RegularRoute: An Efficient Detailed Router Applying Regular Routing Patterns

Yanheng Zhang and Chris Chu

***Abstract*— In this paper, we propose RegularRoute, an efficient detailed router encouraging regular routing patterns. RegularRoute is proposed for potentially better design rule satisfaction based on a correct-by-construction methodology. Given the underlying spacing-based design rules, 2-D global routing solution, and 3-D detailed routing tracks, RegularRoute generates a 3-D detailed routing solution in a bottom-up layer-by-layer framework. At the beginning, local nets, i.e., nets or subnets that are inside one G-cell, are routed using vertical spine routing topology. Routing usage of local nets is treated as blockage when assigning global segments. Then, for each layer we formulate the problem of global segment assignment inside each panel, i.e., grouped routing tracks, as a maximum weighted independent set (MWIS) problem. We propose a fast and effective heuristic to solve the MWIS problem. Unassigned segments are partially routed by a greedy technique. For the unrouted portion of each segment, its terminals are promoted so that the assignment is deferred to the upper layers. At the top layers, we apply the panel merging and maze routing techniques to improve routability. RegularRoute generates a detailed routing solution that satisfies the basic spacing-based design rules. To satisfy all the design rules, we propose an abstract idea of local optimization based on local shift and rip-up-and-reroute, assuming that most design rules are complex functions of local and neighboring geometries. Because of the unavailability of proper academic grid-based detailed routing benchmarks, we propose two sets of detailed routing test cases derived from ISPD98 and ISPD05/06 placement benchmark suites, respectively. Our experimental results demonstrate the effectiveness and efficiency of RegularRoute.**

***Index Terms*— Detailed routing, physical design, routing, VLSI computer-aided design (CAD).**

I. Introduction

BECAUSE of its enormous computational complexity, VLSI routing is usually carried out through consecutive global routing and detailed routing stages. In the global routing stage, rough routing solutions are generated based on G-cell-to-G-cell (i.e., global routing bin to perform global routing) connections on the global routing grid graph. Detailed routing, on the other hand, realizes routing paths honoring exact geometrical constraints based on the global routing solution. Detailed routing is an important design stage in the sense that it is critical for design rule satisfaction and routing completion. The solution quality impacts various design metrics such as timing, signal integrity, chip yield, etc.

Detailed routing has been extensively studied since the 1970s (e.g., [1] and [2]) but the topic is not frequently seen in recent publications. For modern designs in which over-the-cell routing is applied, the most common technique for detailed routing is rip-up-and-reroute, such as the one in Mighty [3]. However, such a sequential net-by-net approach is ineffective in handling congested designs and usually creates unnecessary detour. DUNE [4] and MR [5] proposed to handle full-chip gridless routing by similar multilevel approaches, in which the routing undergoes a coarsening as well as an uncoarsening phase. But these multilevel routers still rely on the sequential rip-up-and-reroute technique, and nets at the upper levels of the hierarchy are routed based on inaccurate information. There are several attempts that consider nets in a more simultaneous manner during detailed routing. Nam *et al*. [6] proposed a detailed router for field-programmable grid arrays based on Boolean satisfiability. Though this approach achieves good solution quality, the runtime is extremely long. Zhou *et al*. [7] introduced track assignment as an intermediate step between global and detailed routing. In track assignment, segments extracted from global routing solution are assigned to routing tracks. This problem is NP-complete and is solved by a heuristic based on weighted bipartite matching. However, the connections of a segment to pins or segments in other layers are not completed during track assignment. They are postponed to detailed routing, which may fail to connect the different parts of a net. Mustafa [8] presented an insightful technique to perform escape routing for dense pin clusters, which is a bottleneck of detailed routing. A multicommodity flow based optimal solution and a Lagrangian relaxation based heuristic were proposed. Nevertheless, the technique was not meant to solve whole-chip-scale detailed routing. Gester *et al*. [9] proposed an algorithm to solve whole-chip routing based on a combinatorial approximation scheme for min-max resource sharing for global routing and a shape-based data structure for detailed routing. However, the algorithm proposed was for gridless routing and it was not easily adaptive to solve grid-based routing problems.

With diminishing feature size, many complex design rules are imposed to ensure manufacturability. It has been reported that for a 32-nm process, the number of rules reaches several thousands [10] and the design rule manual has roughly one thousand pages [11]. The dramatic increase in the number and complexity of the design rules makes detailed routing progressively complicated and time consuming. We notice that

Manuscript received November 10, 2011; revised July 8, 2012; accepted July 21, 2012.

The authors are with the Department of Electrical and Computer Engineering, Iowa State Unviersity, 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.2012.2214491

many of those complex rules are triggered only by nontrivial routing patterns. Here we define regular patterns as those avoiding jogs and unnecessary detours as much as possible. Fig. 1 illustrates two routing solutions for the same problem. The top one is irregular routing with many jogs and detours, while the bottom one is regular routing which only uses simple patterns. If only regular patterns are used, it is not even necessary to check many design rules and the routing solution would correct by construction. On the other hand, if a routing solution is irregular, even though it may not violate any design rule, it is likely to be detrimental to both yield and routability. Moreover, regular routing introduces fewer vias and jogs and less wirelength, and hence is better in terms of timing, signal integrity, and power consumption. Note that the regular routing approach is along the lines of the restrictive design rule approach that the industry has started applying to the device layers to enhance manufacturability. In this paper, we extend it to the interconnect layers.

Potentially, regular routing may adversely affect routability because it is more restrictive and may be less effective in resolving congestion. But we will show that, with an appropriate algorithm, regular routing can be effective since the solution space can be explored much more effectively and efficiently. On the contrary, for routing with general patterns, the solution space is much larger. But it is also much harder to be explored. The best known approach is to route the nets one by one using maze routing together with rip-up-and-reroute. Such an approach is very time consuming (especially if complicated design rules need to be checked repeatedly throughout the routing process) and is prone to getting stuck in local minima.

In this paper, we propose RegularRoute, an efficient detailed router applying regular routing patterns for potentially better design rule satisfaction. In RegularRoute, we handle the detailed routing problem only with basic design rules including the spacing rules for metal–metal, metal-via, and via–via on each layer. The solution will be a valid detailed routing solution with regular patterns and satisfies the basic design rules. Assuming that all design rules are complex functions involving local and neighboring geometries, we propose the general idea of local optimization based on local shift and rip-up-and-reroute to resolve the design rule violations considering all rules. The entire flow can be regarded as a two-phase framework to obtain better design rule satisfaction by applying regular routing and local optimization.

The two-phase framwork is a feasible and effective approach for detailed routing for the following reasons.

1) In regular routing phase, we only consider basic design rules to facilitate better efficiency with less restrictive routing constraints.
2) The routing shapes around certain nets keep changing, so it is inefficient for handling all design rules at early phase of detailed routing. The primitive knowledge of design rule violations might only impede the router to resolve congestion.
3) Local optimization will be feasible by applying local shift and a little rip-up-and-reroute given sufficient free space. The free space is manageable during regular routing with a proper routing density target.

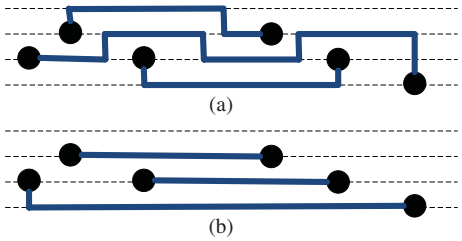

Fig. 1. (a) Nontrivial routing patterns. (b) Regular routing patterns.

The novelties of this paper include the following.

1) We propose an efficient detailed routing algorithm called RegularRoute for potential better design rule satisfaction using regular routing patterns.
2) We introduce a new bottom-up layer-by-layer and panel-by-panel framework in RegularRoute.
3) We propose to route local nets by vertical spine routing topology (VSRT) and fix the solution when assigning global segments.
4) We formulate the global segment assignment problem for each panel as an maximum weighted independent set (MWIS) problem. The formulation facilitates a certain level of parallelism of assigning global segments in one panel.
5) We employ various optimizing techniques including partial assignment, terminal promotion, and panel merging for better routability.
6) We propose the general idea of local optimization after regular routing to satisfy most of the design rules. We also propose to maintain free space in RegularRoute by honoring detailed routing density target.

We have implemented RegularRoute and tested its performance on detailed routing test cases derived from ISPD98 [12] and ISPD05/06 [13], [14] placement benchmarks, respectively. Experimental results demonstrate the effectiveness and efficiency of RegularRoute.

The rest of this paper is organized as follows. Section II provides the problem formulation and an overview of RegularRoute and the whole framework. Section III discusses the methods used to route local nets. In Section IV, we introduce techniques for assigning global segments. In Section V, we present techniques for local optimization after regular routing to satisfy all design rules. Experimental results are given in Section VI.

II. Preliminaries

In this section, we present the terminologies, problem formulation, and the flow of RegularRoute and local optimization.

A. Terminologies and Problem Formulation

1) Terminologies: In this paper, because regular routing is considered, we model the routing resource as a 3-D regular grid graph. Each grid edge, i.e., the edge between two grid points, can accommodate one unit of wire usage. Edges that are covered by macros, preroutes, or other nets are not

assignable. We define the space between grid points to make sure the routed wires respect metal–metal spacing rule for each layer. The spacing rules for metal–via and via–via are specifically checked during the assignment.

Each layer of the routing grid has a preferred routing direction, and the preferred directions of adjacent layers are perpendicular to each other. We assume that the preferred direction of lowest layer (metal 1) is horizontal. For each layer, the routing usage that is in the preferred direction is called preferred usage. Otherwise, the routing usage that is perpendicular to the preferred direction is called nonpreferred usage. A sequence of grid edges along the preferred routing direction of each layer is called a routing track. Routing tracks can be either regularly spaced or iregularly spaced in our formulation.

2) Problem Formulation: The inputs are the underlying spacing-based design rules, a placed netlist, and a corresponding 2-D global routing solution. In this paper, we assume that all pins are on metal 1.[1] The detailed routing problem is defined to route all nets in the netlist on the detailed routing grid based on the 2-D global routing solution. The routed usage should respect the underlying spacing-based design rules i.e., metal–metal, metal–via, and via–via spacing rules.

Note that we are actually solving both layer assignment and detailed routing since we employ 2-D global routing solution as input. Common practices in global routing generate 3-D solutions. But 2-D solutions are more favorable in our problem because: 1) there are more constraints in 3-D solution as the layer of each global route has been specified, the detailed routing problem would be more restrictive to solve, and 2) there is insufficient detailed information to perform layer assignment in global routing correctly (e.g., some global edges may have larger capacity than reality). But we will discuss the extension of RegularRoute to handle layer constraints in Section IV-G.

The primary objective of detailed routing is to complete routing as many nets as possible. The secondary objectives include minimizing nonpreferred usage, via count, and wirelength. In industrial applications, common design metrics include timing performance, signal integrity, power consumption, chip yield, etc. These metrics can potentially be incorporated into our framework but they will not be dealt with in this paper.

In our framework, the net or subnet that resides inside one G-cell is named a local net. Local nets are not captured in global routing. The 2-D global routing solution of each net is reorganized into a set of global segments by breaking it at the turning points. Each segment is a horizontal (or vertical) route which spans multiple G-cells in a row (or column).

A valid detailed routing solution in our framework needs the handling of both the local nets and the global segments. In other words, the detailed routing could also be formulated as assigning both the local nets and the global segments to the routing tracks.

Ideally, each segment should be assigned to one track. In order to make routing less restrictive, assigning a segment to more than one track connected by short nonpreferred usage or via is allowed but discouraged. We define a panel to be the collection of all tracks on one layer within one row (for odd layer) or one column (for even layer) of G-cells. Fig. 2 shows the definitions of track, segment and panel. Note that each segment can only be assigned to tracks on a deck of panels that are on different layers but are associated with the same row/column of G-cells spanned by the segment. In other words, it is natural to perform the global segment assignment panel by panel.

[1]RegularRoute could handle pins on other layers. The extension will be discussed in Section IV-E.

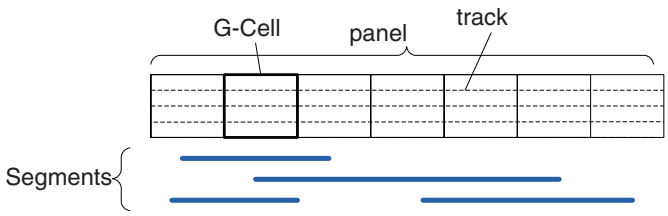

Fig. 2. Definitions of track, segment, and panel.

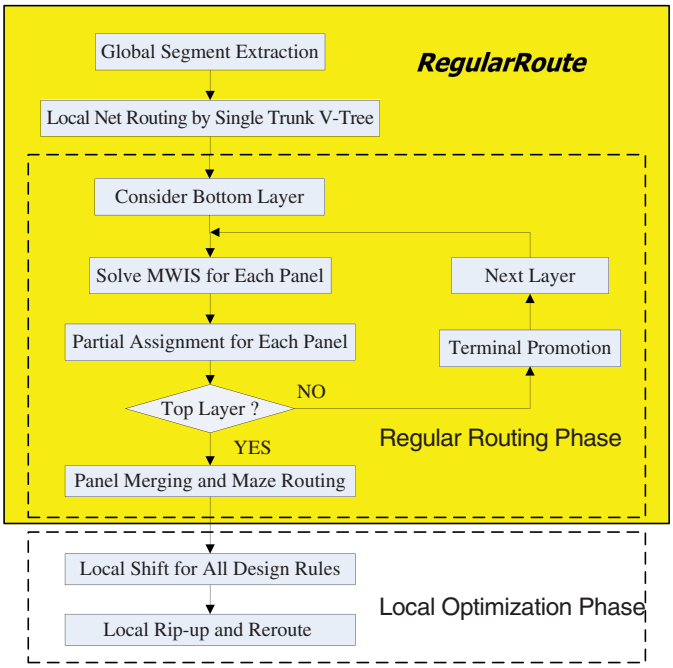

Fig. 3. Flowchart for RegularRoute and the proposed algorithm.

B. Algorithm Flow

The flow of the proposed algorithm is illustrated in Fig. 3. In the figure, there are two main blocks with dotted line. The top one represents the regular routing phase and the bottom one shows the local optimization phase. The highlighted region represents the major steps for RegularRoute we implement to solve regular routing.

RegularRoute starts with extracting and reorganizing global segments by breaking the 2-D global routing solution. Then, local nets are prerouted using VSRT. In the following global segment assignment, the routed usage of local nets is treated as blockages. Next, we perform global segment assignment in a bottom-up layer-by-layer manner. At each layer, the segment assignment of each panel is handled independently. For each panel, we formulate global segment assignment using

regular routing patterns as an MWIS problem and solve it by an effective heuristic. We then apply a partial assignment technique to improve the utilization of each panel. If we do not reach the top horizontal or vertical layers, for the unassigned segments, we promote their terminals and defer their assignment to the upper layers. For the unassigned segments at the top layers, we propose a panel merging technique to provide more flexibility by assigning the segments in the merged panel. Maze routing is applied to further improve routability. After RegularRoute, we obtain a valid detailed routing solution with regular routing patterns satisfying the basic design rules. A local optimization phase is then proposed to satisfy the rest of the design rules. We propose local shift and rip-up-and-reroute in local optimization.

III. Local Net Routing

As the detailed routing problem we have formulated in Section II-A, the net or subnet that resides inside one G-cell is called a local net. In RegularRoute, local nets are routed before assigning global segments. The routing solutions of the local nets are marked as blockage in the following global segment assignment.

In this section, we first introduce local net routing by VSRT. We then demonstrate the topology that can better preserve routing resources to be used in global segment assignment. We will also provide techniques to resolve conflicts when routing multiple local nets. The alternative flows for processing local nets are discussed in the final part.

A. Vertical Spine Routing Topology

Spine routing has been proposed to predict the routing usage or interconnect properties at early design stages [15]. In this paper, we apply a topology, namely, the VSRT, to route the local nets. Consider one local net inside a G-cell. The x-coordinate of spine, i.e., the vertical trunk, is the median of the x-coordinates of all pins. The spine spans from the minimum y-coordinate to the maximum y-coordinate of all pins. The spine is routed using metal 2. We connect each pin to the spine with routes on metal 1, which we call a branch. Fig. 4 shows an example of the vertical spine routing tree.

VSRT can be easily built in a time complexity which is linear to the number of pins in a local net. In our test cases, the average local net pin count is relatively small (around 3). So the runtime is negligible compared to overall runtime.

Common practices to construct a routing topology include, for instance, rectilinear Steiner minimum tree (RSMT) and rectilinear minimum spanning tree. VSRT is applied to reserve routing resources on metal 2. In Fig. 4, VSRT and RSMT are compared for the same five-pin net. In metal 1, five horizontal tracks are blocked in both cases. In metal 2, only one track is blocked for VSRT, but three tracks are blocked for RSMT. As VSRT blocks fewer tracks on metal 2, global segment assignment would become easier for upper layers.

When building VSRT, the vertical spine coordinate is first determined. If the total pin count is odd, the vertical spine has only one location for minimum wirelength. Each branch has only one location to connect to the spine. If there are multiple local nets inside one G-cell, possible conflicts among these local nets can arise. To resolve these conflicts, we shift both spine and branches locally by trying the neighboring tracks. More specifically, we first determine the spine that is not in conflict with other local nets. Then branches of each local tree are connected to its spine. If there are persistent conflicts, we make a detour of some connection to avoid the conflict. If the simple local optimizing technique still cannot resolve the conflict, we utilize routing resources at higher metal layers (e.g., metals 3 and 4). In our experiments, most local nets can be routed only using metal 1 and metal 2. Detailed experimental results are presented in Section VI.

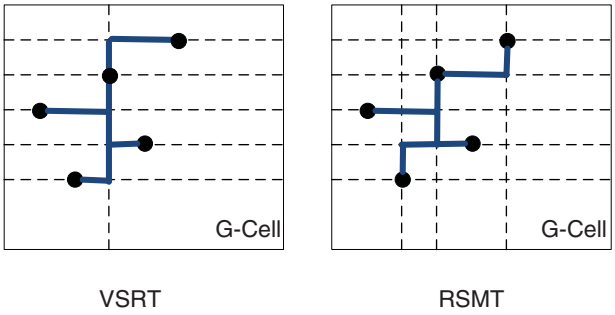

Fig. 4. Blocked track count for VSRT and RSMT.

B. Alternative Flows and Discussion

In RegularRoute, we route local nets before assigning global segments. There are alternative flows that process local nets simultaneously with global segments or route local nets after global segment assignment. There are reasons for our adopting the flow in RegularRoute (i.e., route local nets first), which are as follows.

1) Local nets have a lower level of flexibility than global segments. A local net typically contains three to four pins. The flexibility in terms of routing topology is less than for nets that span over multiple G-cells.
2) Local nets usage involves small portion of track usage and, if they are routed after global segment assignment, it would bring much more routing blockages to upper layers.
3) The via count would also be increased if local nets are routed later.
4) It is also possible to route local nets and global segments simultaneously by integrating local net routing into the MWIS framework in Section IV-A. The negative effect is the larger problem size. We observe that it is more desirable to perform local net routing before global segment assignment for both routability and computational complexity.

IV. Global Segment Assignment

In this section, we will present the technical details of global segment assignment. First, we present how we assign global segments with regular routing patterns in one panel. The problem is converted into an MWIS. The problem is solved by a fast and effective heuristic. Second, we discuss partial

assignment for better routing resource utilization. Third, the terminal promotion technique is discussed. It is an effective technique to defer the unassigned segments to the upper layers. Finally, we present the technique to handle unassigned segments on the top horizontal and vertical layers, and we develop effective panel merging to improve routability.

A. Global Segment Assignment for One Panel

In Section II-A, we mentioned that the global segment assignment problem for each layer is solved in a panel-by-panel framework. Solving the assignment problem in one panel is a fundamental component. In this subsection, we will investigate this problem. Without loss of generality, we consider horizontal panels (metal 1, metal 3, etc.). The panel is a collection of tracks inside one row of G-cells for the horizontal case. The concept is introduced to facilitate global segment assignment inside one row of G-cells, where horizontal segments have same y-coordinate. As introduced in Section II-A, a global segment is a horizontal or vertical route in 2-D global routing solution that spans G-cells in one row or column. The remaining portion of the net at either end of a segment is represented by a terminal. When a segment is assigned to a track, each of its ends should be connected to its associated terminal. A terminal can be a pin, a partially routed portion of a segment, or an attaching segment. An attaching segment is a segment that shares one of its terminals with the segment. The concept of terminal is illustrated in Fig. 5. In this figure, we show two assigned segments. The segment above is connected to a partial wire on the left and a pin on the right. The segment below is connected to an attaching segment that has not been assigned (dotted line) on the left and a pin on the right. The route between an assigned segment and its terminal is called a terminal connection route, or simply a terminal route. Note that in Fig. 5 we assume the pin and partial route are on the same layer with the segment, but this is not necessarily true. We will have more discussion on terminals in later subsections.

As the name suggests, regular routing suppresses the usage of routing bends and jogs. In RegularRoute, our method to assign a segment is to make it fully assigned to one track. We introduce the concept of choice for assigning a segment in regular routing. A choice is a valid candidate solution to assign a segment using a regular routing pattern when other global segments are ignored. A choice is determined by the track being used and the terminal connection being routed. In particular, a choice for a segment can be represented by (t, R). t is a track in the panel that the segment would be assigned to and R is the two-terminal connection route consisting of a collection of short wires and vias. A simple example is shown in Fig. 6. In this example, segment b has one choice $b1$, and segment a have two choices $a1$ and $a2$. Each choice specifies both the track and terminal connection routes. The terminal connection routes are denoted as $R1$, $R2$, and $R3$, respectively. For instance, terminal routes of choice $b1$ contain two short wires for both terminals. One major difference between our problem formulation and track routing [7] is that our formulation seeks to generate a valid detailed

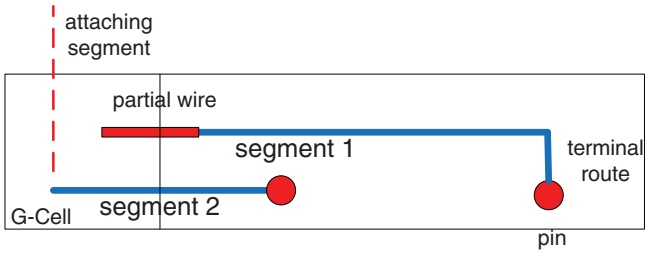

Fig. 5. Illustration of terminals: pin, partial wire, and attaching segment.

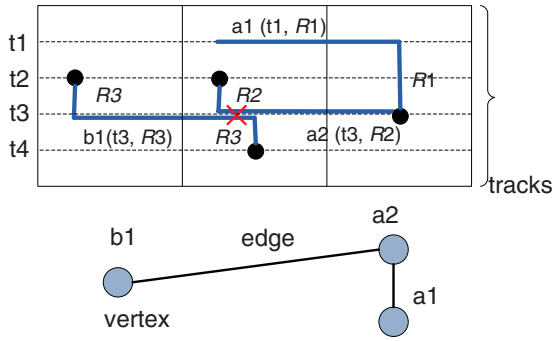

Fig. 6. Conflicting choices and the corresponding conflict graph.

routing solution for all nets, whereas a typical track routing formulation neglects terminal connection issue. When there is conflict between two choices, the two segments cannot be coassigned in the way specified by the two choices. For instance, there is a conflict between $b1$ and $a2$. It suggests that segment a and segment b cannot be assigned to track $t3$ simultaneously as specified in choices $b1$ and $a2$. It is also obvious that choices derived from the same segment should conflict with each other. For example, $a1$ conflicts with $a2$.

We formulate the global segment assignment problem for one panel as an MWIS problem. We model the conflicts among the choices by a conflict graph, as shown in Fig. 6. In the conflict graph, each choice is modeled as a vertex. Each conflict between two choices is modeled as an edge. Each vertex is assigned a weight specifying the priority of the assignment represented by the routing objectives. Vertices with higher weight need to be routed with higher priority. The problem is therefore converted to choose the set of independent vertices to maximize the total weight.

B. Weight Computation for Each Vertex

The weight assignment for each vertex is critical in the MWIS problem. In RegularRoute, it contains both factors to differentiate different segments as well as factors to differentiate the choices derived from the same segment. The first category of factors is utilized to weight different segments. And the second category is used to weight choices derived from the same segment.

In particular, we employ the following function to compute the weight for a vertex (choice):

$$W(v) = L + \alpha_1 \times \Omega + \alpha_2 \times \|R\| + \alpha_3 \times F$$
$$\Omega = \frac{\sum_{b \in B} (D_b)^2}{\|B\|}. \quad (1)$$

There are four major components in the function: L is the segment length measured by number of detailed routing grids, or more specifically, is equal to global routing length i.e., number of G-cells the segment travels times the number of detailed routing grids in one G-cell; Ω is the global segment density measured at each G-cell boundary; $\|R\|$ represents the total cost of the terminal connection route including short wires and vias for both terminals; F is the total flexibility length measured by number of detailed routing grids when the segment has terminal connected to the attaching segment. They will be introduced in more detail in the following context.

1) *Segment Length:* In (1), L represents the total number of detailed grids that the segment spans, which is equal to the number of G-cells times the detailed routing grids in one G-cell as shown in Fig. 7(a). RegularRoute encourages the assignment of longer segments, since longer segments are harder to assign in regular routing. If we assign short segments earlier, they are prone to creating more local blockages and thus impede the assignment of long segments later on. Another merit of this component is that we encourage the utilization of current panel/layer as much as possible to maximize the routability.
2) *Global Segment Density:* We define the notion of global segment density as the count of global segments crossing each G-cell boundary. When there is high global segment density, the segments are harder to be assigned because of the higher chance of conflict. One segment may pass a number of G-cell boundries, so we compute the average quadratic segment density. In Fig. 7(b), segment a passes through the first and second G-cell boundary, and segment b passes through the second and third G-cell boundary. Segment a is harder to be assigned since it travels through more congested G-cells. From another perspective, segment b is more flexible. If segment b is assigned earlier, segment a might become more restrictive to assign or even unassignable. In (1), B is the set of G-cell boundaries the segment passes. D_b is the density of boundary b. We sum the quadratic value of density and compute the average.
3) *Terminal Connection:* As introduced earlier in this section, terminal connection route stands for the collection of routes consisting of short wires and vias connecting the assigned segment to both of its terminals. For instance, $R1$, $R2$, and $R3$ in Fig. 6 represent terminal connection for segment $a1$, $a2$, and $b1$, respectively. The example is also illustrated in Fig. 7(c). The longer or more complicated the terminal connection route is, the more constraints it imposes to other segments. In order to compute the cost of terminal connection route, we divide the terminal connection usage into three categories: the preferred usage along routing tracks, the via usage, and the nonpreferred usage. Each category is weighted by a specific coefficient. For instance, if via count is critical, we charge a higher cost for the via usage in the terminal connection. In our experiment, the weight of via is bigger than the nonpreferred usage and the weight of preferred usage is the lowest. Actually, the exact value of each coefficient would bring no difference when we are trying to obtain the most cost-efficient route.

   There are various terminal connection routes. And the variations in terminal routes would combine with specific tracks to form an exponential number of choices. To simplify the solution space, we simply generate a terminal connection route by maze routing.[2] We generate the most cost-efficient routing path and use it as terminal connection route.
4) *Attaching Segments:* As noted earlier, the attaching segment is the one that shares one of its terminals with the segment. Since the two segments need to be connected eventually, the convenience of assigning the attaching segment (if has not been assigned yet) becomes another factor to weight each choice. Choices that entail more flexibility of assigning attaching segments in the future are granted higher priority. In Fig. 7(d), we show the assignment of one horizontal segment on the three tracks ($t1$, $t2$, and $t3$). The vertical segment in dotted line represents its attaching segment that has not been assigned. The three vertical tracks are the candidate tracks to assign the attaching segment ($t4$, $t5$, and $t6$). There are three tracks that can be used to assign the horizontal segment. But each track yields a different number of vertical tracks which the attaching segments can be assigned to without detour and short nonpreferred usage. Note that the short nonpreferred usage is detrimental to routability, as it potentially inhibits a number of tracks from being used. For instance, if $t2$ is used, the attaching segment can only be assigned to track $t4$ without detour and short nonpreferred usage. On the contrary, if the attaching segment is assigned to $t6$, then the route needs go up from the cross point of $t2$ and $t4$ using a via and then a nonpreferred route from $t4$ to $t6$ horizontally. The nonpreferred assignment would block all the three tracks $t4$, $t5$, and $t6$. If cases

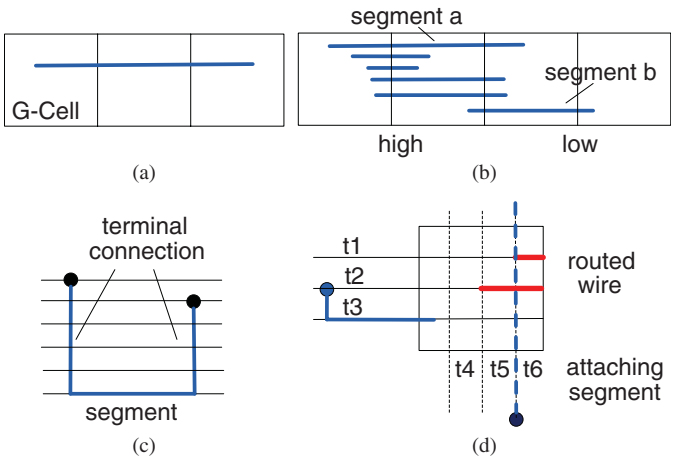

Fig. 7. (a) Segment length. (b) Global segment density for each G-cell boundary. (c) Terminal connection route. (d) Assignment of attaching segments.

[2]We could reduce the use of maze routing by terminal promotion, which will be discussed later.

like that frequently happens, the routability would be significantly degraded. Similarly, if $t1$ is used, there will be two tracks $t4$ and $t5$ assignable. The best one should be $t3$, where all the three tracks are assignable. Therefore, we use F as the total number of tracks that the attaching segments can be assigned for both terminals.

Each component is important to weighing the vertex. In general, the first two components are used to differentiate segments and the second two components are used to differentiate the choices derived from the same segment. In the equation, there are three coefficients (α_1, α_2, and α_3) to leverage the significance of each factor and they are experimentally determined. In our experiment

$$\begin{aligned} \alpha_1 &= 0.1 \times \left(\frac{\text{Avg}D_b}{T_p}\right)^2 \\ \alpha_2 &= 0.9 \times \frac{1}{W_p} \\ \alpha_3 &= 0.3 \times \frac{\text{Avg}S_p}{W_p}. \end{aligned} \tag{2}$$

In the above equations, AvgD_b stands for average global segment density of all G-cell boundaries in panel p. T_p is the number of tracks in the panel. W_p is the width of panel. AvgS_p is the average spacing between tracks in the panel.

C. Heuristic to Solve MWIS

The MWIS problem is NP-complete [16]. Solving it optimally takes a long time. (Typically there are hundreds of panels for each layer and each panel contains thousands of segments.) Alternatively, we develop an efficient and effective heuristic that explores the special structure of the conflict graph. To introduce the heurisitc, we first define the benefit $B(v)$ of assigning each vertex v

$$\begin{aligned} \text{B}(v) &= W(v) - \beta \times W_i(v) - \gamma \times W_o(v) \\ \beta &= 0.4 \times \frac{W(v)}{\max(W_i(v), W_o(v))} \\ \gamma &= 0.2 \times \frac{W(v)}{\max(W_i(v), W_o(v))} \end{aligned} \tag{3}$$

where $W(v)$ represents the weight of vertex v, W_i is the sum of weight for vertices with same-segment conflicts except v, $W_o(v)$ is the sum of weight for vertices with different-segment conflicts, and β and γ are parameters for leveraging their significance. In the experiment, β is set to be the fractional ratio of $W(v)$ against the maximum value of $W_i(v)$ and $W_o(v)$. γ is set to be half of β to increase the priority for different-segment conflicts.

In particular, W_i and W_o are defined with respect to the clique formed by vertices with same-segment conflicts in the conflict graph. W_i represents the sum of weights for vertices inside the clique except the vertex itself, while W_o denotes the sum of weights for all conflicting vertices (different-segment conflicts) outside the clique. The cliques for vertices with same-segment conflicts for two segments are illustrated in Fig. 8. In particular, vertices A, B, C, and D form clique1. And vertices E, F, and G form clique2. Our heuristic ranks

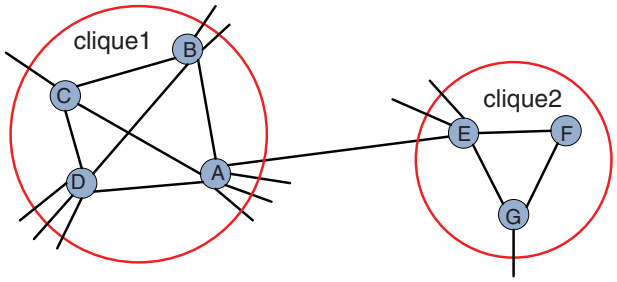

Fig. 8. Cliques formed by choices from two segments in the conflict graph.

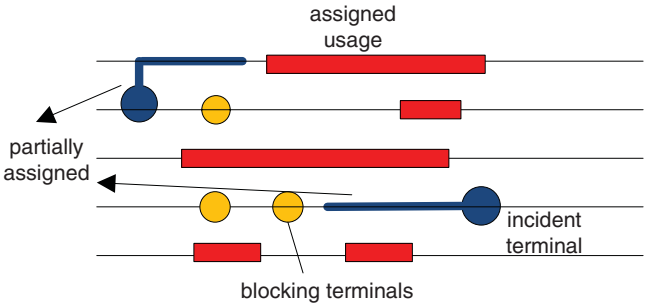

Fig. 9. Partial assignment technique for unassigned segments.

vertices in the order of decreasing benefit. The vertex with maximum benefit is chosen the corresponding global segment is assigned. After assigning each vertex, we update the conflict graph by deactivating all its conflicting vertices. The sum of weight W_i and W_o of vertices are also updated. To facilitate faster runtime, we utilize a heap data structure to store the benefit of all vertices.

Vertices with larger weights and smaller sum of weights for same-segment conflicts and different-segment conflicts are granted higher priority. First, vertices with larger weights indicates that the segment has higher criticality of assignment as discussed in the last subsection. Second, a larger sum weight of same-segment conflicts indicates that the segment is more flexible, since the segment has more choices to be assigned. Its assignment can be deferred so that more restrictive segments are considered earlier. Third, vertices with a larger sum weight for different-segment conflicts indicates that the choice incurs more conflicts and such an assignment should be disencouraged.

D. Partial Assignment

After solving the MWIS problem in one panel, there are potentially a large number of unassigned segments, especially for lower layers. In order to achieve better routing resource utilization, we explore the assignment of segments beyond the simple regular routing patterns defined in the MWIS problem. We implement a partial assignment technique to assign part of the unassigned segments.

For each unassigned segment, we assign part of the segment starting from its terminals. Longer unassigned segments are routed earlier. To evaluate a partial route, we employ the same computing function as in (1). We tentatively assign the partial route for the unassigned segment on all tracks in the panel. The partial route with the highest weight is selected as the formal partial assignment of the segment. The idea is briefly

illustrated in Fig. 9. Both terminals of the unassigned segment are on the current layer. The partial routes of represent the extension of wires from its terminals to the first blockage met. Note that in this example we assume the terminals are in the current layer. We will explain why this is true in later parts of paper.

It is possible for an unassigned segment to become fully assigned by two partial assignments from both terminals and the nonpreferred usage in between. In the experiment, we see it occur frequently, and the partial assignment technique greatly improves the utilization of routing resources of the panel. Besides this technique, it is alternatively feasible to incorporate the partial assignment idea into the MWIS problem. In particular, we could enumerate some partial assignment choices and take them in the algorithm. However, the extra vertices and edges could lead to much longer execution times. So we apply the partial assignment technique as a postprocessing stage after solving the MWIS for each panel.

E. Terminal Promotion

1) Terminals on Lower Layers: We defer the assignment of unassigned segments to upper layers. However, there are terminal connection issues when the segment is assigned in the upper layer but its terminals are located in lower layers. Suppose a segment is assigned to metal 5 and its terminals (pins) are located on metal 1. The terminal connection becomes challenging when routing resources on lower layers are restricted (e.g., routing tracks on lower layers are mostly taken up). In the traditional track routing problem [7], great efforts are made in rip-up-and-reroute to fix the terminal connection problems.

The method used in RegularRoute to handle the terminal connection problem is to promote terminals of unassigned segments to the next layer before handling the next layer. In this way, the terminals would be always localized on the same layer with the segment. The idea is more effective for congested panels where the routing resources are more restrictive. As stated earlier (Section IV-A), the main difference of RegularRoute and track routing [7] is that RegularRoute generates a valid detailed routing solution after assigning all segments. Yet, track routing may spend a lot of rip-up-and-reroute efforts on correcting a failed terminal connection.

In Fig. 10, we show how we promote the terminal to the next layer. The horizontal tracks are the routing tracks on the current layer and the dotted tracks are the tracks on the next layer. The vertical short lines in light color are candidate via locations. The terminal is located on track $t1$. We extend the terminal with a short wire on track $t1$ and use one via guiding up to the next layer.

2) Terminals on Upper Layers: In our problem, we assume all pins to be on metal 1. We have to promote the terminals after handling each of the metal layers. For the test cases with pins on the upper layers, we also need to promote the terminals of the assigned segments to the upper layers such that there is no connection problem between the assigned segments to their pins on the upper layers. Similarly, for the assigned segments with attaching segments that have not been

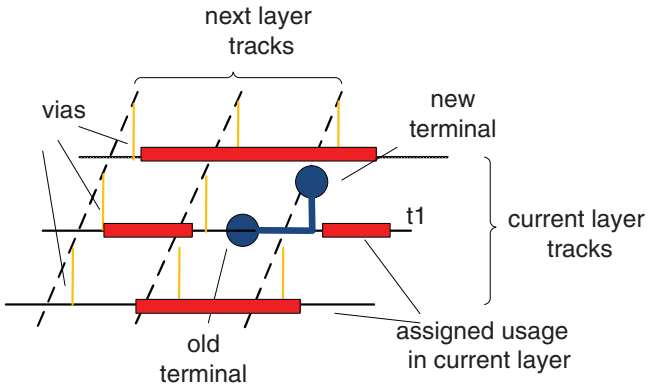

Fig. 10. Terminal promotion to avoid terminal connection failure.

assigned, we also need to promote the terminal of the assigned segment to upper layers to prevent the connection issue with its attaching segments.

We extend the terminal promotion to handle the upper layer pins and attaching segments for the assigned segment. The assigned segment is checked to see whether any of its terminals need to be connected to upper layer pins or attaching segments, and we promote its terminals. In theory, our algorithm is applicable to test cases with pins on different metal layers.

F. Unassigned Segments on Top Layers

For unassigned segments on the top horizontal (vertical) layers, there are no more metal layers to defer the assignment. To further improve the routability, we apply panel merging and eventually maze routing if the test case is congested.

All of our discussion till now was based on the assumption that each segment should respect the 2-D global routing solution. Or, in other words, the global routing solution determines the panel each segment should be assigned to. It is more flexible if one segment can be assigned in other panels given they have sufficient routing resources to accommodate the segment.

More specifically, for the panel with unassigned segments, we try to merge the panel with several neighboring panels and assign those segments in the merged panel. In the experiment, we merge the neighboring one panel (the merged panel has three panels). The number of neighboring panels merged can be adjusted depending on the level of hardness and runtime. The panel-merging technique is effective for the segments near the panel boundary. The merging of panels eliminates the boundary, and the segment becomes more flexible.

For very congested test cases, we propose to apply panel merging in the lower layers instead of waiting till the top layer. Actually, we could run the fast mode of RegularRoute (e.g., less choices) for an initial estimation and mark the panels with many unassigned segments. In the next round, starting from the bottom layer, we start panel merging for the congested panels. This method is a tradeoff between solution quality and runtime.

We eventually resort to a line probe maze routing for hard test cases. Maze routing is the most flexible routing technique but is detour-prone and time consuming. We adopt it as the last effort in RegularRoute.

G. Global Routing Solution With Layer Constraint

In RegularRoute, one of the inputs is the 2-D global routing solution; in other words, the global routing solution does not bear layer assignment. Each global segment can be assigned to any horizontal/vertical layers. Many industry routing tools generate 3-D global routing solution for alleviating the complexity of the job of detailed routing. In this part, we will discuss how we could extend current algorithm of RegularRoute to handle the 3-D global routing solution as well as the disadvantage of applying 3-D global routing solutions.

Different from the 2-D global routing solution, for each global segment in the 3-D global routing solution there is a specified layer range L_d, L_u. L_d specifies the lowest layer the segment can be assigned to, and L_u specifies the highest layer. In some cases, L_d equals L_u, which means that the segment can be assigned only to one certain layer. In our algorithm, we collect all unassigned segments for each layer and compute the weight for each segment. We will add one more component for layer constraint. We incorporate this component to evaluate the benefit of the assignment of a particular global segment. For instance, if a global segment can only be assigned to one layer, we need to increase the weight to make sure the segment can be assigned. On the contrary, if a segment does not have any layer constraint, its weight is kept as before since this segment is regarded as more flexible.

We adopt the 2-D global routing solution for more flexibility when assigning global segments: 1) there are more constraints in 3-D solution as the layer or layer range of each segment is specified, and the detailed routing problem is more restrictive to solve; 2) it is highly likely to encounter congestion issue due to more constraints; and 3) there is insufficient detailed and local information to perform layer assignment in global routing correctly, e.g., the global capacity can hardly capture the actual detailed routing bottlenecks.

V. Local Optimization for All Design Rules

After solving the problem in the regular routing phase by RegularRoute, we obtain the detailed routing solution for all nets considering the basic design rules (simple spacing rules for metal and via on each layer). However, it is required that the detailed routing solution satisfies all design rules. In this section, we will propose the general idea of local optimization based on RegularRoute for satisfying the rest of the design rules assuming that each design rule is a complex function of the local and neighboring geometries.

A. Space Reservation by Density Control

With all design rules, we propose a two-phase flow (regular routing and local optimization) to improve the efficiency of our algorithm. Respecting all design rules throughout the entire detailed routing seems to be a straightforward solution for one detailed router. However, considering the complexity and scale of the exponentially growing design rules, respecting all design rules might become unacceptably ineffective and inefficient. Besides, another drawback of obeying all design rules throughout detailed routing is that unnecessary detours of nets would be created to avoid design rule violations. Usually, this detoured usage exacerbates the routing congestion and introduces many more nontrivial routing patterns which degrade the solution quality and chip manufacturability. As discussed earlier, in the regular routing phase we focus on routing as many nets as possible applying the regular routing patterns with the basic design rules. RegularRoute is the algorithm we employ to obtain the routing solution within a short runtime. The generated solution is a valid detailed routing solution respecting the basic design rules.

In this section, we will focus on optimizing the solution based on regular routing to satisfy the rest of the design rules by local optimization. To achieve the purpose, we need to make sure that there is sufficient freedom for local adjustment for routing shapes. Here the word "freedom" specifies the free space around the routing shape, or, more specifically, is the density of routing usage in a region around the shape. "Density of region" stands for the ratio of total routing usage to the total routing resource (capacity). We need to control the routing density of each region to guarantee that there is sufficient free space.

To facilitate density control, we divide the entire layout on each layer into a number of regions. Each region contains $Q \times Q$ G-cells or T_r tracks. The total routing resource C_r in this region would become $T_r \times W_r$, where W_r is the width of the region. Then the density for each region would be (U_r/C_r) or $(U_r/T_r \times W_r)$. U_r is the total routing usage inside region r. To reserve space for local optimization, we control the density of each region in the regular routing phase. When applying RegularRoute, we make sure that the density of each region is below a user-specified maximum allowable density. In particular, we keep track of each region when assigning both local nets and global segments. When the maximum density is reached for one region, we no longer recommend assigning any new usage.

The value of the maximum density has a great impact on the success of assignment. Routing might become too hard to accomplish if a relatively small maximum density is specified. The experimental results will be shown in Section VI.

B. Local Optimization Techniques

Local optimization suggests applying adjustment locally for satisfying all design rules. We apply local shift as the main local optimization technique. For harder cases, we try rip-up-and-reroute.

Local shift suggests shifting the entire or part of the routing shapes to avoid design rule violations. Most design rules are a combination of local constraints of neighboring shapes. With sufficient reserved local space, it is feasible to satisfy all design rules inside the region by local shift. For instance, in Fig. 11, we show a partial solution after regular routing with four sets of shapes (specified S1 to S4) and five routing tracks. S1 and S3 are located on track $t3$, and S2 and S4 are located on $t2$ and $t4$, respectively. The solution is valid for the regular routing phase, as the routing usage is along the routing tracks and it respects basic spacing rules including metal-to-metal, metal-to-via, and via-to-via spacing. However, S1 contains a double-via enclosure. In some process, it introduces a larger

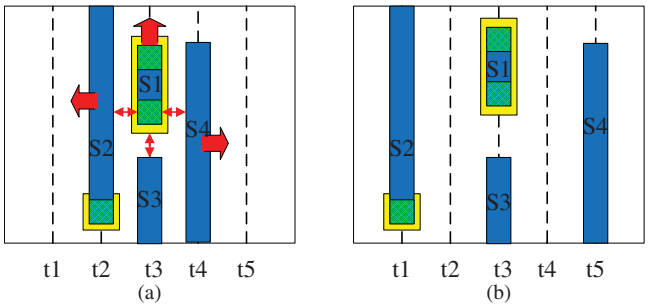

Fig. 11. (a) Before local shift is applied, there are three design rule violations due to double-via metal enclosure. (b) After local shift, no design rule violation exists.

TABLE I
RESULTS FOR LOCAL NET ROUTING ON ISPD98 TEST CASES

		VSRT				RSMT			
Name	#local nets	#unassign local	CPU (s)	metal 2 usage (k)	#unassign global	#unassign local	CPU (s)	metal 2 usage (k)	#unassign global
ibm01	1081	0	0.04	6.3	0	0	0.02	9.6	0
ibm02	1750	0	0.09	12.8	0	0	0.04	15.3	0
ibm07	4479	0	0.18	22.3	0	7	0.05	32.6	5
ibm08	5539	0	0.23	27.8	0	0	0.11	39.6	0
ibm09	5429	0	0.20	28.2	0	9	0.08	37.9	0
ibm10	2984	0	0.27	17.4	0	0	0.12	29.4	1
ibm11	6983	0	0.26	38.9	0	4	0.07	50.1	7
ibm12	2433	0	0.32	14.5	0	0	0.12	26.8	0

spacing requirement than single vias. There are two run-to-run spacing violations introduced by each S1–S2 and S1–S4. And S1–S3 introduces one end-to-end spacing violation.

The double-via enclosure spacing rule is out of reach in regular routing. In the local optimization stage, we apply the local shift to simply move the sets of shapes to avoid the violations. As illustrated in Fig. 11, we shift S2 left to track $t1$ and shift S4 right to track $t5$ to avoid run-to-run spacing violations. And we shift S1 up to avoid end-to-end spacing violations. This is only a simple example how local shift works to restore the violations of the rest of the design rules. Addtionally, the idea could be extended to better handle the emerging design for manufacturability (DFM) issues such as double patterning [17], [18] and resolution enhancement techniques [19]. The methodology can also be applied to design tools in other design phases to obtain more even routing utilization [20]–[22].

However, if there is insufficient local space around, local shift may not be always effective. When local shift cannot work, we apply local rip-up-and-reroute for satisfying all design rules inside the region. Different from the rip-up-and-reroute we proposed in RegularRoute, we aim at design rule satisfaction in addition to routability.

VI. EXPERIMENTAL RESULTS

All our experiments were performed on a machine with a 2.67-GHz Intel Xeon CPU and 32 G memory. We derive two sets of detailed routing test cases from ISPD98 [12] and ISPD05/06 [13], [14] placement benchmark suites.

A. Generating Grid-Based Detailed Routing Test Cases

In the original ISPD98 placement benchmarks, pins are located at the center of each standard cell, so we develop a program to set the pin coordinates randomly inside the standard cell and make sure they satisfy the spacing requirement at the bottom layer. The size of each module in the derived test cases is the same as that of the IBMv2 [23] placement benchmarks. We use Dragon [24] to generate the placed test cases for ISPD98-derived test cases, and FastPlace 3.1 [25] for ISPD05/06-derived test cases. We derive the global routing test cases similar to the format defined by ISPD07/08 global routing benchmarks [26], [27]. We then use FastRoute 4.0 [28], [29] to route the global routing test cases and generate the 2-D global routing solution. Both the global routing test case and the 2-D solution are imported into RegularRoute. Because of the lack of available academic detailed routers, we compared our results with an industrial router, i.e., WROUTE. However, WROUTE does not recognize bookshelf placement format or the global routing test case format we use. We therefore converted the placed test cases by a publicly available conversion tool[3] to LEF/DEF format test cases which we could import into WROUTE. The utility by default generates the routing test case in LEF/DEF format for three layers. We edited the LEF file for each test case to the exact number of layers specified in the test case statistics (Tables II and III). We also updated the pitch size to exactly the same as the case for RegularRoute.

Although the test cases are in different formats, we made sure the basic information (i.e., pitch size, module size, routing region, routing layers) and basic design rules (metal-to-metal, metal-to-via, and via-to-via spacing) were identical for both test cases.[4] In our experimental results, we apply the same set of coefficients of RegularRoute for all test cases.

Note that, in our experimental results, it is not a direct result comparison since in our flow we employ FastRoute 4.0 to generate the global routing solution and use RegularRoute for generating a detailed routing solution, while WROUTE completes both global routing and detailed routing. However, it is still a meaningful comparison since global routing is not a challenging part for both our flow and WROUTEs. And WROUTE is believed to perform a decent job in global routing. Hence global routing should not be a major differentiating factor in the experimental comparison.

B. Results of Local Net Routing

We first show RegularRoute's performance on handling local nets based on the vertical spine routing topology. We report the final unassigned local nets, total CPU time, final metal 2 usage, and unassigned global segment count. Here we only use metals 1 and 2. We compare our results with the RSMT topology.

[3]We use PlaceUtil executable developed by the University of Michigan, http://vlsicad.eecs.umich.edu/BK/PlaceUtils/bin/Sol64.

[4]In the experimental result, we only generate solutions with basic spacing rules.

TABLE II
RESULTS OF REGULARROUTE AND WROUTE FOR ISPD98 TEST CASES

Name	Test case statistics								FR4.0	RegularRoute				WROUTE			
	#Nets	#Layers	Grid	#Seg.	#Loc. Nets	Avg. Deg.	Max. Seg.	Max. Pin	CPU (s)	#unass-igned	CPU (s)	via ×10e5	wlen ×10e5	viol.	CPU (s)	via ×10e5	wlen ×10e5
ibm01	11 507	4	133×132	42 307	1081	3.85	238	463	0.47	0	3.17	0.84	6.9	0	47	0.84	7.1
ibm02	18 427	4	152×151	80 891	1750	4.23	366	616	2.71	0	14.4	2.9	15.9	3	155	3.0	16.1
ibm07	44 394	4	229×228	162 009	4479	3.7	507	877	8.51	0	34.3	3.8	39.9	12	190	3.8	40.6
ibm08	47 944	4	239×238	198 188	5539	4.13	568	1042	10.1	0	54.6	4.4	44.5	0	193	4.4	44.1
ibm09	50 393	4	243×242	179 942	5429	3.73	509	1016	6.11	0	43.1	3.9	37.0	0	184	3.9	37.4
ibm10	64 227	4	316×315	282 041	2984	4.19	640	1045	8.97	0	66.9	6.0	68.5	0	290	6.2	69.5
ibm11	66 994	4	276×275	230 365	6983	3.54	637	1140	15.7	0	68.1	4.8	53.2	23	287	5.1	53.8
ibm12	67 739	4	341×340	336 106	2433	4.34	736	1151	25.4	0	112.1	7.0	97.4	9	422	7.2	98.3

TABLE III
RESULTS OF REGULARROUTE AND WROUTE FOR ISPD05/06 TEST CASES

Name	Test case statistics								FR4.0	RegularRoute				WROUTE			
	#Nets	#Layers	Grid	#Seg.	#Loc. Nets	Avg. Deg.	Max. Seg.	Max. Pin	CPU (s)	#unass-igned	CPU (s)	via ×10e6	wlen ×10e7	viol.	CPU (s)	via ×10e6	wlen ×10e7
adaptec1	219 243	6	893×892	988 418	54 374	4.28	1424	2594	141	0	622	1.5	8.4	0	1201	1.5	8.5
adaptec2	257 659	6	1174×1172	1 040 019	44 356	4.09	1533	3065	189	0	558	1.9	10.2	221	1344	2.0	10.4
adaptec3	466 293	6	1935×1946	1 887 820	44 356	4.01	2142	4950	342	0	1176	3.5	21.8	0	3939	3.6	22.1
adaptec4	515 300	6	1933×1945	1 812 333	85 000	3.70	1884	3820	289	4	1330	3.0	19.8	324	4424	3.2	20.4
adaptec5	867 344	6	1935×1946	3 506 216	135 795	3.99	2203	4518	698	14	2844	6.9	46.6	294	7729	7.2	47.2
newblue1	331 106	6	934×932	1 070 792	69 300	3.68	1442	2957	72	0	297	2.3	8.8	0	914	2.4	9.1
newblue5	1 257 334	6	2122×2132	4 515 965	238 712	3.87	2521	5957	702	6	2654	7.2	46.3	287	7097	7.8	48.8
newblue6	1 286 448	6	2310×2318	4 944 944	208 903	4.09	2718	5238	598	0	2445	8.5	39.9	0	6645	9.0	41.2
bigblue1	282 399	6	893×892	1 182 506	25 288	4.02	1410	2534	134	0	811	2.2	9.8	0	1802	2.2	9.7
bigblue2	576 618	6	1560×1568	1 826 150	92 945	3.60	1648	3804	249	0	1177	3.7	21.2	54	2856	3.9	22.0

In Table I, the first column lists all experimental test cases. Owing to limited space, we only show the results for the ISPD98-derived test cases. The next column shows the total number of local nets for each test case. The following six columns show the results of VSRT and RSMT, respectively. The RSMT is generated by FLUTE [30] using default settings. First, #unassign local is the final unassigned local nets. VSRT has no unassigned local nets. But RSMT incurs some unassigned nets. Second, CPU is the runtime in seconds. FLUTE runs faster than our algorithm. But the local net routing runtime is trivial compared with global segment assignment. So the runtime advantage is not important. Third, metal 2 usage is the total usage on metal 2 after routing local nets. VSRT introduces 20%–30% less metal 2 usage, which saves more resources on metal 2. #unassign global is the final unassigned global segment when either topology is applied. RSMT may incur some unassigned global segments, and further suggests RSMT is inferior in preserving routing resources.

C. Results of RegularRoute for ISPD98 Test Cases

In Table II, we show the results for global segment assignment of RegularRoute on the eight ISPD98 test cases. We compare the results with WROUTE (version 3.0.61). Here we focus on showing the results for RegularRoute, hence there is no density control for each region as mentioned in Section V. The test case statistics are shown in the first seven columns, for total number of nets (#Nets), G-cell grids (Grid), total number of global segments (#Seg.), total number of local nets (#Loc. Nets), the average net degree for the whole netlist (Avg. Deg.), maximum number of segments (Max Seg.) in one panel, and maximum number of pins in one panel (Max Pin), respectively. These statistics provide an overall idea about the complexity of these test cases. The next column shows the runtime for FastRoute 4.0 [28]. The global routing runtime indicates how fast our detailed router is compared to the the global router. The following columns show the results of RegularRoute and WROUTE, respectively. #unassigned is the count of segments that cannot be handled by RegularRoute. CPU is the runtime in seconds. The WROUTE results are reported with similar metrics except viol., which is the number of design rule violations.

First, RegularRoute is capable of routing through all the eight test cases. WROUTE, nevertheless, can route four test cases without violation. Here the number of violation is the number of spacing rule violation caused by the inability to allocate the nets. Second, in terms of runtime, RegularRoute is better compared to WROUTE, which spends a lot of runtime on rip-up-and-reroute. Note that we have set up the same design rules, which include the basic spacing rules for metal and via on each layer. Besides the basic design rules, we could incorporate more design metrics into our framework. The weight function or cost function for solving the MWIS problem can be thus extended to incorporate other design objectives. In this case, potentially we see better chance for

TABLE IV
RESULTS OF EFFECTIVENESS FOR EACH PROPOSED TECHNIQUE ON ISPD98 TEST CASES

Name	Random		w/o Partial		w/o Terminal		w/o Merge		Ours	
	CPU (s)	#unass-igned	CPU (s)	#unass-igned	CPU (s)	#unass-igned	CPU (s)	#unass-igned	CPU (s)	#unass-igned
ibm01	2.94	0	3.06	6	2.85	10	3.17	0	3.17	0
ibm02	15.6	2	13.2	120	15.0	46	14.2	4	14.4	0
ibm07	36.0	0	31.2	223	36.2	26	33.8	8	34.3	0
ibm08	51.2	15	48.9	119	53.2	57	54.4	0	54.6	0
ibm09	47.8	3	40.2	46	42.1	20	42.5	0	43.1	0
ibm10	62.6	0	61.9	79	64.3	14	64.2	3	66.9	0
ibm11	69.7	8	63.4	92	66.5	22	67.1	0	68.1	0
ibm12	116.4	15	104.6	435	107.8	108	107.1	15	112.1	0

TABLE V
RESULTS OF REGULARROUTE FOR DENSITY CONTROL

Name	100%		95%		90%		80%	
	CPU (s)	#unass-igned	CPU (s)	#unass-igned	CPU (s)	#unass-igned	CPU (s)	#unass-igned
ibm01	3.17	0	3.24	0	3.85	0	4.66	6
ibm02	14.4	0	16.8	0	17.6	12	25.3	23
ibm09	34.3	0	37.5	6	43.2	26	60.0	52
ibm10	54.6	0	58.2	0	66.3	15	71.1	59
sum_98	106.5	0	115.7	6	131.0	53	161.1	140
norm_98	1.0	-	1.064	-	1.198	-	1.632	-
adaptec1	622	0	632	0	713	16	837	133
adaptec2	558	0	564	0	588	0	722	38
adaptec3	1176	0	1202	0	1288	0	1465	76
adaptec4	1330	4	1545	18	1660	45	1770	166
sum_0506	3686	4	3943	18	4249	61	4794	413
norm_0506	1.0	1.0	1.053	4.5	1.136	15.3	1.304	103.3

satisfying all design rules, as more and more complicated design rules are triggered by nontrivial routing patterns. The good routing completion rate could also save additional effort during the design rule clean-up stage and thus better manufacturability. Third, wlen stands for the sum of preferred usage and nonpreferred usage for all local nets and global segments. We achieve comparable results as with WROUTE. Forth, similar to our earlier discussion, we note that WROUTE employs both global routing and detailed routing in its framework. However, in our part, we import the solution of FastRoute 4.0 into RegularRoute. Though the comparison is not complete, it is still a meaningful one because the global routing in our experiments is not challenging. WROUTE does a decent global routing and it routes all nets without overflow in the global routing part. For more details regarding the runtime of each major step of RegularRoute, we break down the runtime on average normed to total runtime for all ISPD98-derived test cases in Fig. 12. We notice that most runtime is spent on processing lower metal layers when the unassigned segment count is high.

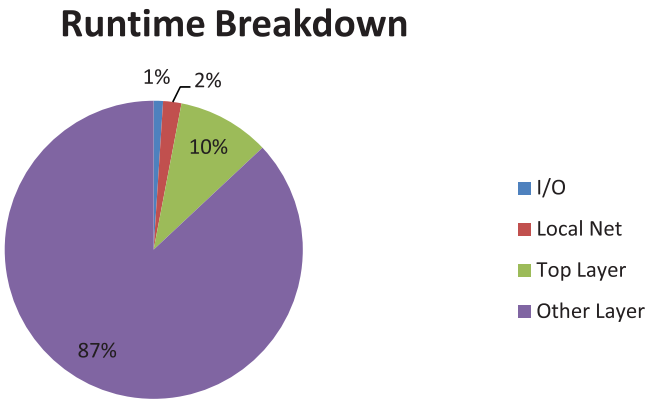

Fig. 12. Runtime breakdown for RegularRoute.

D. Results for ISPD05/06 Test Cases

In Table III, we show the complete results on 10 test cases derived from ISPD05/06 [13], [14] placement benchmarks. They are much bigger in problem size and more challenging in complexity than the ISPD98-derived test cases. We only show the results for 10 test cases (16 benchmarks in total) because the other 6 test cases could not be routed by WROUTE (because of some unknown machine or set-up issue). As mentioned earlier, these test cases are made by following a similar procedure as for the ISPD98 test cases. We use FastPlace 3.1 [25] to place all the placement benchmarks with default setting. Likewise, the results are also compared with WROUTE and here we do not employ density control. We show the number of unassigned segments (#unassigned), the CPU time, the via count, and total wirelength for RegularRoute. We additionally show the violation (viol.) count for WROUTE.

RegularRoute is capable of routing seven test cases without unassigned segments, and WROUTE can route five test cases. WROUTE is likely to incur a number of design violations.

E. Effectiveness of Proposed Techniques

We show the effectiveness of each technique we proposed for RegularRoute in Table IV. In this part, we show the experimental results for RegularRoute without each of the techniques we have proposed. In particular, Random denotes the adopted random ordering to process the vertices when solving MWIS for each panel, instead of the heuristic for ordering each segment choice based on weight and vertex connection. In the table, random ordering shows unpredictable behavior. It shows better results for three test cases with faster runtimes. But the general routability is degraded as there are five unroutable test cases. w/o Partial is RegularRoute without partial assignment, as discussed in Section VI-D. The routability of all test cases is degraded. Partial assignment is an effective technique to improve the routability of regular routing. w/o Terminal is RegularRoute without terminal promotion as discussed in Section VI-E. All test cases are still not routable, although the congestion degradation is mildly mitigated compared to the experiments without partial assignment. For the last comparison, w/o Merge represents RegularRoute without panel merging. The panel merging technique is helpful for congested test cases on the top metal layers. Since some test cases are not congested on the top layers with the rest of the techniques, the routability improvement is not as obvious as the last two techniques. Therefore, all the major techniques in RegularRoute are important to effectively and efficiently solve the regular routing problem for modern circuits.

F. Results of RegularRoute for Density Control

In Table V, we will present the results for RegularRoute with density control as discussed in Section VI-B. We show the results for four ISPD98 and four ISPD05/06 test cases for the impact of density control of routing usage for all regions. Note that the region consists of $Q \times Q$ G-cells. In the experiment, we use 4×4 G-cells to form one region. And experimental results will be shown for each test case on the count of unassigned segments and execution time for the maximum density of 100%, 95%, 90%, and 80%, respectively. sum_98 and norm_98 specify the sum and norm of the results for ISPD98 placement benchmark-derived test cases. Likewise, sum_0506 and norm_0506 specify the ISPD05/06 placement benchmark-derived test cases.

From the results, we notice that there is a minor impact for maximum density of around 95%. Seven test cases are routable given the density control for 95% out of the total capacity. If the control becomes more restrictive, more segments become unroutable. We notice that most of the unroutable segments are longer segments that could not be easily fitted in. As the trend continues, there will be major congestion if the density limit goes below 80%.

VII. Conclusion

In this paper, we proposed an effective detailed routing called RegularRoute for applying regular routing patterns. The algorithm proceeds in a bottom-up layer-by-layer manner. The problem for each layer was partitioned into subproblems by panels. Inside each panel, the global segment assignment problem was formulated as an MWIS problem. An effective heuristic and a few postprocessing techniques were developed. The generated solution by RegularRoute respects the basic design rules. To satisfy rest of the design rules, we applied local optimization based on local shift and rip-up-and-reroute. We showed the experimental results of RegularRoute on detailed routing test cases derived from academic placement benchmark suites. In the future, we intend to further improve the performance of RegularRoute and try to develop the approach we proposed in local optimization. We will also try to incorporate more design objectives to make our tool more suitable for industrial applications. In addition, we would be interested in making a parallel version of our tool for further runtime reduction.

Acknowledgment

The authors would like to thank Dr. H. Leung for the valuable discussions which inspired their work and the Computer-Aided Design Group, University of Michigan, Ann Arbor, for the helpful placement utility tools to convert the bookshelf placement format to the library exchange format/design exchange format.

References

[1] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. ACM/IEEE Design Autom. Conf.*, Jan. 1971, pp. 155–169.

[2] T. Yoshimura and E. Kuh, "Efficient algorithms for channel routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 1, no. 1, pp. 633–647, Jan. 1982.

[3] H. Shin and A. Vincentelli, "A detailed router based on incremental routing modifications: Mighty," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 6, no. 6, pp. 942–955, Nov. 1987.

[4] J. Cong, J. Fang, and K. Khoo, "DUNE-a multilayer gridless routing system," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 5, pp. 633–647, May 2001.

[5] Y. Chang and S. Lin, "MR: A new framework for multilevel full-chip routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 793–800, May 2004.

[6] G. Nam, K. Sakallah, and R. Rutenbar, "A new FPGA detailed routing approach via search-based Boolean satisfiability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 6, pp. 674–684, Nov. 2006.

[7] S. Batterywala, N. Shenoy, W. Nicholls, and H. Zhou, "Track assignment: A desirable intermediate step between global routing and detailed routing," in *Proc. Int. Conf. Comput.-Aided Design*, 2002, pp. 59–66.

[8] M. Ozdal, "Detailed-routing algorithms for dense pin clusters in integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 3, pp. 340–349, Mar. 2009.

[9] M. Gester, D. Muller, T. Nieberg, C. Panten, C. Schulte, and J. Vygen, "Algorithms and data structures for fast and good VLSI routing," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 2012, pp. 459–464.

[10] D. Pan, M. Cho, and K. Yuan, "Manufacturability aware routing in nanometer VLSI," *Found. Trends Electron. Design Autom.*, vol. 4, no. 1, pp. 1–97, Jan. 2010.

[11] L. Capodieci, "Layout printability verification and physical design regularity: Roadmap enablers for the next decade," in *Proc. EDPS*, 2006.

[12] *IBM-Place 1.0 Benchmark Suites*. (1998) [Online]. Available: http://er.cs.ucla.edu/benchmarks/ibm-place/

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

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

[15] H. Chen, C. Qiao, F. Zhou, and C. Cheng, "Refined single trunk tree: A rectilinear Steiner tree generator for interconnect prediction," in *Proc. ACM Int. Workshop Syst. Level Interconnect Predict.*, 2002, pp. 85–89.
[16] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.
[17] Y. Lin, Y. Ban, D. Pan, and Y. Li, "DOPPLER: DPL-aware and OPC-friendly gridless detailed routing with mask density balancing," in *Proc. Int. Conf. Comput.-Aided Design*, 2011, pp. 283–289.
[18] J. Gao and D. Pan, "Flexible self-aligned double patterning aware detailed routing with prescribed layout planning," in *Proc. ACM/SIGDA Int. Symp. Phys. Design*, 2012, pp. 25–32.
[19] D. Ding, J. Gao, K. Yuan, and D. Pan, "AENEID: A generic lithography-friendly detailed router based on post-RET data learning and hotspot detection," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 2011, pp. 795–800.
[20] Y. Zhang and C. Chu, "CROP: Fast and effective congestion refinement of placement," in *Proc. Int. Conf. Comput.-Aided Design*, 2009, pp. 344–350.
[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.*, Mar. 2012, pp. 597–602.
[23] *IBM-Place 2.0 Benchmark Suites*. (2004) [Online]. Available: http://er.cs.ucla.edu/benchmarks/ibm-place2/
[24] 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.
[25] N. Visvanathan, 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.
[26] *ISPD07 Global Routing Contest Benchmarks*. (2007) [Online]. Available: http://www.sigda.org/ispd2007/contest.htm
[27] *ISPD08 Global Routing Contest Benchmarks*. (2008) [Online]. Available: http://www.sigda.org/ispd2008/contest.htm
[28] 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.
[29] 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.
[30] C. Chu and Y. Wong, "FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 70–83, Jan. 2008.

Yanheng Zhang received the B.S. degree in electrical engineering and information science from the University of Science and Technology of China, Hefei, China, and the Ph.D. degree in computer engineering from the Department of Electrical and Computer Engineering, Iowa State University, Ames, in 2006 and 2010, respectively.

He is currently a Senior Member of the Technical Staff with the Placement Technology Group, Cadence Design Systems, Inc., San Jose, CA. He has authored or co-authored papers in conferences, such as the Design Automation Conference, the International Conference on Computer-Aided Design, and the International Symposium on Physical Design. His current research interests include very large-scale integration physical designs, specifically in algorithms for routing and congestion-driven placement.

Dr. Zhang was a recipient of the SIGDA/ACM ISPD Global Routing Contest Award in 2008.

Chris Chu received the B.S. degree from the University of Hong Kong, Hong Kong, in 1993, and the M.S. and Ph.D. degrees from the University of Texas at Austin, Austin, in 1994 and 1999, respectively, all in computer science.

He is currently an Associate Professor with the Electrical and Computer Engineering Department, Iowa State University, Ames. His current research interests include computer-aided design of very large-scale integration physical design, and design and analysis of algorithms.

Dr. Chu was a recipient of the IEEE TCAD Best Paper Award in 1999, the IEEE TCAD Best Paper Award in 2010, the ISPD Best Paper Award in 2004, the ISPD Best Paper Award in 2012, and the Bert Kay Best Dissertation Award from the Department of Computer Sciences, University of Texas at Austin, in 1998 and 1999. He is currently an Associate Editor of the IEEE TCAD and the ACM TODAES. He was on the technical program committees of several major conferences, including the Design Automation Conference (DAC), the International Conference on Computer-Aided Design, the International Symposium on Physical Design, ISCAS, DATE, ASP-DAC, and SLIP.