ratios, ρ's, are set to 1.05, 1.1, 1.2, and 1.3 for circuits with the utilization rates less than 70%, between 70% and 80%, between 80% and 90%, and larger than 90%, respectively.

At first glance, the main differences between our linear-programming formulation and that of [12] lie in Inequalities (2) and (3). As a matter of fact, our formulation/process has the following advantages over the work [12]:

- Instead of optimizing macro displacement alone, we introduce the concept of reference position expansion to expand the displacement reference position linearly w.r.t. the gravity center of standard cells. This scheme results in a better macro placement that preserves a continuous standard-cell placement region and thus leads to a better final placement.
- We construct the geometric constraints only on the reduction edges of TCG, instead of building one constraint edge between each pair of macros. Thus, the problem size and computation time of our linear programming can significantly be reduced without loss of solution quality.
- Our linear programming combined with adaptive SA can further optimize the macro orientations and thus obtain better macro positions.
- It will be clear in Section IV-A that our linear programming formulation achieves 6% shorter HPWL on average than that of [12] with comparable running time.

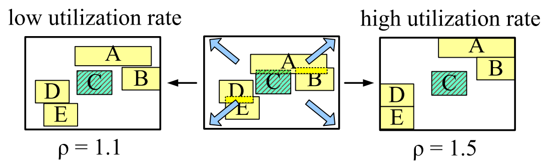

Fig. 3. Different expansion ratios for placements with different utilization rates.

C. Macro Placement Evaluation

To evaluate the quality of a macro placement, we define the cost function Φ of a macro placement P as follows:

$$\Phi(P) = \alpha D + \beta S + \gamma R + \delta I, \qquad (6)$$

where D is the total macro displacement, S is the minimum slack of a TCG, R is the placement region cost, I is the macro wirelength, and α, β, γ, and δ are user-specified weighting parameters. Here, α controls the degree of displacement; a higher α is recommended when the initial placement contains small overlaps. We usually use a higher β to make the infeasibility cost a dominating term so that it is easier to obtain a legal placement result. Since S always equals zero for feasible TCGs, the SA will be guided by the other three terms for the optimization. Since the effect of R is opposite to that of D, if a larger α is applied to minimize the displacement, γ is set to be smaller, and vice versa. In our implementation, we usually use a smaller δ since the macro wirelength is much smaller compared to those of standard cells. We have explained the computation of S in Section III-A and will explain those for D, R, and I in the following.

Cost D is the total macro displacement, defined as $D = \sum_{i=1}^{n}(|x'_i - x_i| + |y'_i - y_i|)^2$, where (x_i, y_i) is the initial position of the macro m_i, and (x'_i, y'_i) is the current position of macro m_i determined by the linear programming described in Section III-B. The quadratic penalty is used to prevent a large displacement for a macro.

The macro wirelength I is calculated by the half-perimeter wirelength (HPWL) of all nets connected to the macros. Since the standard-cell positions are not yet determined, we consider only the interconnections among macros at this stage. (As mentioned earlier, nevertheless, the wirelength among the macros and standard cells have been considered during the initial placement and implicitly handled by minimizing the macro displacement.) The cost R is used to evaluate the standard-cell placement region, and is computed by the two methods: (1) the overlapping area among macros and standard cells, and (2) the overlapping area among macros and a given ellipse used to approximate a continuous standard-cell placement region. Figure 4 illustrates the computation of R using exact standard-cell positions and an approximated ellipse. The underlying idea is to generate a macro placement that causes the minimum disturbance to the initial standard-cell placement (thus implicitly optimizing the wirelength by minimizing the disturbance), and preserve a better standard-cell placement region. We also impose a penalty of placing macros in the standard-cell congested regions.

To compute R, we first divide the placement region into k uniform bins. For each bin g_i, we compute the two values t_{m_i} and t_{s_i}, where t_{m_i} (t_{s_i}) is the area ratio of movable macros (standard cells) to the bin. The cost of the overlapping area equals $\sum_{i=1}^{k} t_{m_i} t_{s_i}$. Alternatively, we can find an ellipse that covers 80% of the standard-cell prototyping positions with foci f_1 and f_2, and the semimajor axis a. Then, t_{e_i} of the bin g_i is computed as follows:

$$t_{e_i} = \frac{2a}{dist(g_i, f_1) + dist(g_i, f_2)}, \tag{7}$$

where $dist(g_i, f_1)$ ($dist(g_i, f_1)$) denotes the distance between the center of the bin g_i and the focus f_1 (f_1). This formulation gives the central region a higher weight. Then, the cost R is defined as $R = \sum_{i=1}^{k} t_{m_i}(t_{s_i} + t_{e_i})$. This equation sums up the costs of the overlapping areas for all bins. In Figure 4, darker bins imply higher costs.

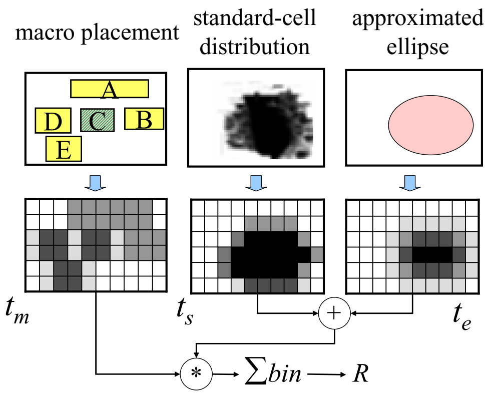

Fig. 4. Computing the standard-cell placement region cost. Macro placement, standard-cell placement, and an approximated ellipse are first quantified by the bin structure. For each bin, t_s and t_e are summed up and then multiplied by t_m. Finally, the cost R is calculated by summing up the product of each bin.

D. The Macro Placement Flow

Figure 5 shows the flow of our macro placer with *adaptive SA*. Given the circuit information, we construct an initial TCG. Adaptive SA is then used to find the desired macro placement. We adopt the following four operations defined in [14], [15] to perturb a TCG: (1) rotation: rotate a module, (2) swap: swap two nodes in both of C_h and C_v, (3) reverse: reverse a reduction edge in C_h or C_v, and (4) move: move a reduction edge from one TCG to another. By applying these operations and the cost function proposed in Section III-C, the geometric relations and orientations of macros can be optimized.

We apply two methods to evaluate a new solution. In traditional floorplanning, packing is applied to determine the placement result. In our macro placement, however, the macro positions are determined by solving a linear program, which typically leads to a better solution, but has a much higher time complexity, compared to the traditional packing. Therefore, we resort to an adaptive scheme for our SA: When the solution quality is poor (e.g., the TCG is infeasible or the macro displacement is large), we determine the macro positions by corner packing. The corner packing procedure places macros around the corners of the placement region using the following equations:

$$\begin{aligned}
x'_i &= l_{v_i} \quad && \forall\, i, v_i \in C_h, l_{v_i} \leq \tfrac{\max(l_{v_i})}{2},\\
x'_i &= u_{v_i} \quad && \forall\, i, v_i \in C_h, l_{v_i} > \tfrac{\max(l_{v_i})}{2},\\
y'_i &= l_{v_i} \quad && \forall\, i, v_i \in C_v, l_{v_i} \leq \tfrac{\max(l_{v_i})}{2},\\
y'_i &= u_{v_i} \quad && \forall\, i, v_i \in C_v, l_{v_i} > \tfrac{\max(l_{v_i})}{2},
\end{aligned}$$

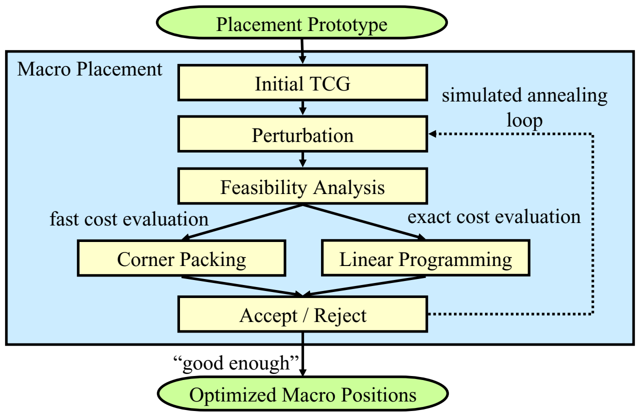

Fig. 5. Our adaptive SA flow, the second stage of the flow illustrated in Figure 1.

where l_{v_i} and u_{v_i} are given by the feasibility analysis described in Section III-A.

To smooth the cost evaluation, we propose a cost-smoothing technique to reduce the gap between the two evaluation methods, linear programming and corner packing. Let Ψ_l denote the cost evaluated by linear programming and Ψ_c be the cost derived from corner packing. We introduce a state variable r to scale the two costs, Ψ_l and Ψ_c. The smoothed cost Ψ_s is defined as follows:

$$\Psi_s = r\Psi_c + (r_m - r)\Psi_l, \tag{8}$$

where r_m is the maximum number of states. In our experiments, r_m equals 5. The variable r is updated in each iteration. If the current solution quality meets the criteria of applying linear programming, r is decreased by one. Otherwise, r is increased by one. The value of r is limited between r_m and zero. Both corner packing and linear programming are performed when $r_m < r < 0$. Figure 6 illustrates the determination of the state variable r. Initially, the cost is evaluated by corner packing, and the variable r is set to r_m. Then, while changing the evaluation from corner packing to linear programming, the cost evaluation method is also changed gradually by reducing the weight of corner packing and increasing the weight of linear programming. As shown in Figure 6, after r_m iterations of the first change of the evaluation, the evaluation method is totally switched to linear programming, and the corner packing process stops. This cost-smoothing technique prevents the cost from dramatic changes as the evaluation method switches and helps the SA obtain a better solution.

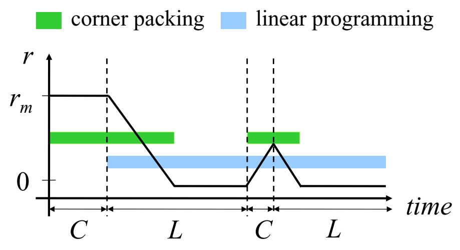

Fig. 6. An example of changing the variable r. The colored bars indicate the duration when corner packing and linear programming are performed. "C (L)" denotes the duration when the solution quality is suitable for corner packing (linear programming).

After evaluating the macro placement, we decide whether we should accept the new solution or not, according to the quality of the macro placement and the current temperature. The SA continues until the predefined termination condition is reached.

IV. Experimental Results

To test our macro placer, we conducted three experiments based on the ISPD'06 placement contest benchmarks [1]. We changed all fixed macros in the ISPD'06 placement benchmarks to movable ones to test the macro placement algorithms. All the experiments on the ISPD'06 placement contest benchmarks were performed on an AMD Opteron 1.8GHz machine with 8 GB memory. The linear programming solver applied here was lp_solve 5.5.

We applied three state-of-the-art publicly available academic mixed-size placers, NTUplace3 [7], [8] mPL6 [5], and Capo 10.2 [18], where NTUplace3 and mPL6 are two leading analytical placers while Capo 10.2 is a partitioning-based placer. We also compared our macro placer with the leading macro placer, MP-tree [9], [10], and the macro legalizer, XDP [12].

We conducted three experiments. For the first experiment, we compared the effectiveness and efficiency of different macro placement algorithms by integrating our macro placer, MP-tree, and XDP into NTUplace3, separately. For the second experiment, we studied the effects of different chip utilization rates on those macro placement algorithms. For the third experiment, we further combined our macro placer with mPL6 and Capo 10.2 to evaluate our macro placement algorithm with different placers.

A. Comparisons among Academic Macro Placers

In this experiment, we compared the resulting HPWLs and the total runtimes of various placers based on the ISPD'06 placement benchmarks with the default utilization rates. Every tested macro placer took the global placement results of NTUplace3 as its initial placement, and then tried to generate legal macro placement results. All macros were fixed after macro placement, and standard cells were then placed by NTUplace3. Table II summarizes the experimental results. Note that the CPU times reported here contain both the runtimes of macro placement and standard-cell placement.

As shown in Table II, our macro placer combined with NTUplace3 obtains the smallest wirelength on average. Compared with the results of NTUplace3 alone, our macro placer reduces the average HPWL by 15%. Compared with MP-tree and XDP integrated with NTUplace3, our macro placer integrated with NTUplace3 improves the average HPWL by 11% and 6%, respectively. As shown in Table II, NTUplace3 combined with our macro placer is about three times faster than NTUplace3 alone, and achieves 58% shorter runtime compared with MP-tree combined with NTUplace3.

B. Effects of Chip Utilization Rates

In this experiment, we compared the effects with different chip utilization rates. We modified the core region of the ISPD'06 benchmarks to obtain the three utilization rates, 70%, 80%, and 90%. Table III shows the resulting HPWLs with different chip utilization rates.

Integrating our macro placer with NTUplace3, we can obtain legal placements for all the benchmarks. In contrast, NTUplace3 alone cannot generate legal placements for newblue3 with all utilization rates, and XDP integrated with NTUplace3 also fails for the same circuit under the 90% utilization rate. Compared with NTUplace3 alone on those legalized benchmarks, our macro placer integrated with NTUplace3 results in 4%, 4%, and 14% better average HPWL for 70%, 80%, and 90% utilization rates, respectively. Further, while the utilization rate decreases from 90% to 70%, our placer combined with NTUplace3 outperforms that of MP-tree by larger margins (from 0% up to 9%), implying that our macro placer is robust for designs with various utilization rates. In contrast, the solution quality of XDP combined with NTUplace3 degrades as the utilization rate increases.

C. Integration with Other Placers

In addition to NTUplace3, we also integrated our macro placer with mPL6 and Capo 10.2, which are based on the analytical and min-cut placement techniques, respectively. Again, we tested the ISPD'06 placement benchmarks with the default chip utilization rates. Our macro placer took the global placement results of mPL6 and Capo as initial solutions, and then optimize macro positions and orientations. Finally, the macro positions were fixed, and mPL6 and Capo were again applied to place the standard cells.

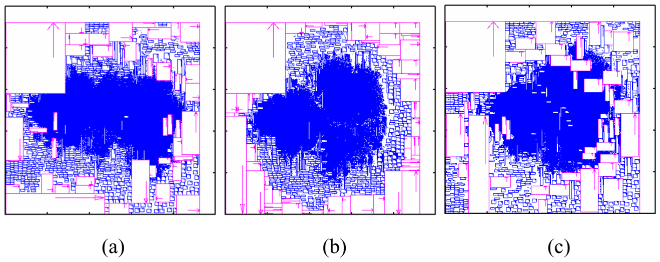
Fig. 7. The placement result for adaptec5 with the 80% chip utilization rate. (a) Our macro placer can determine non-compacted macro positions and optimize the placement region. (b) MP-tree tends to pack macros along the chip boundaries. (c) XDP does not optimize the placement region, and thus some macros stay in the chip center.

Table IV shows the results without and with our macro placer. As shown in the table, mPL6 failed to find legal solution in all circuits. However, integrated with our macro placer, mPL6 can obtain legal solutions for all circuits, and the resulting HPWLs are further reduced by 6% for the circuits other than newblue3 and newblue7. For Capo, we report the wirelengths and runtimes after the global placement and legalization stages (without detailed placement), since the detailed placement of Capo for the modified benchmarks cannot function correctly for some unknown reason. As shown in Table IV, Capo is robust in finding legal placements since macro positions are guaranteed to be overlap-free during the global placement. However, the quality is not good. Integrated with our macro placer, Capo reduced 5% average HPWL than that without our macro placer. The results show that our macro placer is very flexible and robust to be effectively integrated into various placers (mPL6, Capo, and NTUplace3 as well).

V. Conclusions

We have proposed a novel constraint graph-based macro placement algorithm to remove macro overlaps and optimize macro positions/orientations effectively and efficiently. Experimental results have shown that the proposed algorithm can further improve the quality of state-of-the-art academic placers. Unlike the previous MP-tree work that focuses only on high-density designs, further, our macro placer is robust and can consistently provide high-quality macro placement solutions for various utilization rates.

Acknowledgments

This work was partially supported by Cadence Design Systems, Inc., Etron, SpringSoft, TSMC, and National Science Council of Taiwan under Grant No's NSC 96-2628-E-002-248-MY3, NSC 96-2628-E-002-249-MY3, NSC 96-2221-E-002-245, and NSC 96-2752-E-002-008-PAE.

References

[1] *ISPD 2006 Program.* http://www.ispd.cc/program.html.

[2] S. N. Adya and I. L. Markov. Combinatorial techniques for mixed-size placement. *ACM TODAES*, 10(5), pages 58–90, January 2005.

[3] E. Wein and J. Benkoski. Hard macros will revolutionize SoC design. *EETimes Online*, August 2004.

[4] A. R. Agnihotri, S. Ono, and P. H. Madden. Recursive bisection placement: Feng Shui 5.0 implementation details. In *Proc. ISPD*, pages 230–232, April 2005.

[5] T. Chan, J. Cong, J. Shinnerl, K. Sze, and M. Xie. mPL6: Enhanced multilevel mixed-size placement. In *Proc. ISPD*, pages 212–214, April 2006.

[6] C.-C. Chang, J. Cong, and X. Yuan. Multi-level placement for large-scale mixed-size IC designs. In *Proc. ASP-DAC*, pages 325–330, January 2003.

[7] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. A high quality analytical placer considering preplaced blocks and density constraint. In *Proc. ICCAD*, pages 187–192, November 2006.

[8] —. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE TCAD*, 27(7), pages 1228–1240, July 2008.

TABLE II
THE RESULTING HPWLS AND CPU TIMES OF OUR MACRO PLACER WITH NTUPLACE3 ("OURS"), NTUPLACE3 ALONE ("W/O"), MP-TREE WITH NTUPLACE3 ("MPT"), AND XDP WITH NTUPLACE3 ("XDP") USING THE DEFAULT UTILIZATION RATES.

	HPWL (× e7)				CPU (sec)			
Circuit	Ours	w/o	MPT	XDP	Ours	w/o	MPT	XDP
adaptec5	29.46	29.03	31.01	31.08	6847	5165	2621	4253
newblue1	6.23	6.06	6.50	6.32	1364	1423	1479	1278
newblue2	18.89	28.09	22.60	18.90	1837	4461	5710	2172
newblue3	30.18	53.48	37.57	37.64	2572	40772	11134	2315
newblue4	21.38	22.83	23.77	22.01	9208	3685	2691	3036
newblue5	42.92	39.91	43.71	45.41	6379	12161	6452	8425
newblue6	44.93	44.24	50.50	46.43	5313	7153	8856	5432
newblue7	99.03	100.06	108.06	102.21	22030	26253	64878	35686
Comparison	1.00	1.15	1.11	1.06	1.00	3.25	1.58	0.99

TABLE III
THE RESULTING HPWLS (× E7) FOR DIFFERENT CHIP UTILIZATION RATES OF OUR MACRO PLACER WITH NTUPLACE3 ("OURS"), NTUPLACE3 ALONE ("W/O"), MP-TREE WITH NTUPLACE3 ("MPT"), AND XDP WITH NTUPLACE3 ("XDP"). NR: NO LEGAL RESULTS CAN BE OBTAINED.

	utilization 70%				utilization 80%				utilization 90%			
Circuit	Ours	w/o	MPT	XDP	Ours	w/o	MPT	XDP	Ours	w/o	MPT	XDP
adaptec5	30.00	29.83	32.83	30.37	29.72	31.02	32.72	31.48	29.15	56.84	30.48	34.79
newblue1	6.09	5.93	6.56	5.95	6.12	6.11	6.39	6.14	6.18	6.48	6.38	6.31
newblue2	18.22	22.10	20.51	22.63	18.28	20.50	22.75	21.42	18.79	20.78	19.29	22.64
newblue3	35.09	NR	32.17	36.28	30.05	NR	32.61	43.99	26.57	NR	29.64	NR
newblue4	21.78	22.35	24.86	22.76	20.95	22.10	22.52	21.88	23.11	23.97	22.68	28.92
newblue5	40.44	42.94	46.51	42.29	39.15	39.46	44.61	48.18	54.74	44.79	47.97	53.94
newblue6	43.22	43.34	49.33	46.54	44.96	46.94	49.24	47.12	52.64	67.80	47.59	52.20
newblue7	99.87	101.23	107.90	102.68	96.33	98.48	107.62	101.28	148.56	111.08	152.26	145.61
Comparison	1.00	(1.04)	1.09	1.06	1.00	(1.04)	1.11	1.13	1.00	(1.14)	1.00	(1.09)

TABLE IV
THE RESULTING HPWLS AND CPU TIMES FOR DIFFERENT PLACERS WITH ("W. OURS ") AND WITHOUT ("W/O") OUR MACRO PLACER USING DEFAULT UTILIZATION RATES. NOTE THAT "HPWL" DENOTES THE WIRELENGTH AFTER THE WHOLE PLACEMENT PROCESS WHILE "G+L HPWL" GIVES THE WIRELENGTH AFTER THE GLOBAL PLACEMENT AND LEGALIZATION STAGES WITHOUT RUNNING DETAILED PLACEMENT.

	mPL6				Capo 10.2			
	HPWL (× e7)		CPU (sec)		G+L HPWL (× e7)		CPU (sec)	
Circuit	w. Ours	w/o	w. Ours	w/o	w. Ours	w/o	w. Ours	w/o
adaptec5	26.08	28.48	13150	12847	34.59	32.30	42486	48628
newblue1	5.78	6.32	4211	4799	8.10	9.67	13267	9042
newblue2	19.11	18.09	8372	10622	26.46	29.79	17133	21781
newblue3	23.72	NR	13547	NR	43.38	54.18	24645	40106
newblue4	19.35	20.75	8804	14009	25.71	25.17	19671	19603
newblue5	35.95	39.59	11126	29253	47.81	45.42	58587	58779
newblue6	40.14	41.84	30601	20769	49.46	49.09	84245	53960
newblue7	91.46	NR	65374	NR	110.35	109.45	190773	135416
Comp.	1.00	(1.06)	1.00	(1.16)	1.00	1.05	1.00	1.01

[9] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Liu, and D. Liu. MP-trees: A packing-based macro placement algorithm for mixed-size designs. In *Proc. DAC*, pages 447–452, June 2007.

[10] —. MP-trees: A packing-based macro placement algorithm for modern mixed-size designs. *IEEE TCAD*, 27(9), September 2008.

[11] J. Cong, M. Romesis, and J. R. Shinnerl. Fast floorplanning by look-ahead enabled recursive bipartitioning. In *Proc. ASPDAC*, pages 1119–1122, January 2005.

[12] J. Cong and M. Xie. A robust detailed placement for mixed-size ic designs. In *Proc. ASP-DAC*, pages 188–194, January 2006.

[13] A. B. Kahng and Q. Wang. A faster implementation of APlace. In *Proc. ISPD*, pages 218–220, April 2006.

[14] J.-M. Lin and Y.-W. Chang. TCG: A transitive closure graph-based representation for non-slicing floorplans. In *Proc. DAC*, pages 764–769, June 2001.

[15] —. TCG: A transitive closure graph based representation for general floorplans. *IEEE Trans. VLSI Systems*, pages 288–292, February 2005.

[16] M. D. Moffitt, A. N. Ng, I. L. Markov, and M. E. Pollack. Constraint-driven floorplan repair. In *Proc. DAC*, pages 1103–1108, July 2006.

[17] B. Obermeier, H. Ranke, and F. M. Johannes. Kraftwerk: a versatile placement approach. In *Proc. ISPD*, pages 242–244, April 2005.

[18] J. Roy, D. Papa, A. Ng, and I. Markov. Satisfying whitespace requirements in top-down placement. In *Proc. ISPD*, pages 206–208, April 2006.

[19] T. Taghavi, X. Yang, B.-K. Choi, M. Wang, and M. Sarrafzadeh. Dragon2006: Blockage-aware congestion-controlling mixed-size placer. In *Proc. ISPD*, pages 209–211, April 2006.