Fast and Effective Placement Refinement for Routability

Yanheng Zhang and Chris Chu

***Abstract*—In this brief, we propose congestion refinement of placement (CROP) for improving the congestion of mixed-size placement solutions. CROP consists of a congestion-driven module shifting technique and a congestion-driven detailed placement (CDDP) technique. The shifting technique is proposed for better allocation of routing resources. We shift modules based on the shifting of G-cell boundaries. Shifting in each direction can be formulated as a linear program (LP) for resizing each cell in the global routing grid (i.e., G-cell). We degenerate and solve the LP by a very efficient longest path computation. Then the CDDP technique is proposed for distributing the routing demands better. Congestion reduction is realized by weighting the half-perimeter wirelength with the congestion factor during detailed placement. Theoretically, our tool is capable of handling most mixed-size placement benchmarks with movable and/or fixed macro (FM) blocks. In order to better analyze its performance, the ISPD-GR benchmark suite (ISPD05/06 derived global routing benchmarks) with FM modes is developed. The experimental results show that CROP effectively alleviates congestion for unroutable placement solutions in short runtimes for different placers.**

***Index Terms*—Congestion, physical design, placement, routing.**

I. Introduction

The success of routing is critical in VLSI design flow. With the ever-decreasing feature size, the routability issue has become more and more complicated. Nowadays, the mixed-size system-on-chip contains up to millions of standard cells and thousands of big macros in one design.

In a typical ASIC design flow, routing and placement are independent stages. In the placement stage, the design is optimized for half perimeter wirelength (HPWL). The congestion and routing shapes are handled in the following routing stage. However, the HPWL optimization during the placement stage may lead to a hard-to-route or even unroutable solution. It is desirable to integrate the routability factor into the placement. Placement is a more flexible stage for improving routability. The shifting and relocating of modules could effectively alleviate congestion. Therefore, congestion-driven placement techniques have received much attention from academia in recent years.

There have been many works proposed for routability-driven placement. In general, previous techniques can be categorized into four groups. The first group formulates and incorporates the routability component into a placement optimizing objective. Spindler and Johannes [1] proposed the RUDY congestion estimation technique and modified the density term to incorporate both the routing density and module density. In [2], Jiang *et al*. applied Lagrangian relaxation to soften the routability constraints. Similarly, Tosta *et al*. [3] integrated the wire density term into the analytical placement framework. The second group applies the implicit or explicit white space allocation (WSA) technique inside or after the placement flow. Yang *et al*. [4] proposed three WSA methods and integrated one of them in the detailed placement (DP) flow of Dragon. mPL-R with WSA [5] distributed the white space by adjusting the cut-lines of hierarchically sliced placement based on the available white space and congestion. In [6], the authors proposed inflating the cells inside the congested region, which is an implicit method for allocating white space. The third group guides placement by global routing. IPR [7] integrated FastRoute2.0 [8] into FastPlace [9] and performed full global routing to guide the placement flow. The fourth group mixes some of the three features above. For instance, ROOSTER [10] proposed to both optimize RSMT in their global placement objective and apply WSA in their DP flow.

Manuscript received November 10, 2011; revised May 16, 2012; accepted July 21, 2012.

The authors are with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50010 USA (e-mail: yanhengzhang@gmail.com; cnchu@iastate.edu).

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

Digital Object Identifier 10.1109/TVLSI.2012.2214408

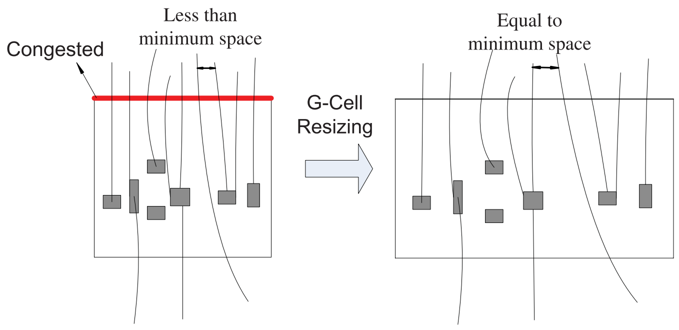

Fig. 1. Basic idea of CDMS.

In this brief, we propose a fast and effective mixed-size placement refinement tool called congestion refinement of placement (CROP) for routability improvement. CROP consists of a congestion-driven module shifting (CDMS) technique and a congestion-driven detailed placement (CDDP) technique. Both techniques are guided by congestion information obtained by global routing. The first technique works by adjusting the boundary of each G-cell and shifting the modules accordingly. Fig. 1 illustrates the basic idea.

CROP is a fast and effective refinement tool for mixed-size placement solution with several nice properties.

1) CROP is independent of any placer. It can be easily integrated into various placement tools.
2) CROP shows good performance of improving routability. For example, CROP is more precise when shifting modules based on the boundary of each G-cell. Moreover, our congestion shifting model differentiates the vertical and horizontal directions.
3) CROP runs very fast. The runtime overhead is negligible compared to the original placement runtime.

In summary, our technical contributions include the introduction of the following:

1) a placement routability refinement flow which is independent of any placer or router;
2) a better refined and directional module shifting model;
3) a longest path computation method facilitating fast runtime of module shifting;
4) a congestion-driven global swap technique by weighting HPWL with congestion.

We apply CROP to refine placement solutions generated by various placers: FastPlace 3.1 [11], NTUplace3 [12], mPL6 [13], and R-NTUplace3 [2]. We set up the ISPD-GR benchmarks to verify the performance of CROP. The results reveal that CROP effectively reduces congestion within a very short runtime.

The rest of this brief is organized as follows. Section II presents the preliminaries of global routing and an overview of our tool. Section III introduces the CDMS technique. Section IV explains CDDP technique. In Section V, we compare the results on the ISPD-GR benchmarks. Conclusions are drawn in Section VI.

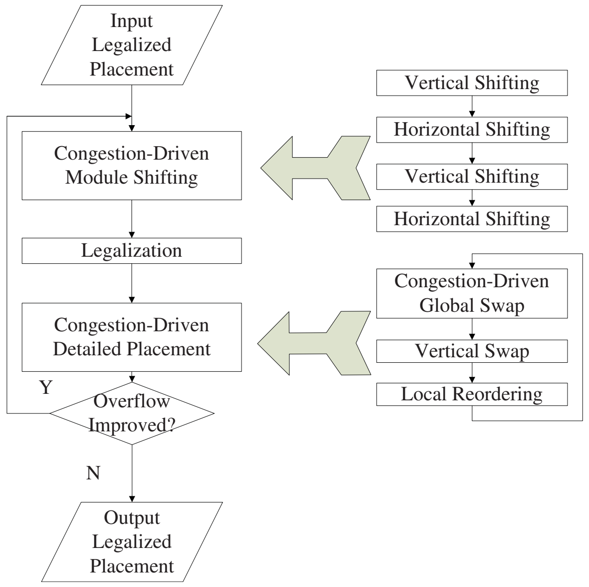
Input Legalized Placement
Congestion-Driven Module Shifting
Legalization
Congestion-Driven Detailed Placement
Overflow Improved?
Y
N
Output Legalized Placement
Vertical Shifting
Horizontal Shifting
Vertical Shifting
Horizontal Shifting
Congestion-Driven Global Swap
Vertical Swap
Local Reordering

Fig. 2. Algorithm flow.

II. Preliminaries and Overview

A. Preliminaries on Global Routing

The placement region is partitioned into a set of G-cells to perform global routing. In the global routing grid graph, each G-cell is represented by a node and each G-cell boundary is abstracted as the global routing edge. If the usage U_e is over the capacity C_e for any edge e, the overflow is calculated as $O_e = U_e - C_e$.

B. CROP Flow

The flow of CROP [14] is illustrated in Fig. 2. The input is any legalized placement design. The first step is CDMS. Modules are shifted in X and Y directions iteratively. After a number of rounds (two rounds in the flow), the post-shifting placement is legalized. Then CDDP is called to compensate the wirelength loss and further improve the routability. The shifting–legalization–DP procedure is repeatedly called until the solution stops improving. Global routing is applied when design modules are relocated. We use FastRoute 4.0 [15], [16] to perform global routing.

III. CDMS

A. Resizing G-Cells by Linear Programming

We formulate the problem of resizing G-cells to accommodate routing demands into a linear program (LP). We first assume that each module (M_k) is a standard cell (with smaller area than the G-cell).

We first partition the placement region into $m \times n$ G-cells. Let $B_{i,j}$ represents each G-cell, where i ($i \in \{1, \ldots, m\}$) denotes the row and j ($j \in \{1, \ldots, n\}$) denotes the column. $x_{i,j}$ and $x_{i,j+1}$ denote the x-coordinate for the left boundary and the right boundary, respectively. Likewise, $y_{i,j}$ and $y_{i+1,j}$ denote the y-coordinate for the bottom boundary and the top boundary, respectively. There are $m \times (n+1)$ x-variables and $(m+1) \times n$ y-variables. We use $u^l_{i,j}$, $u^r_{i,j}$, $u^b_{i,j}$, and $u^t_{i,j}$ to represent the global routing usage through left, right, bottom, and top boundary of $B_{i,j}$. H, W, hTile, and wTile represent the height of placement area, width of placement area, defined height, and width of the G-cell.

Without loss of generality, we investigate the horizontal shifting of vertical boundaries. Similar formulation can be derived for the vertical shifting case

$$\max : \sigma$$

s.t.

$$x_{i,j+1} - x_{i,j} \geq \sigma \times \text{MAX}\left(f^{-1}\left(u^b_{i,j}\right), f^{-1}\left(u^t_{i,j}\right)\right) \quad (1)$$

$$0 \leq \sigma \leq 1 \quad (2)$$

$$x_{i,j+1} - x_{i,j} \geq \frac{\sum_{k \in B_{i,j}} \text{area}(M_k)}{h\text{Tile}} \quad (3)$$

$$|x_{i,j} - x_{i+1,j}| \leq C \quad (4)$$

$$0 \leq x_{i,1} \quad (5)$$

$$x_{i,n+1} \leq W. \quad (6)$$

Next, we briefly explain the LP we formulated.

1) *Routability Constraints (1) and (2):* As shown in Fig. 1, the routing capacity is proportional to the length of the G-cell boundary. This constraint designates the G-cell width ($x_{i,j+1} - x_{i,j}$) to be proportional to maximum wiring demands of top and bottom sides. The function $f^{-1}(u)$ computes the sufficient width for wiring demand u. If the placement solution is very congested, the routability constraints may be too hard to satisfy. We introduce a variable σ to relax the routability constraints, which can be viewed as a scaling factor over the original constraint. The value of σ is bounded between 0 and 1.
2) *G-Cell Area Constraints (3):* These constraints ensure that each G-cell has enough space to hold the modules inside.
3) *Movement Constraints (4):* We introduce movement constraints to preserve the original placement solution by restricting movement of adjacent G-cell boundaries. In the equations, C is a constant denoting the degree of flexibility of moving adjacent G-cell boundaries. Because of the absolute sign, it can be expanded as follows:

$$x_{i+1,j} - x_{i,j} \geq -C \quad \forall i, j \quad (7)$$

$$x_{i,j} - x_{i+1,j} \geq -C \quad \forall i, j. \quad (8)$$

4) *Placement Region Constraints (5) and (6):* We need to ensure that all the boundaries are within the placement region ([0, W]). Note that the other constraints (e.g., G-cell area constraints) implicitly guarantee that $x_{i,j} \leq x_{i,j+1}$ for $j \in \{1, \ldots, n\}$.

B. Longest Path-Based Solution

When σ is fixed, the LP becomes a feasibility check problem (only contains constraints). The strategy consists in applying an outer loop which keeps decreasing σ until the LP is feasible. For each iteration (σ), we check feasibility by longest path computation.

When σ is fixed, we introduce G-cell boundary graph, or B-graph $G(V, E)$. Each vertex $v_{i,j} \in V$ represents a G-cell boundary. Each difference constraint in the form $x_d - x_s \geq Q$ is represented by a directed edge $e \in E$ pointing from v_s to v_d with a cost $\|e\|$ of Q. E_r, E_a, and E_m are the sets of edges incurred by routability constraints, G-cell area constraints, and movement constraints, respectively. Fig. 3 illustrates an example of B-graph with the three types of edges. The longest path distance to $v_{i,j}$ from the vertices associated with the leftmost boundaries of the placement region is the minimum value of $x_{i,j}$ that satisfies (1), (3), (5), (7), and (8). So the feasibility of the constraints can be determined by checking whether $x_{j,n+1} \leq W$ for all i (6).

We observe that the proposed B-graph contains directed cycles that are caused by movement edges (E_m). As suggested by [17], it is NP-complete to find the longest path for a graph with directed cycles.

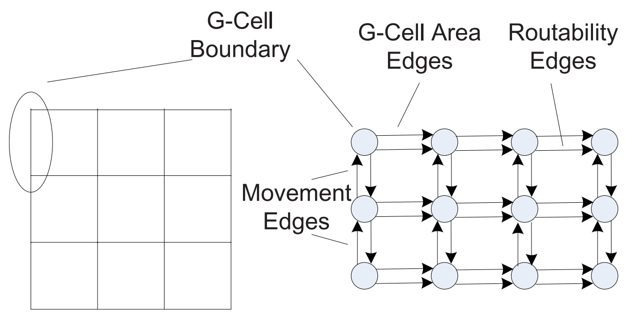

Fig. 3. Converting the G-cell boundary into B-graph.

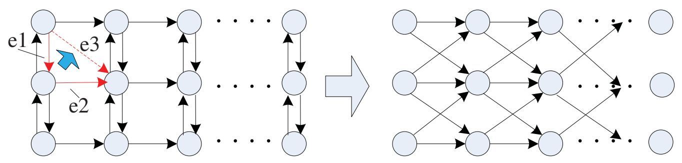

Fig. 4. Replacing movement edges with diagonal edges to facilitate longest path computation.

We need to convert the graph to an acyclic graph. As illustrated in Fig. 4, we merge the perpendicular edges ($e1$ and $e2$) and replace them with the diagonal edge ($e3$). The cost of a diagonal edge is the total cost of the perpendicular edges ($\|e3\| = \|e1\| + \|e2\|$).

Initially, σ is set to 1. If the resulting longest path length $L_p = \text{MAX}_{i=1,\ldots,m}(x_{i,n+1})$ is larger than W, we reduce σ to scale the current longest path into placement region. Suppose L represents set of edges along the longest path. We define the edges that cannot be scaled as hard edges and those scalable as soft edges. For instance, routability edges can always be reduced in magnitude and they are thus soft edges. Area edges are hard edges due to their nonscalability. Diagonal edges (E_d) can be either hard or soft edges depending on whether hard or soft edges dominate the edge weight. We divide the edges along the longest path L into two parts: 1) hard edge $E_h^L = (E_a \cup E_d^h) \cap L$ and 2) soft edge $E_s^L = (E_r \cup E_d^s) \cap L$. $L_h = \sum_{e \in E_L^h} \|e\|$ and $L_s = \sum_{e \in E_L^s} \|e\|$. To scale L_p inside fixed outline W, we have $s \times L_s + L_h = W$. Therefore $s = (W - L_h)/L_s$. Each iteration σ will be scaled by a scaling factor s to configure the soft edges into fixed outline. But we may not be able to compact all paths into a fixed outline at the same time. First, other paths may still be longer than W even after scaling. Second, the current path may not be scaled correspondingly based on s. Hence the scaling in the outer loop will be performed iteratively until all the paths fit into the fixed outline ($L_p = W$). The algorithm terminates in at most m iterations because at least one more x_i ($i \in \{1, \ldots, n+1\}$) will become less than or equal to W in each iteration.

We hold (5) as the precondition and use (6) as the feasibility check condition. The solution (G-cell boundary coordinates) will be aligned to the left side in the case when longest path is out of the fixed outline ([0, W]). And the cells will be packed to the left side. To avoid this phenomenon, we solve the counterpart LP which holds (6) as the precondition and use (5) as the feasibility check condition. It makes the solution packed to the right. We obtain two sets of solutions for each G-Cell boundary, say, $x_{i,j}^l$ and $x_{i,j}^r$. The two sets of solutions actually define the valid range of the boundary location. Let $X_{i,j}$ denotes the original G-cell boundary coordinate ($X_{i,j} = (j-1) \times w\text{Tile}$). If $X_{i,j}$ is within the valid range, ($x_{i,j} = X_{i,j}$). Otherwise, we move the boundaries to the closest of either $x_{i,j}^l$ or $x_{i,j}^r$.

After adjusting each G-cell boundary, the modules inside each G-cell will be shifted accordingly. CROP updates the module location by maintaining the ratio of distance to both boundaries before and after G-cell resizing.

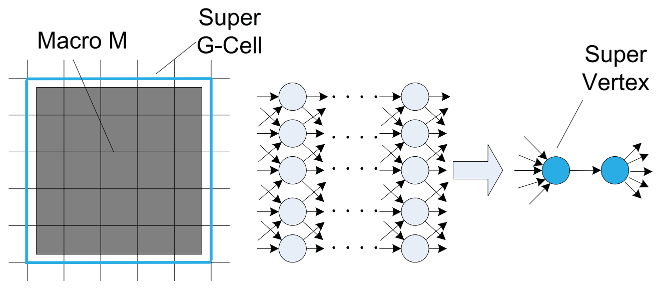

Fig. 5. Merging of G-cells for macro blocks.

C. Macro Block Handling

1) Movable Macros: In case the macros are movable, they can be legally relocated to the place of better routability. We apply a methodology similar to that we use to handle standard cells. We merge the covered G-cells to become a super G-cell. A macro will thus be repositioned based on the super G-cell boundary coordinates. As shown in Fig. 5, CROP merges the G-cells that are covered or partially covered by the big macro.

2) Fixed Macros: Certain IP cores and FMs are preplaced on chip, considering area, power, and timing issues. In analog designs, there are design constraints that need be respected for macros, such as mirroring constraints (e.g., two blocks are the mutually viewed images for a fixed axis), alignment constraint, distance constraint, etc. It is therefore not allowed to move the preplaced big macros, as they may degrade many design specifications and violate design constraints.

When there are FMs in the design, we need to make sure the corresponding super G-cell boundaries unmoved during G-cell resizing. We mark the super vertices as fixed by a Boolean variable and record its original position. We monitor the fixed vertices along the longest path. If any vertice is found to be fixed, we scale the path accordingly for preventing the violation of this constraint. Note that we keep the uniform scaling factor over the entire algorithm, which still seeks the solution to the LP problem we originally proposed.

IV. CDDP

DP is commonly applied after global placement to improve HPWL for legalized placement solution. We develop a congestion-driven DP technique to retrieve the netlength during the shifting stage and to further improve the routability. The flow of our proposed DP is shown in Fig. 2, which contains congestion-driven global swap, vertical swap, and local reordering. The whole DP flow is based on FastDP [18].

Global swap step seeks to swap modules for improving HPWL based on a greedy pairwise position exchange. In CROP, the swapping evaluation function incorporates the congestion component; in other words, the HPWL is weighted by the congestion factor of α_n for net n. Simply put

$$rHPWL_n = HPWL_n \times \alpha_n \tag{9}$$

where α_n is computed by averaging congestion of all possible Z paths inside the bounding box

$$\alpha_n = \frac{w_{\text{tot}}}{E} = \frac{\sum_{p \in P} \sum_{e \in p} w(e)}{E} \tag{10}$$

where P is the set of all Z routing paths. e represents one global routing edge along path p. E is the edge set, and $w(e)$ is the congestion cost of edge e. w_{tot} represents the sum of congestion cost. To save computational effort, instead of enumerating all Z paths, we propose a more efficient lookup table-based method to get the sum of total congestion cost inside the specified bounding box.

TABLE I
CROP RESULTS ON ISPD-GR BENCHMARKS AND FM MODE

Metrics	Tools		a1	a2	a3	a4	a5	b1	b2	b3	n1	n2	n3	n4	n5	n6
Routing overflow	FP3.1	w/o	/	1260	4	/	/	18 755	769	/	/	/	9642	/	46	/
		w	/	0	0	/	/	0	0	/	/	/	9019	/	0	/
		FM	/	0	0	/	/	0	293	/	/	/	8998	/	0	/
	NTUPlace3	w/o	2885	2369	1621	141	/	15 896	19 793	15 259	/	60	9442	3394	/	8
		w	0	0	0	0	/	0	0	0	/	0	8480	0	/	0
		FM	0	0	19	0	/	0	5928	10	/	0	8416	0	/	0
	mPL6	w/o	20	18 535	22 539	5703	7307	46 995	1736	4678	/	9	8835	5649	12475	4495
		w	0	11 289	660	0	0	0	0	0	/	0	8405	285	0	0
		FM	0	7465	12 225	0	28	0	0	0	/	0	8220	0	10 186	12
	R-NTUplace	w/o	51	2849	94	20	16	12 887	38 616	2264	/	898	10 065	385	/	/
		w	0	0	0	0	0	0	0	0	/	0	8551	0	/	/
		FM	0	0	6	0	0	0	11 670	0	/	0	8392	0	/	/
CROP CPU (second)	FP3.1	FM	/	92	200	/	/	73	193	/	/	/	297	/	406	/
			/	186	480	/	/	133	377	/	/	/	590	/	1056	/
	NTUplace3	FM	70	98	238	227	/	70	308	453	/	203	270	198	/	487
			169	196	605	446	/	122	416	977	/	456	455	366	/	945
	mPL6	FM	75	143	374	279	506	92	229	472	/	216	366	272	630	292
			142	213	491	559	967	165	457	887	/	554	774	466	1306	982
	R-NTUplace	FM	65	97	272	216	301	66	224	690	/	225	256	189	/	/
			121	168	456	435	765	111	394	1409	/	390	414	301	/	/
CROP Iterations	FP3.1	FM	/	2	2	/	/	2	2	/	/	/	2	/	2	/
			/	4	4	/	/	4	4	/	/	/	4	/	4	/
	NTUplace3	FM	2	2	2	2	/	2	3	2	/	2	2	2	/	2
			4	4	4	4	/	4	4	5	/	4	4	5	/	4
	mPL6	FM	2	2	3	2	3	2	3	2	/	2	2	2	3	2
			4	4	5	4	5	4	4	4	/	5	4	4	5	4
	R-NTUplace	FM	2	2	2	2	2	2	2	3	/	2	2	2	/	/
			4	4	5	4	4	3	5	4	/	3	4	4	/	/
Routing CPU (second)	FP3.1	w/o	/	280	223	/	/	1492	1042	/	/	/	13 331	/	154	/
		w	/	34	107	/	/	39	282	/	/	/	13 832	/	32	/
		FM	/	33	215	/	/	36	762	/	/	/	13 328	/	171	/
	NTUplace3	w/o	2031	311	1096	389	/	1945	2517	2692	/	362	12 726	1271	/	1088
		w	144	20	101	26	/	46	87	80	/	28	13 006	68	/	78
		FM	181	20	218	44	/	52	1298	445	/	33	13 114	88	/	729
	mPL6	w/o	420	1478	6795	2859	2927	1941	448	2397	/	131	13 661	2329	5756	6085
		w	58	974	1066	42	202	75	256	69	/	23	13 554	1581	200	155
		FM	63	588	2919	120	732	129	148	94	/	28	13 224	408	2697	964
	R-NTUplace	w/o	642	311	975	248	531	1619	3869	6439	/	933	12 801	515	/	/
		w	101	17	108	24	47	41	303	136	/	31	13 079	25	/	/
		FM	140	18	339	40	782	47	1713	1287	/	58	13 009	29	/	/
Routed wirelength ($\times 10e7$)	FP3.1	w/o	/	0.31	0.85	/	/	0.27	0.47	/	/	/	0.81	/	1.70	/
		w	/	0.31	0.85	/	/	0.26	0.48	/	/	/	0.84	/	1.65	/
		FM	/	0.31	0.89	/	/	0.26	0.47	/	/	/	0.83	/	1.73	/
	NTUplace3	w/o	0.30	0.30	0.84	0.71	/	0.28	0.47	0.83	/	0.46	0.74	0.81	/	0.95
		w	0.29	0.30	0.82	0.72	/	0.28	0.47	0.79	/	0.45	0.74	0.78	/	0.92
		FM	0.29	0.30	0.84	0.75	/	0.28	0.49	0.81	/	0.47	0.74	0.79	/	0.96
	mPL6	w/o	0.27	0.32	0.89	0.71	0.79	0.27	0.52	0.79	/	0.45	0.77	0.77	1.29	1.0
		w	0.26	0.32	0.85	0.72	0.77	0.26	0.54	0.77	/	0.45	0.77	0.78	1.40	0.96
		FM	0.26	0.32	0.91	0.75	0.80	0.28	0.52	0.79	/	0.46	0.76	0.77	1.34	1.07
	R-NTUplace	w/o	0.30	0.30	0.87	0.74	0.97	0.28	0.51	0.91	/	0.48	0.77	0.83	/	/
		w	0.29	0.30	0.83	0.73	0.86	0.27	0.49	0.84	/	0.46	0.76	0.79	/	/
		FM	0.29	0.30	0.87	0.74	0.96	0.27	0.52	0.92	/	0.48	0.75	0.80	/	/

V. EXPERIMENTAL RESULTS

All our experiments are performed on a machine with a 2.4-GHz AMD Opteron processor and 4G of memory. In order to better analyze the performance of CROP, we propose the ISPD-GR benchmarks. The benchmark suite is derived from ISPD05/06 [19], [20] placement contest benchmarks with all macro blocks movable.

The ISPD-GR benchmarks test routability of the global routing stage. In order to test the performance of CROP for test cases with FM, we fix all movable macros in ISPD-GR to become the FM version of ISPD-GR.

A. ISPD-GR Benchmarks

We show the experimental results on the ISPD-GR benchmarks derived from ISPD05/06 placement benchmarks. We utilize four publicly available academic placers to generate the initial legalized placement solution.

In particular, the initial legalized placement solutions are generated by FastPlace3.1 [11], NTUplace3 [12], mPL6 [13], and R-NTUplace [2]. In the experiment, FastRoute 4.0 [15] is utilized to report the global routing results.

Table I shows the results in detail. For each placer, we show the routing results before and after applying our tool. The entry with "/" means the original placement is routable, so we do not apply CROP. Before applying CROP, there are 6, 11, 13, and 11 unroutable cases for the solutions of FastPlace3.1, NTUplace3, mPL6 and R-NTUplace, respectively. The number is reduced to 1, 1, 4, and 1 with CROP. Out of these benchmarks, newblue3 is proved to be unroutable (more out pins than the capacity in a G-cell). From these results, we see that CROP is very effecitve in congestion reduction. We also report the CROP execution runtime in Table I. The runtime of our tool is trivial compared with the original placement runtime. Noticeably, the routing runtime is also saved considerably. We could achieve roughly $6\times$ speedup on average. The routing runtime improvement suggests the placement solution after applying CROP becomes easier to route. Additionally, the total wirelengths are 0.5% better, 1% better, 0.5% worse, and 5% better for FP3.1, NTUplace3, mPL6, and R-NTUplace, respectively. Generally speaking, the routed wirelength is similar to that of original design, which indicates the original placement solution is well maintained in CROP.

B. FM Solutions

In order to better evaluate the performance of placement benchmarks with FMs, we create the FM mode of the ISPD-GR benchmarks in which each macro is fixed. We conduct similar experiments as in Section V-A. In Table I, for each test case, the row marked with "FM" is the corresponding benchmark in the FM mode. We could observe that the congestion is consistently improving. For each experimental benchmark, the overflow is better than the input placement solution. Second, although the congestion is improving, the enhancement is usually less compared with the case with movable macros, and the required runtime is longer. But we can see that the original placement solution is well maintained.

VI. Conclusion

In this brief, we presented CROP to improve routability for placement solution as a refinement process. Our tool is independent of any placer or router. The main techniques involve CDMS and CDDP. ISPD-GR benchmarks with FM modes are utilized to demonstrate the efficiency and effectiveness of CROP. We propose to continue working to improve the performance of the tool and apply CROP to enhance routability for other design stages, such as detailed routing [21], [22].

References

[1] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Proc. Conf. Design, Autom. Test Eur.*, 2007, pp. 1226–1231.

[2] Z. Jiang, B. Su, and Y. Chang, "Routability-driven analytical placement by net overlapping removal for large-scale mixed-size designs," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 2008, pp. 167–172.

[3] K. Tsota, C. Koh, and V. Balakrishnan, "Guiding global placement with wire density," in *Proc. Int. Conf. Comput.-Aided Design*, 2008, pp. 212–217.

[4] X. Yang, B. Choi, and M. Sarrafzadeh, "Routability-driven white space allocation for fixed-die standard-cell placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 4, pp. 410–419, Apr. 2003.

[5] C. Li, M. Xie, C. Koh, J. Cong, and P. Madden, "Routability-driven placement and white space allocation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 5, pp. 167–172, May 2008.

[6] U. Brenner and A. Rohe, "An effective congestion-driven placement framework," in *Proc. ACM/SIGDA Int. Symp. Phys. Design*, 2002, pp. 6–11.

[7] M. Pan and C. Chu, "IPR: An integrated placement and routing algorithm," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 2007, pp. 59–62.

[8] M. Pan and C. Chu, "FastRoute 2.0: A high-quality and efficient global router," in *Proc. Asia South Pacific Design Autom. Conf.*, 2007, pp. 250–255.

[9] N. Viswanathan and C. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model," in *Proc. ACM/SIGDA Int. Symp. Phys. Design*, 2004, pp. 26–33.

[10] J. Roy and I. L. Markov, "Seeing the forest and the trees: Steiner wirelength optimization in placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 4, pp. 632–644, Apr. 2007.

[11] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *Proc. Asia South Pacific Design Autom. Conf.*, 2007, pp. 135–140.

[12] T. Chen, Z. Jiang, T. Hsu, H. Chen, and Y. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1228–1240, Jul. 2008.

[13] T. F. Chan, J. Cong, M. Romesis, J. R. Shinnerl, K. Sze, and M. Xie, "mPL6: A robust multilevel mixed-size placement engine," in *Proc. ACM/SIGDA Int. Symp. Phys. Design*, 2005, pp. 227–229.

[14] Y. Zhang and C. Chu, "CROP: Fast and effective congestion refinement of placement," in *Proc. Int. Conf. Comput.-Aided Design*, 2009, pp. 344–350.

[15] Y. Xu, Y. Zhang, and C. Chu, "FastRoute 4.0: Global router with efficient via minimization," in *Proc. Asia South Pacific Design Autom. Conf.*, 2009, pp. 576–581.

[16] Y. Zhang, Y. Xu, and C. Chu, "FastRoute 3.0: A fast and high quality global router based on virtual capacity," in *Proc. Int. Conf. Comput.-Aided Design*, 2008, pp. 344–349.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.

[18] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *Proc. Int. Conf. Comput.-Aided Design*, 2005, pp. 48–55.

[19] *ISPD05 Placement Contest Benchmarks*. (2005) [Online]. Available: http://www.sigda.org/ispd2005/contest.htm

[20] *ISPD06 Placement Contest Benchmarks*. (2006) [Online]. Available: http://www.sigda.org/ispd2006/contest.htm

[21] Y. Zhang and C. Chu, "RegularRoute: An efficient detailed router with regular routing patterns," in *Proc. ACM/SIGDA Int. Symp. Phys. Design*, 2011, pp. 45–52.

[22] Y. Zhang and C. Chu, "GDRouter: Interleaved global routing and detailed routing for ultimate routability," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 2012, pp. 597–602.