CROP: Fast and Effective Congestion Refinement of Placement *

Yanheng Zhang and Chris Chu
Electrical and Computer Engineering
Iowa State University
Ames, IA 50010
email: {zyh,cnchu}@iastate.edu

ABSTRACT

Modern circuits become harder to route with the ever decreasing design features. Previous routability-driven placement techniques are usually tightly coupled with the underlying placers. So usually they cannot be easily integrated into various placement tools. In this paper, we propose a tool called CROP (Congestion Refinement of Placement) for mixed-size placement solutions. CROP is independent of any placer. It takes a legalized placement solution and then relocates the modules to improve routability without significantly disturbing the original placement solution.

CROP interleaves a congestion-driven module shifting technique and a congestion-driven detailed placement technique. Basically the shifting technique targets at better allocating the routing resources. Shifting in each direction can be formulated as a linear program (LP) for resizing each G-Cell. Instead of solving the computationally expensive LP, we discover that the LP formulation could be relaxed and solved by a very efficient longest-path computation. Then the congestion-driven detailed placement technique is proposed to better distribute the routing demands. Congestion reduction is realized by weighting the HPWL with congestion coefficient during detailed placement.

The experimental results show that CROP is capable of effectively alleviating the congestion for unroutable placement solutions. We apply it to placement solutions generated by four different placers on the ISPD05/06 placement benchmarks [1] [2]. Within a very short runtime, CROP greatly improves the routability and saves execution time for the routing stage after refinement.

1. INTRODUCTION

The routing problem has become more difficult with the decreasing design features. Nowdays, the mixed-size SOC contains up to millions of standard cells and thousands of big macros in one single design. The existence of big macros and large problem size make the routability issue more and more challenging.

In traditional design flow, routing and placement are separated. In the placement stage, usually HPWL (Half Perimeter Wirelength) is set as primary objective for optimization. Nevertheless, such placement solution is very likely unroutable partially because the HPWL emphasis in the placement stage may not be a direct indicator for hardness of routing that follows. Hence, many people turn to develop congestion-driven placement techniques for probing better routability at placement stage.

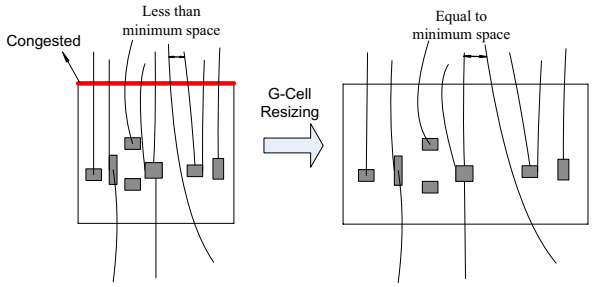

Figure 1: Basic idea of congestion-driven module shifting.

There have been many works proposed for routability-driven placement. In general, previous techniques could be categorized into four groups: the first group incorporates routability components into placement optimizing objective. Spindler and Johannes [3] proposed RUDY congestion estimation technique and modified the density term to contain both the routing density and module density. In [4], Jiang et al. applied Lagrangian relaxation to relax the routability constraints. Similarly, [5] integrated the wire density term into the analytical placement framework. The second group applies implicit or explicit White Space Alllocation (WSA) technique inside or after the placement flow. Yang et al. of [6] proposed three WSA methods and integrated one of them in the detailed placement flow of Dragon. mPL-R with WSA [7] distributes the white space by adjusting the cut-lines of hierarchically sliced placement based on the available white space and congestion. In [8], the authors proposed inflating the cells inside the congested region, which is an implicit manner for allocating white space. The third group guides placement by global routing. IPR [9] integrates FastRoute2.0 [10] into FastPlace [11] and performs full global routing to guide the placement flow. The fourth group mixes some of the above three features. For instance, ROOSTER [12] proposed to optimize RSMT in their global placement objective and apply WSA in their detailed placement flow.

In this work, we propose a fast and effective mixed-size placement refinement tool called CROP for routability improvement. CROP interleaves congestion-driven module shifting technique and congestion-driven detailed placement technique. Both techniques are guided by congestion information obtained by global routing. The congestion-driven shifting technique targets at better allocating the routing resources. It is achieved by adjusting the boundary of each G-Cell and shifting the modules according to the new G-Cell shape. Figure 1 illustrates the basic idea. In the figure, the G-Cell has insufficient capacity for accomodating the routing tracks. As we know, the global routing capacity is directly related to the length of the G-Cell boundary. If the G-Cell is enlarged proportional to the demand of routing tracks, theoretically no routing overflow would occur. We will show that the resizing of G-Cell can be formulated as two LPs for vertical and horizontal shifting respectively. Instead of solving the computational expensive LP, we relax the LP and solve it by an efficient longest-path computation, which is the

*This work was partially supported by IBM Faculty Award and NSF under grant CCF-0540998

ICCAD'09, November 2-5, 2009, San Jose, California, USA.

major factor contributing to our fast runtime when shifting the modules. After performing module shifting, we will legalize the placement solution. Then we will apply our second technique, the congestion-driven detailed placement (DP) to probe better routability. It aims at better distributing routing demands. Congestion reduction is realized by weighting the HPWL with a congestion coefficient during detailed placement. The Shifting-Legalization-DP procedure forms one iteration of refinement. We will call the refinement repeatedly until the solution is good enough. Practically only a small number of iterations (usually 2-3) is sufficient to achieve good routability.

CROP is a fast and effective refinement tool for mixed-size placement solution. First, CROP is independent of any placer. Congestion-driven techniques proposed by [3], [4], [5], etc. are very tightly coupled with the underlying placer. Their proposed methods cannot be easily integrated into various placement tools. Second, CROP shows good performance in improving the routability. The difference of our model of module shifting from previous works (e.g. mPL-R+WSA [7]) is that ours is more refined. Instead of shifting the cut-lines of a hierarchically sliced layout, CROP shifts the boundary of each G-Cell. Moreover, previous techniques usually lump vertical and horizontal congestion together. Yet our congestion shifting model differentiates the vertical and horizontal directions. Third, CROP runs very fast. For instance, the design with 800k modules and 800k nets (adaptec5) takes less than 10 minutes to execute.

For achieving the advantages we mentioned above, we propose the following techniques:

- A placement routability refinement flow which is independent of any placer and router
- A more refined and directional module shifting model
- A longest-path computation method aiding fast runtime of module shifting
- A congestion-driven global swap technique by weighting HPWL with congestion.

We apply CROP to refine placement solutions obtained from FastPlace 3.1 [13], NTUplace3 [14], mPL6 [15] and R-NTUplace3 [4], on ISPD05/06 benchmarks [1] [2]. The results reveal that CROP effectively reduces congestion within a very short runtime. For instance, before applying refinement, there are 6, 11, 13 and 11 unroutable placement solutions for FastPlace3.1, NTUplace3, mPL6 and R-NTUplace respectively. With CROP, the number is reduced to 1, 1, 4 and 1.

The rest of paper is organized as follows: Section 2 provides preliminaries on global routing and general flow of CROP. Section 3 introduces the details of congestion-driven module shifting technique. Section 4 explains congestion-driven detailed placement. In Section 5, we make comparison for results on ISPD05/06 Benchmarks and conclusion will be made in Section 6.

2. PRELIMINARIES

In this section we will introduce some terminology and present the overview of CROP. We will also talk about the congestion estimation method that guides CROP.

2.1 Motivation

The placement region is partitioned into a set of G-Cells to perform global routing. The G-Cells are illustrated in Figure 2. The global routing will be performed across the boundaries between adjacent G-Cells. In the global routing grid graph, each G-Cell will be represented by a node and each G-Cell boundary will be represented by an edge between two nodes, which is referred to as global routing edge. If the usage U_e is over capacity C_e for any edge e, the overflow is calculated as $O_e = U_e - C_e$. If there is no congested edge, then the design is routable in global routing stage.

To improve the routability, the first method is to supply more routing resources, or in other words, to increase global routing capacities. The

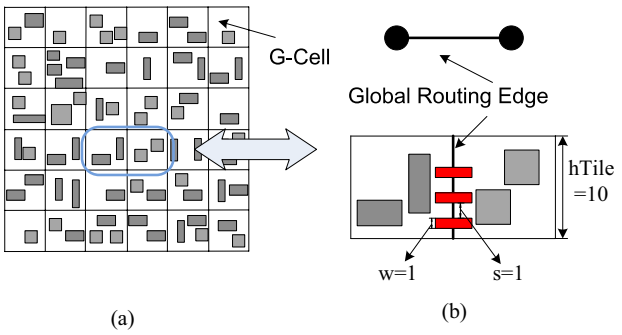

Figure 2: An illustration of G-Cells and global routing across G-Cell boundary.

global routing edge capacity is proportional to the length of the G-Cell boundary. For instance, in Figure 2, the highlighted G-Cell boundary has a capacity of 5. The capacity equals $\frac{hTile}{w+s}$, where $hTile$, w and s denote the height of G-Cell, wire width and wire spacing respectively. It motivates our congestion-driven module shifting technique by reshaping each G-Cell for better allocating routing resources. The second method for routability improvement is to reduce routing demands. We proposed the congestion-driven detailed placement to compensate the wirelength loss at the module shifting stage and further improve routability. The two techniques will be discussed in Section 3 and Section 4 respectively.

2.2 CROP Flow

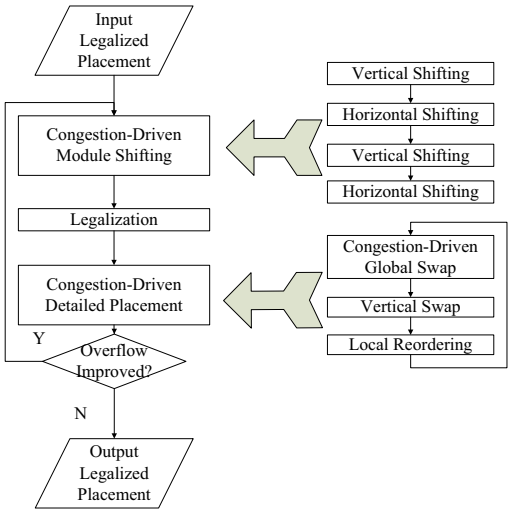

Figure 3: Algorithm flow.

The flow of CROP is illustrated in Figure 3. The input design is a legalized mixed-size placement solution. We first apply congestion-driven module shifting to adjust the position of modules by resizing G-Cells. The module shifting is performed in a directional manner, which means X and Y direction is individually processed. After two rounds of module shifting, the post-shifting placement solution will be legalized. Then congestion-driven detailed placement will be called to further improve the solution quality. The above process will be repeatedly applied until the solution gets stable. Normally it only takes two to three iterations to end the process.

2.3 Congestion Estimation

Previous congestion estimation methods could be roughly grouped into two categories, bounding box based or global routing based. Usually the bounding box method is a very rough way for measuring congestion since no routing structure is constructed. The global routing based estimation, on the other hand, constructs routing topology. So it

is sometimes referred to as topology based estimation. Usually simple pattern routing such as L/Z routing is applied.

In this paper, we use global routing as congestion estimator. Nevertheless, in addition to using the simple L/Z pattern routing, we incorporate the fast and more accurate 3-bend routing. The 3-bend routing technique was proposed in FastRoute 4.0 [16]. It has been shown in [16] that the time complexity of 3-bend routing is $O(mn)$ for a net spanning a region of $m \times n$ G-Cells, which is as fast as Z routing. Besides the fast computing time, it is more flexible in choosing routing paths as it allows necessary detours. The rendered routing solution will be more close to the final solution and thus more accurate.

In CROP, both congestion-driven module shifting and congestion-driven detailed placement are guided by global routing. It is applied before each round of vertical shifting, horizontal shifting and congestion-driven global swap.

3. CONGESTION-DRIVEN MODULE SHIFTING

In this section, we will discuss the congestion-driven module shifting method for moving standard cells and big macros. As discussed before, we first shift the boundary of each G-Cell, and then update the module positions accordingly. In the following subsections, we first formulate LPs of X and Y directions for resizing each G-Cell. Then we show that the LPs can be relaxed and solved by a more efficient longest-path computation. Module updating method and our approach to handle big macros will also be discussed in details.

3.1 Resizing G-Cells by Linear Programming

We will formulate linear program for resizing G-Cells to accomodate routing usage. To facilitate the formulation, let's first assume each module (M_k) is a standard cell (with smaller area than G-Cell) for the time being.

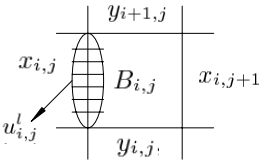

Figure 4: Notation for the LP formulation.

We first partition the placement region into $m \times n$ G-Cells. Let $B_{i,j}$ represents each G-Cell, where i ($i \in \{1, ..., m\}$) denotes the row and j ($j \in \{1, ..., n\}$) denotes the column. We introduce coordinate variables for each of the G-Cell boundary. For $B_{i,j}$, let $x_{i,j}$ and $x_{i,j+1}$ denote the x-coordinate for **left boundary** and **right boundary** respectively. And likewise $y_{i,j}$ and $y_{i+1,j}$ denote the y-coordinate for **bottom boundary** and **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 across left, right, bottom and top boundary for $B_{i,j}$. We also introduce H, W, $hTile$ and $wTile$ to denote height of placement region, width of placement region, original height of G-Cell and original width of G-Cell.

Without loss of generality, we only consider the horizontal shifting of vertical boundaries. Similar equations can be derived for the vertical shifting case.

$$\textbf{max : } \sigma$$

s.t.

$$x_{i,j+1} - x_{i,j} \geq \sigma \times MAX(f^{-1}(u^b_{i,j}), f^{-1}(u^t_{i,j})) \qquad \forall i,j \tag{1}$$

$$0 \leq \sigma \leq 1 \tag{2}$$

$$x_{i,j+1} - x_{i,j} \geq \frac{\sum_{k \in B_{i,j}} area(M_k)}{hTile} \qquad \forall i,j \tag{3}$$

$$|x_{i,j} - x_{i+1,j}| \leq C \qquad \forall i,j \tag{4}$$

$$0 \leq x_{i,1} \qquad \forall i \tag{5}$$

$$x_{i,n+1} \leq W \qquad \forall i \tag{6}$$

Then we will explain each of the constraints.

• *Routability Constraints* (Equations 1 and 2)
As we mentioned in Section 2, routing capacity is proportional to the length of G-Cell boundary. Ideally, if each G-Cell is sufficiently large, there would be no congestion because each G-Cell boundary is capable of holding the crossing routing wires. In the routability constraints, $x_{i,j+1} - x_{i,j}$ represents the width of $B_{i,j}$. On the right hand side, $f^{-1}(u)$ is the inverse function of $f(l)$, which maps G-Cell boundary length l to routing capacity. So $f^{-1}(u)$ is the function translating the given usage to sufficient length of boundary. In particular, $f(l)$ is given in Equation 7.

$$f(l) = gB \times (\frac{l}{p} \times gL + \frac{l}{p} \times (layer - 1)) \tag{7}$$

p is wire pitch (width plus spacing), $layer$ represents number of 3D routing layers for vertical direction. gB is the guard band coefficient and gL is the ground layer reduction coefficient. Our Length-Capacity model is borrowed from the ISPD07/08 Global Routing Contest [17] [18].

However, if the placement solution is very congested, the routability constraints may be too hard to satisfy. In other words, the formulated LP might be infeasible. Therefore we place a relaxing variable σ in the routability constraints. σ here can be viewed as a scaling factor over original constraint. The value of σ is bounded between 0 and 1. When σ is 1, we do not relax the constraints. With smaller σ value, the routability constraints become less restrictive.

• *G-Cell Area Constraints* (Equation 3)
These constraints ensure that each G-Cell has enough space to hold the modules inside. Otherwise, it would create huge overlaps between modules when the non-congested G-Cells shrink excessively. In the formulation, since we consider shifting in X direction, the height of G-Cell is fixed.

• *Movement Constraints* (Equation 4)
The input design has already been a legalized placement solution. It is necessary not to disturb the original placement too much. Hence we introduce movement constraints to restrict the shifting between adjacent G-Cell boundaries. In the equations, C is a constant which represents the degree of flexibility of moving adjacent G-Cell boundaries. In experiment we set C to be $0.5 \times wTile$. Because of the absolute sign, it can be expanded as follows,

$$x_{i+1,j} - x_{i,j} \geq -C \qquad \forall i,j \tag{8}$$

$$x_{i,j} - x_{i+1,j} \geq -C \qquad \forall i,j \tag{9}$$

• *Placement Region Constraints* (Equations 5 and 6)
Finally, these constraints ensure that all the boundaries should be within the placement region. Note that the other constraints (e.g., G-Cell area constraints) implicitly guarantee that $x_{i,j} \leq x_{i,j+1}$ for $j \in \{1, \ldots, n\}$.

3.2 Longest Path based Solution

The LP is expensive in terms of solving time. Next we will introduce a technique based on longest path to solve it efficiently.

When we investigate the proposed LP in Section 3.1, we find that if σ is fixed, the LP becomes a feasibility check problem (only has constraints). Since Equations 1,3,8 and 9 are all difference constraints, we propose the following strategy to solve the LP. We use an outer loop which keeps decreasing σ until the LP is feasible. Inside the loop, for a fixed σ, we check feasibility by longest path computation. If infeasible, the longest path solution will suggests how σ should be decreased.

Let's first assume σ is fixed. The feasibility of the constraints can be checked by performing a longest path computation on a directed graph called G-Cell boundary graph(B-graph) $G(V, E)$. Each G-Cell boundary associated with $x_{i,j}$ is represented by a vertex $v_{i,j} \in V$. 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. To distinguish

different edge types, we name E_r, E_a and E_m for the set of edges incurred by routability constraints, G-Cell area constraints and movement constraints respectively. Figure 5 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 Equations 1,3,5,8 and 9. So the feasibility of the constraints can be determined by checking whether $x_{j,n+1} \leq W$ for all i (Equation 6).

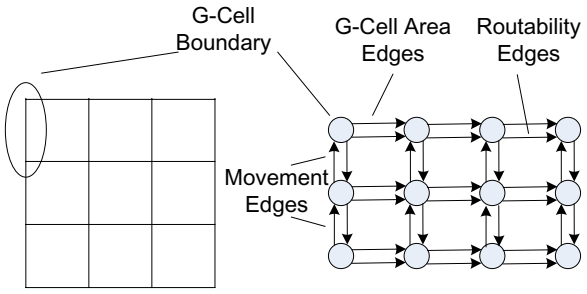

Figure 5: Convert the G-Cell boundary into B-graph.

We observe that the proposed B-graph contains directed cycles. The directed cycles are caused by movement edges (E_m), which are used to control the disturbance of original placement. However, as suggested by [19] , it's NP-Complete to find the longest path for a graph with directed cycles. The neat longest path algorithm cannot be applied in this case.

The issue can be resolved by introducing the diagonal edges E_d to replace the hard-to-handle movement edges. Diagonal edges are an alternative method of maintaining original placement solution. Figure 6 illustrates the idea. We merge the perpendicular edges ($e1$ and $e2$) and replace $e1$ with the diagonal edge ($e3$). Note that here $e2$ represents the longer one of area edge and the routability edge. The cost of a diagonal edge is the total cost of the perpendicular edges, which is to say, $\|e3\| = \|e1\| + \|e2\|$.

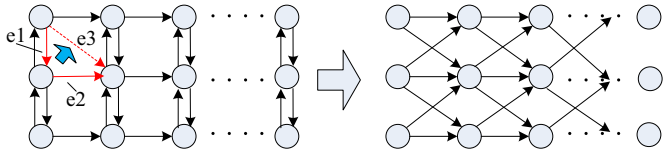

Figure 6: Replace movement edges with diagonal edges to facilitate longest path computation.

After replacing the movement edges with diagonal edges, the B-graph has become a DAG. Now we are able to perform longest path computation. It can be done very effecitvely by a fast scanning of each vertex in B-graph according to the topological order.

Next we will discuss the outer loop for determining the maximized σ to solve the LP. Initially σ is set to 1. If the resulting longest path length $L_p = MAX_{i=1,\ldots,m}(x_{i,n+1})$ 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 divide the edges along the longest path L into two parts: hard edge $E_h^L = (E_a \cup E_d) \cap L$; and soft edge $E_s^L = E_r \cap L$. $L_h = \sum_{e \in E_h^L} \|e\|$ and $L_s = \sum_{e \in E_s^L} \|e\|$. 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 fixed outline at one time. Some other path may still be longer than W even after scaling. 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 $x_{i,n+1}$ for at least one more i will become less than or equal to W in each iteration. In practice, it usually takes less than 10 iterations. Figure 7 shows our complete algorithm to solve the LP.

Algorithm for solving the LP
Input: B-graph G(V,E)
Output: Maximized σ
begin
  $\sigma = 1$
  while(1)
    Perform Longest-path algorithm
    $L_p = MAX(x_{i,n+1})\ \forall i$
    if($L_p \leq W$)
      Break
    else
      $\sigma = \sigma \times \frac{W - L_h}{L_s}$
  end while
end

Figure 7: The longest path algorithm and iterative scaling for deciding boundary locations

Up till now, we have discussed the algorithm for solving the resizing problem by longest path based solutions. And our algorithm assumes $x_{i,1} = 0\ \ \forall\, i$. However, the potential problem for such assumption is it will result in packing the design to left bound. Please refer in Figure 8, the front curve of spreading G-Cell boundaries under the longest path computation would be maintained after the scaling. The placement will be compacted excessively for less congested regions. To resolve this problem, we assume $x_{i,n+1} = W\ \ \forall i$, and make $x_{i,1} \geq 0$ as the feasibility check. In this way, we could obtain two sets of x-coordinate for each G-Cell boundary, let's say $x_{i,j}^l$ and $x_{i,j}^r$. Actually the two sets of solution represents the two extreme cases in which the design is either packed to left or right. We thus name the two coordinates as valid range and the valid range will be used in determining the new boundary coordinates. Let $X_{i,j}$ denote the original G-Cell boundary coordinate ($X_{i,j} = (j-1) \times wTile$). If $X_{i,j}$ is within the valid range, it means the resulting packing is not too tight for both cases, and we will not move the boundaries ($x_{i,j} = X_{i,j}$). Otherwise, we move boundaries to the closer of $x_{i,j}^l$ or $x_{i,j}^r$.

$$x_{i,j} = \begin{cases} x_{i,j}^r & if \quad X_{i,j} > x_{i,j}^r \\ x_{i,j}^l & if \quad X_{i,j} < x_{i,j}^l \\ X_{i,j} & otherwise \end{cases} \tag{10}$$

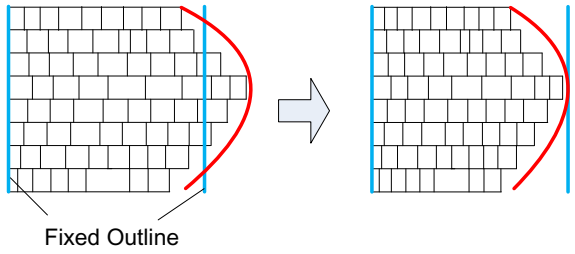

Figure 8: Problem of compacting design to left

3.3 Module Relocation

After adjusting each G-Cell boundary, the modules inside each G-Cell will be shifted based on the new G-Cell boundary coordinates. CROP updates the module location by maintaining the same raio of distance to both boundaries after G-Cell boundary adjustment. As illustrated in Figure 9, $L1$ and $R1$ is the original distance between center of module m to left boundary and right boundary respectively. The module will be relocated such that $L1/R1 = L2/R2$.

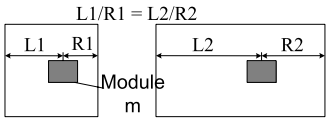

Figure 9: Module shifting illustration.

3.4 Macro Block Handling

In the traditional design flow, macro blocks and IP blockages are first placed and fixed. Then the standard cells are filled within the "gaps" between the blockages. Usually, certain amount of routing resources will be reserved for inner routing. The existence of big macros make the routability issue more complicated. Hence we need to extend our method of handling standard cells to handling mixed-size placement solution. In order to explore larger possibility of congestion reduction, in this paper we will assume all the macros are movable

The way we handle big macro shifting is similar as what we have discussed for standard cells. As in Figure 10, CROP merges the G-Cells that are covered or partially covered by the big macro M. The merged G-Cell is named super G-Cell. Likewise, in the B-graph, we merge vertex along the super G-Cell boundary and delete those inner nodes. Two factors need be considered for the macro shifting.

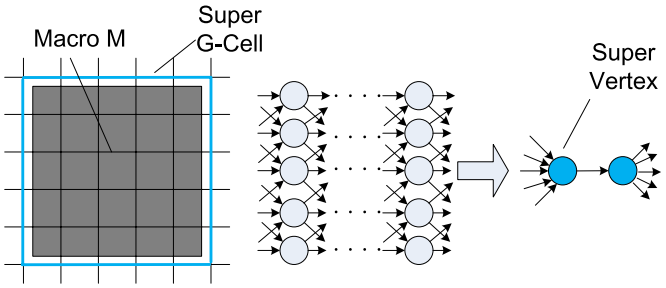

Figure 10: Merging of G-Cells for macro blocks.

First, in VLSI routing, certain amount of routing resources above big macros will be reserved for inner routing purpose. This is also referred to as block porosity effect. To cope with block porosity, we increase the cost of the corresponding routability edges (E_r) in B-graph to compensate the capacity loss. In our experiment, we stick with the porosity reduction ratio used in generating ISPD07/08 global routing benchmarks.

Second, although the overlapping between standard cells is permissible, big macros need be kept apart from each other. Therefore, we place additional edges in the B-graph for guaranteeing the non-overlapping property between big macros.

4. CONGESTION-DRIVEN DETAILED PLACEMENT

Detailed placement (DP) is commonly applied after global placement to improve HPWL for legalized placement solution. We develop a congestion-driven DP technique based on FastDP [20]. The flow of our proposed DP is shown in Figure 3. We only make global swap to be congestion-aware. The vertical swap and local reordering are very local and contribute little to congestion reduction, so we still keep them HPWL targeted.

4.1 Congestion-Driven Global Swap for 2-pin Nets

Global swap exchanges modules to improve HPWL based on a greedy pairwise position exchange. In [20], global swap is the stage contributes most to the reduction of HPWL. However, HPWL hardly reflects routability, especially when cells are swapped into highly congested regions. Hence, we change the swapping evaluation function incorporating the congestion component. The HPWL is weighted by the congestion factor of α_n for net n. Simply put,

$$rHPWL_n = HPWL_n \times \alpha_n \tag{11}$$

Then the global swap will be guided by the congestion weighted HPWL. If we swap standard cell *A* with standard cell *B*, the gain after swapping should be, $Gain_{A-B} = \sum_{n \in N_A}(rHPWL_n - rHPWL'_n) + \sum_{n \in N_B}(rHPWL_n - rHPWL'_n)$. where N_A and N_B are the sets of nets that *A* and *B* are connecting to. $rHPWL'_n$ is the new cost after tentative swapping.

Now we discuss how to set α_n in CROP. For simplicity, let's first consider 2-pin nets. The straightforward approach, for instance, is to calculate the average congestion inside the bounding box. But this method is too rough to be reliable. So alternatively, we turn to incorporate a more accurate model instead of simply lumping congestion together. The method we propose is to enumerate all possible Z routing paths inside the bounding box and calculate α_n by the average congestion along all the paths. Hence,

$$\alpha_n = \frac{w_{tol}}{E} = \frac{\sum_{p \in P}\sum_{e \in p} w(e)}{E} \tag{12}$$

In the equation, P is the set of all Z routing paths inside the bounding box. e represents each global routing edge along path p. E is the total number of global edges for all paths, and $w(e)$ is the weight of edge e. w_{tol} represents the sum of weight. $w(e)$ is calculated by Equation 13. If there is overflow, the weight will be increased quadratically. The fact we set the weight to 1 for non-congested edges means we keep penalizing wirelength. Or it may end up prolonging the wirelength significantly.

$$w(e) = \begin{cases} 1 & if \quad O_e = 0 \\ (U_e/C_e)^2 & if \quad O_e > 0 \end{cases} \tag{13}$$

4.2 Speedup Technique based on Look-up Tables

The proposed method in Sec. 4.1 would be very expensive in terms of runtime, since we need to add up edge weights along all Z paths. For a $p \times q$ sized 2-pin net bounding box, the time complexity would be $O((p+q)^2)$. In order to speed up the summing of total weight, we propose a look-up table method.

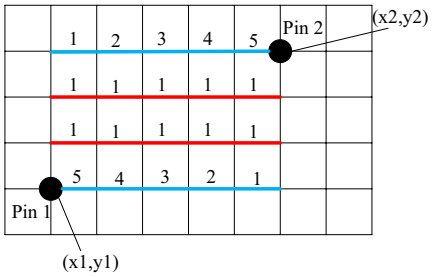

Figure 11: Number of Z routing paths passing through each horizontal global routing edge.

We let $w_{tol} = w_h + w_v$, where w_h is the total weight of horizontal global edges and vice versa for w_v. Without loss of generality, let's discuss the calculation of w_h. Similar results can be derived for w_v. Figure 11 illustrates a 2-pin net in global routing grid graph. Let $e^h_{i,j}$ denotes each horizontal global routing edge in the global routing grid graph. We mark the number of Z routing paths passing through each horizontal global routing edge. Suppose the coordinates are (x1, y1) and (x2, y2) for pin 1 and pin2 respectively. As suggested by Figure 11, $w_h = w1 + w2 + w3$, where $w1 = \sum_{j=x1}^{x2-1}((x2-j) \times w(e^h_{y1,j})$, $w2 = \sum_{i=y1+1}^{y2-1} w(e^h_{i,j})$, and $w3 = (j - x1 + 1) \times w(e^h_{y2,j})$.

We introduce five $m \times n$ tables, $T1$, $T2$, $T3$, $T4$ and $T5$. Each entry in the table corresponds to one grid point in the routing grid graph. The meaning of entry (r, c) for each table is summeried in Table 1.

T1	$T1_{(r,c)} = \sum_{j=\{1,\ldots,c-1\}} w(e^h_{r,j})$
T2	$T2_{(r,c)} = \sum_{j=\{c,\ldots,n-1\}} w(e^h_{r,j})$
T3	$T3_{(r,c)} = \sum_{j=\{1,\ldots,c-1\}} (c-j) \times w(e^h_{r,j})$
T4	$T4_{(r,c)} = \sum_{j=\{c,\ldots,n-1\}} (j-c+1) \times w(e^h_{r,j})$
T5	$T5_{(r,c)} = \sum_{i=\{1,\ldots,c-1\},j=\{1,\ldots,r\}} w(e^h_{i,j})$

Table 1: Notation of look-up tables.

Based on the notations in Table 1, $w1 = T4_{(x2,y1)} - T4_{(x1,y1)} - (x2-x1) \times T2_{(x1,y1)}$, $w2 = T3_{(x1,y2)} - T3_{(x2,y2)} - (x2-x1) \times T1_{(x2,y2)}$, and $w3 = T5_{(x2,y2)} - T5_{(x1,y2)} - T5_{(x2,y1)} + T5_{(x1,y1)}$.

With the help of five look-up tables, the computing of w_{tol} can be done very efficiently. Now the time complexity is $O(1)$.

All the proposed tables can be constructed very efficiently by dynamic programming. In Table 2, we also show how dynamic programming is performed. Basically, the computing of current entry can be broken into subproblems using the value of entry that has already been computed.

T1	$T1_{(r,c)} = T1_{(r,c-1)} + w(e^h_{r,c})$
T2	$T2_{(r,c)} = T2_{(r,c+1)} + w(e^h_{r,c})$
T3	$T3_{(r,c)} = T3_{(r,c-1)} + T1_{(r,c)}$
T4	$T4_{(r,c)} = T4_{(r,c+1)} + T2_{(r,c)}$
T5	$T5_{(r,c)} = T5_{(r,c-1)} + T5_{(r-1,c)} - T5_{(r-1,c-1)} + w(e^h_{r,c})$

Table 2: Apply dynamic programming to construct look-up tables.

4.3 Multi-pin Nets Handling

We have discussed our method for weighting congestion for 2-pin nets. But in real design, many nets are multi-pin nets. Without knowing the exact routing path and topology, it is not easy to weight the congestion for a multi-pin net. Hence we derive the multi-pin net weighting model in a simple way. Suppose the investigated module m has multi-pin connection. Suppose the multi-pin net without m forms a bounding box $BBox$. If m is within $BBox$, we igore weighting process. Otherwise we just introduce a 2-pin net from m to the nearest pin in $BBox$. With this simple method, module m will only has 2-pin connections.

5. EXPERIMENTAL RESULTS

All our experiments are performed on a machine with 2.4GHz AMD Opteron processor and 4G of memory. We run the experiments on 14 ISPD05/06 Benchmarks [1] [2]. The way we derive the global routing grid from the placement solution follows the rule of ISPD07/08 global routing contest benchmarks [17] [18]. But we adjust some capacity parameters to make them harder to route.

5.1 Effectiveness of Proposed Techniques

In this subsection we show the effectiveness of our techniques for routability improvement on design adaptec3 and bigblue1 generated by routability-driven NTUplace (R-NTUplace) [4] in Table 3. The overflow is the global routing result obtained from the technique mentioned in Section 2.3. First, the total overflow is consistently improved after each round of shifting. Second, the shifting in one direction will introduce congestion overhead. For instance, when we shift modules in the X direction, the horizontal overflow becomes worse. The overhead comes from the extra wirelength. However, the overhead is smaller than the gain, which attributes to the movement constraints in our LP formulation. Third, the placement solution during module shifting is not legalized. The congestion data cannot be fully trusted. Hence, we also show the overflow after congestion-driven global swap, when the design has been legalized. It shows that the congestion is improved over the original one, which better indicates the effectiveness of congestion reduction of proposed techniques in CROP.

	adaptec3				bigblue1			
	H o.f.	V o.f.	Total o.f.	Legal?	H o.f.	V o.f.	Total o.f.	Legal?
Before	53428	102804	156232	Y	50531	34409	84940	Y
Y shifting	40283	103637	143920	N	25320	45628	70948	N
X shifting	41168	93149	134317	N	31275	21525	52800	N
Y shifting	34239	96298	130537	N	23306	31703	55009	N
X shifting	37295	90350	127645	N	27025	19419	46444	N
Legalization	42548	93122	135670	Y	28223	20145	48478	Y
Global Swap	32335	81013	113348	Y	17238	17094	34322	Y

Table 3: Congestion reduction during module shifting

5.2 ISPD05/06 Benchmarks

In this subsection we show the full experimental results on 14 sets of ISPD05/06 mixed-size placement benchmarks. Most of these benchmarks have a lot of fixed macros. As we mention in Section 3.4, in order to explore a larger possibility of improving routability, we assume the fixed macros are movable. However, we discover that the actual disturbance is not huge compared with the original design. We apply CROP for the legalized placement solutions generated by FastPlace3.1 [13], NTUplace3 [14], mPL6 [15], and R-NTUplace [4]. Note that here we use the full global router to measure the routability, not the global routing mentioned in Section 2.3. FastRoute 4.0 [16] is utilized to do the full routing. We also tried other routing tools such as NTHURouter [21] etc. and they report similar results. The reason for choosing FastRoute 4.0 is it runs comparably faster than other available routing tools. It is noted that the final evaluation of routability should be detailed routing. In our paper, we just consider the global routing stage as the routability indicator.

Table 4 shows our results in detail. For each placer, we show the routing results before and after applying our tool. The entry with "/" means the original placement has already been routable so we do not apply CROP. Before applying our tool, there are 6, 11, 13, and 11 unroutable cases for FastPlace3.1, NTUplace3, mPL6 and R-NTUplace respectively. After applying CROP, the number is reduced to 1, 1, 4 and 1. Please note that newblue3 is proved to be unroutable. Hence, CROP is very effecitve in reducing congestion.

We also report the CROP execution runtime in Table 4. The runtime of our tool is trivial comparing with original placement runtime. For instance, the total runtime to process 14 benchmarks by mPL6 on our platform is more than 24 hours. While the total execution time of CROP is around one hour. Noticeably, the saved runtime in the routing is significant. For mPL6, the speed-up of routing time is 7× on average.

Another aspect for evaluating CROP is the routed wirelength. After applying CROP, the total wirelength 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 on the same level with original design. But in many cases, we notice the routed wirelength becomes better. The reason is the new placement solution is easier to route, such that the router does not need to make huge detours and eventually saves the wirelength.

In Figure 12, we show the congestion plot of bigblue1 generated by NTUplace3 before and after applying CROP. The congested region is shown by the red color. Obviously, CROP effectively alleviates the congestion problem.

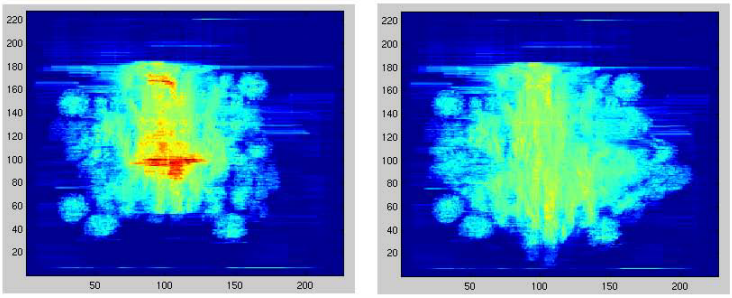

Figure 12: Congestion plot of bigblue1 before and after CROP

6. CONCLUSION

In this work, we have presented CROP to improve routability for placement solution as a refinement process. Our tool is independent of any placer and router. The main techniques involves congestion-driven module shifting and congestion-driven detailed placement. CROP produces promising results for various placement tools. We will further improve its performance and stability.

7. REFERENCES

[1] ISPD05 placement contest benchmarks. http://www.sigda.org/ispd2005/contest.htm.

Metrics	Tools		a1	a2	a3	a4	a5	b1	b2	b3	n1	n2	n3	n4	n5	n6
Routing Overflow	FP3.1	w/o	/	1260	4	/	/	18755	769	/	/	/	9642	/	46	/
		w	/	0	0	/	/	0	0	/	/	/	9019	/	0	/
	NTUPlace3	w/o	2885	2369	1621	141	/	15896	19793	15259	/	60	9442	3394	/	8
		w	0	0	0	0	/	0	0	0	/	0	8480	0	/	0
	mPL6	w/o	20	18535	22539	5703	7307	46995	1736	4678	/	9	8835	5649	12475	4495
		w	0	11289	660	0	0	0	0	0	/	0	8405	285	0	0
	R-NTUplace	w/o	51	2849	94	20	16	12887	38616	2264	/	898	10065	385	/	/
		w	0	0	0	0	0	0	0	0	/	0	8551	0	/	/
CROP CPU (second)	FP3.1		/	92	200	/	/	73	193	/	/	/	297	/	406	/
	NTUplace3		70	98	238	227	/	70	308	453	/	203	270	198	/	487
	mPL6		75	143	374	279	506	92	229	472	/	216	366	272	630	292
	R-NTUplace		65	97	272	216	301	66	224	690	/	225	256	189	/	/
CROP iterations	FP3.1		/	2	2	/	/	2	2	/	/	/	2	/	2	/
	NTUplace3		2	2	2	2	/	2	3	2	/	2	2	2	/	2
	mPL6		2	2	3	2	3	2	3	2	/	2	2	2	3	2
	R-NTUplace		2	2	2	2	2	2	2	3	/	2	2	2	/	/
Routing CPU (second)	FP3.1	w/o	/	280	223	/	/	1492	1042	/	/	/	13331	/	154	/
		w	/	34	107	/	/	39	282	/	/	/	13832	/	32	/
	NTUplace3	w/o	2031	311	1096	389	/	1945	2517	2692	/	362	12726	1271	/	1088
		w	144	20	101	26	/	46	87	80	/	28	13006	68	/	78
	mPL6	w/o	420	1478	6795	2859	2927	1941	448	2397	/	131	13661	2329	5756	6085
		w	58	974	1066	42	202	75	256	69	/	23	13554	1581	200	155
	R-NTUplace	w/o	642	311	975	248	531	1619	3869	6439	/	933	12801	515	/	/
		w	101	17	108	24	47	41	303	136	/	31	13079	25	/	/
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	/
	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
	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
	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	/	/

Table 4: CROP results on ISPD05/06 Benchmarks

[2] ISPD06 placement contest benchmarks. `http://www.sigda.org/ispd2006/contest.htm`.

[3] P.Spindler and F.M.Johannes. Fast and accurate routing demand estimation for efficient routability-driven placement. In *Proc. Conf. on Design, Automation and Test in Europe*, pages 1226–1231, 2007.

[4] 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 Automation Conf.*, pages 167–172, 2008.

[5] K.Tsota, C.Koh, and V.Balakrishnan. Guiding global placement with wire density. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 212–217, 2008.

[6] X.Yang, B.Choi, and M.Sarrafzadeh. Routability-driven white space allocation for fixed-die standard-cell placement. *IEEE Trans. on Computer-Aided Design and Integrated Circuits and Systems*, 22(4):410–419, April 2003.

[7] C.Li, M.Xie, C.Koh, J.Cong, and P.Madden. Routability-driven placement and white space allocation. *IEEE Trans. on Computer-Aided Design and Integrated Circuits and Systems*, 26(5):167–172, May 2008.

[8] U.Brenner and A.Rohe. An effective congestion-driven placement framework. In *Proc. ACM/SIGDA Intl. Symp. on Physical Design*, pages 6–11, 2002.

[9] M.Pan and C.Chu. IPR: An integrated placement and routing algorithm. In *Proc. ACM/IEEE Design Automation Conf.*, pages 59–62, 2007.

[10] M.Pan and C.Chu. FastRoute 2.0: A high-quality and efficient global router. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 250–255, 2007.

[11] N.Viswanathan and C.Chu. FastPlace: efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Proc. ACM/SIGDA Intl. Symp. on Physical Design*, pages 26–33, 2004.

[12] J.Roy and I.L.Markov. Seeing the forest and the trees: Steiner wirelength optimization in placement. *IEEE Trans. on Computer-Aided Design and Integrated Circuits and Systems*, 26(4):632–644, April 2007.

[13] N.Viswanathan, M.Pan, and C.Chu. FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 135–140, 2007.

[14] 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. on Computer-Aided Design and Integrated Circuits and Systems*, 27(7):1228–1240, July 2008.

[15] 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 Intl. Symp. on Physical Design*, pages 227–229, 2005.

[16] Y.Xu, Y.Zhang, and C.Chu. FastRoute 4.0: Global router with efficient via minimization. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 576–581, 2009.

[17] ISPD07 global routing contest benchmarks. `http://www.sigda.org/ispd2007/contest.htm`.

[18] ISPD08 global routing contest benchmarks. `http://www.sigda.org/ispd2008/contest.htm`.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, NY, 1979.

[20] M.Pan, N.Viswanathan, and C.Chu. An efficient and effective detailed placement algorithm. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 48–55, 2005.

[21] Y.Chang, Y.Lee, and T.Wang. NTHU-Route 2.0: A fast and stable global router. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 338–343, 2008.