RegularRoute: An Efficient Detailed Router with Regular Routing Patterns

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

Abstract

Detailed routing is an important phase of realizing exact routing paths for optimizing various design objectives and satisfying increasingly complicated design rules. In this paper, we propose RegularRoute, a fast detailed router trying to use regular routing patterns in a correct-by-construction strategy for better routability and design rule satisfaction. Given a 2-D global routing solution and the underlying routing tracks, we generate a detailed routing solution in a bottom-up layer-by-layer manner. For each layer, the routing tracks are partitioned into a number of panels. We formulate the problem of assigning global segments into different tracks of each panel as a Maximum Weighted Independent Set (MWIS) problem. We propose a fast and effective heuristic to solve the MWIS problem. Then unassigned segments after MWIS 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 upper layers. At top layers, we apply panel merging and maze routing techniques to achieve better routability. Due to the unavailability of academic detailed routing benchmarks, we proposed two sets of detailed routing testcases derived from ISPD98 [1] and ISPD05 [2] placement benchmark suites respectively. The experimental results demonstrate the effectiveness and efficiency of RegularRoute.

1 Introduction

Because of the problem complexity, VLSI routing is usually divided into global routing and detailed routing. In the global routing stage, rough routing decisions are made based on G-Cell-to-G-Cell (e.g., global routing cell) connection on a global routing grid graph. Detailed Routing, on the other hand, realizes exact routing paths considering geometrical constraints based on the global routing solution. Detailed routing is an important stage in the sense that it is directly related to the routing completion and design rule satisfaction. It also impacts many design metrics such as timing, manufacturability, etc.

Detailed routing has been extensively studied since 70's (e.g., [3,4]) 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 [5]. However, such a sequential net-by-net approach is ineffective in handling congested designs and it usually creates unnecessary detour. DUNE [6] and MR [7] proposed to handle full chip gridless routing by similar multilevel approaches, in which the routing undergoes a coarsening phase and 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 which consider nets in a more simultaneous manner during detailed routing. Nam et al. [8] proposed a detailed router for FPGA based on Boolean satisfiability. Though this approach achieves good solution quality, the runtime is extremely long. Zhou et al. [9] 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 weighted bipartite matching based heuristic. 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 fails to connect different parts of a net. Mustafa [10] presented an insightful technique to perform escape routing for dense pin clusters, which is a bottleneck of detailed routing. A multi-commodity flow based optimal solution and a Lagrangian relaxation based heuristic are proposed. Nevertheless, the technique is not proposed to solve whole-chip scale detailed routing.

With diminishing feature size, many complex design rules are imposed to ensure manufacturability. It has been reported that for 32nm process, the number of rules reaches several thousands [11] and the design rule manual has roughly a thousand pages [12]. The dramatic increase in the number and complexity of the design rules makes detailed routing progressively complicated and time-consuming. We notice that many of those complex rules are triggered only by non-trivial routing patterns. Here we define regular patterns as those avoiding jogs and unnecessary detours as much as possible. Figure 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 will be 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 for both yield and routability. Moreover, regular routing introduces less vias, jogs and wirelength, and hence is better in terms of timing, signal integrity and power consumption. In order to reduce the implementation complexity and runtime of detailed routers and to improve the electrical properties, yield and routability of circuits, we propose to perform detailed routing based on regular patterns. Note that this 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. This paper shows 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 works, we present a fast and effective algorithm called RegularRoute to perform detailed routing with regular patterns.

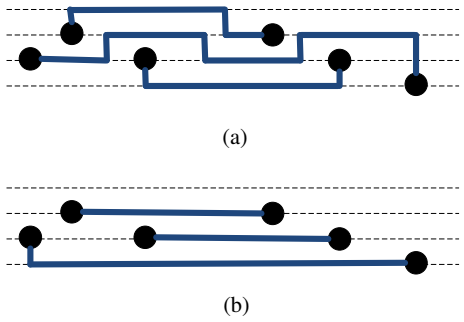

Figure 1: (a) Non-trivial routing patterns. (b) Regular routing patterns.

Novel techniques in RegularRoute are listed below:

- We introduce a new bottom-up layer-by-layer framework for detailed routing.
- We propose a single trunk V-Tree topology for routing nets local to a G-Cell.
- We decompose the routing problem of nets spanning multiple G-cells into assignment of global segments into panels. This approach facilitates parallel processing as assignment for different panels are independent of one another.
- We formulate the global segment assignment problem for each panel as a Maximum Weighted Independent Set (MWIS) problem. This formulation enables all segments to be considered simultaneously.
- We present a fast and effective heuristic to solve MWIS.
- We employ a technique to maximize the usage of a panel by partially assigning some of the remaining segments after MWIS.
- We introduce a terminal promotion technique to connect various segments of each net assigned to different layers.
- We present panel merging and maze routing techniques to handle unassigned segments at the top layers.

We implemented RegularRoute and tested its performance on detailed routing testcases derived from ISPD98 [1] and ISPD05 [2] placement benchmarks respectively. Experiments show that RegularRoute performs well in both quality and runtime.

The rest of paper is organized as follows: Section 2 provides the problem formulation and an overview of RegularRoute. Section 3 discusses the routing for local nets. In Section 4, we introduce techniques for handling global segments assignment. Experimental results are shown in Section 5.

2 Preliminaries

In this section, we will present some terminologies, the problem formulation and the algorithm flow of RegularRoute.

2.1 Terminologies and Problem Formulation

In this paper, as regular routing is considered, we model the routing resource as a 3-D regular grid graph. Each grid edge can accommodate one wire except for edges with blockage, which cannot be used. Each layer of the graph has a *preferred routing direction* and the preferred directions of adjacent layers are perpendicular to each other. We assume the preferred direction of lowest layer (metal1) 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 *non-preferred usage*. A sequence of unblocked grid edges along the preferred routing direction of each layer is called a *routing track*.

Assume a placed netlist with exact pin locations and a corresponding 2-D global routing solution are given. In this paper, we assume all pins are on metal1. The detailed routing problem is to route all nets on the grid according to the global routing solution such that routes of different nets do not intersect. The primary objective of detailed routing is to complete as many nets as possible. The secondary objectives include minimizing non-preferred usage, via count and wirelength. In industrial applications, there may be many other design metrics such as timing, crosstalk, yield, etc. These metrics can potentially be incorporated into our framework but they will not be handled directly in this paper.

In our framework, the global routing solution of each net is partitioned into a set of *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). Then detailed routing of global nets is formulated as assigning 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 tracks connected by short non-preferred 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. Figure 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 in a panel-by-panel manner.

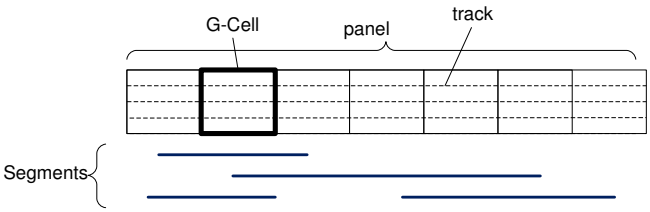

Figure 2: Definitions of track, segment and panel.

2.2 Algorithm Flow

We show the flow of RegularRoute in Figure 3. RegularRoute starts with extracting global segments by breaking the 2-D global routing solution. Then local nets are pre-routed using the single trunk V-Tree topology. In the following global segment assignment, the routed path of local nets are treated as blockages. Next, we perform global segment assignment in a bottom-up layer-by-layer manner. At each layer, the segment assignment of different panels are handled independently. For each panel, we formulate global segment assignment using regular routing patterns as a MWIS problem and solve it by an effective heuristic. After that, we apply a partial assignment technique to increase the utilization of the panel. Then if we have not reached the top horizontal or vertical layers, for the unassigned segments, we promote their terminals and defer their assignment to upper layers. For the unassigned segments at the top layers, we utilize a panel merging technique which provides more

flexibility in the assignment by allowing segments to deviate from the global routing solution. Finally, maze routing is applied for the residual unassigned segments.

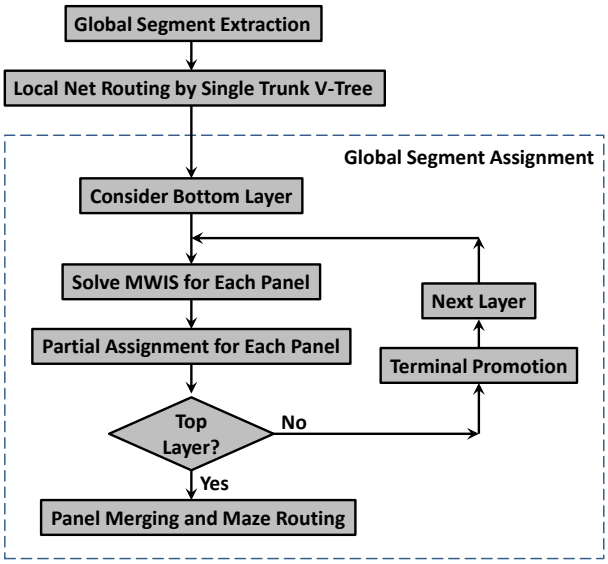

Figure 3: Flow chart for RegularRoute.

3 Local Net Routing

In detailed routing, the net or sub-net that resides totally 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 treated as blockages in the following global segment assignment. It is possible to route local nets and global segments simultaneously by integrating local net routing into the MWIS framework in Sec. 4.1. But in order to reduce the size of MWIS problems and hence the runtime of the whole algorithm, we choose to handle local nets before global segments.

In this section, we first introduce local net routing by single trunk V-Tree topology. We then demonstrate that single trunk V-Tree topology can better preserve routing resources to be used in global segment assignment. It is possible to have conflicts among the single trunk V-Trees of different nets. Hence we also present a branch and trunk shifting technique to resolve the conflicts.

3.1 Single Trunk V-Tree Topology

Single trunk tree has been proposed to predict the routing usage or interconnect properties at early design stages [13]. In here, we use a single trunk tree with the trunk being vertical to route the local nets. We call this topology *single trunk V-tree*. Consider all the pins inside a G-Cell. The x-coordinate of the trunk is set to be the median of the x-coordinates of all pins. The trunk spans from the minimum y-coordinate to the maximum y-coordinate of all pins. The trunk is on metal2. We connect each pin to the trunk with a metal1 (horizontal) connection, which we call a branch, and a via. Figure 4 shows an example of single trunk V-Tree.

Single trunk V-Tree can be easily constructed in time linear to the number of pins in a local net. In our testcases, the average pin count is very small (around 3). So the runtime is negligible compared to other steps.

There are many candidate topologies to construct the trees for local nets. For instance, RSMT and RMST are promising candidates.

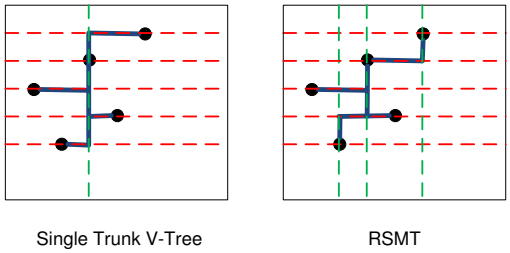

Figure 4: Track blockage count for single trunk V-Tree and RSMT.

The reason we choose single trunk V-Tree is for the sake of saving routing resources on metal2. In Figure 4, single trunk V-Tree and RSMT are presented for the same 5-pin net. In metal1, five tracks are blocked in both cases. In metal2, only one track is blocked for single trunk V-Tree, but three tracks are blocked for RSMT. As single trunk V-Tree blocks fewer tracks on metal2, global segment assignment will have more tracks to use later on.

3.2 Trunk and Branch Shifting

During the local net routing, we first determine the vertical trunk. If the total pin number is odd, the trunk has only one choice for minimum wirelength. Each branch has only one choice too. If there are multiple local nets inside one G-Cell, there is a risk of conflict among trees of different local nets. To avoid the conflict, we apply trunk and branch shifting by trying neighboring tracks. Any unresolved conflict can be resorted to higher layers (e.g., metal3 and metal4). But in our experiments, all local nets can be routed using only metal1 and metal2. The results will be shown in Section 5.

4 Global Segment Assignment

In this section, we cover the details of global segment assignment. We first present the detailed formulation of assigning global segments to a panel using regular routing patterns. The problem is then converted into a Maximum-Weighted Independent Set (MWIS) problem which is solved by a fast and effective heuristic. We next discuss the technique to perform a partial assignment for increasing the routing resource utilization of current layer. Then we talk about the terminal promotion techniques to defer the unassigned segments to upper layers. For the unassigned segments on the top layers, we develop effective panel merging and maze routing techniques for final routing closure.

4.1 MWIS based Solution for One Panel

In Section 2.1, we have mentioned the global segment assignment problem for each layer is solved by a panel-by-panel strategy. Solving the segment assignment problem of one panel is a fundamental component to the whole algorithm. In this subsection, we will investigate this problem. Without loss of generality, our examples only consider horizontal panels (metal1, metal3, etc.).

A global segment is a horizontal only or vertical only portion of a net extracted from the 2-D global routing solution. The remaining portion of the net in 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 partial assignment of the segment or a neighboring segment. The concept of terminal is illustrated in Figure 5. In this figure, we show two assigned segments. The first segment is incident to a partial segment on the left and a pin on the right. The second segment is incident to a neighboring segment (which has not been

assigned and is shown in dotted line) on the left and a pin on the right. The connection between an assigned segment and its terminal is called a terminal connection. Note that this example assumes the pin and partial wire are on the same layer with the segment, but it is not necessary to be true. We will have more discussion on terminals in later subsections.

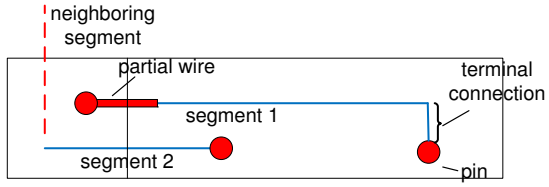

Figure 5: Illustration of terminal.

We introduce the concept of choice for assisting the assignment of a segment. 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 connections being made. In particular, a choice for a segment can be represented by (t, R). t is a track in the panel that the segment is assigned to, and R is a collection of short wires and/or vias that the assigned segment used to connect to its terminals. A simple example is shown in Figure 6. In this example, segment i has one choice c_i1, and segment j have two choices c_j1 and c_j2. Each choice specifies both the track and the short connections to the terminals. The terminal connections are highlighted as *R1*, *R2* and *R3* respectively. When two choices cannot co-exist, we said there is a conflict between the two choices. For instance, c_i1 conflicts with c_j2. Besides, different choices for the same segment mutually conflict with each other. For example, c_j1 conflicts with c_j2.

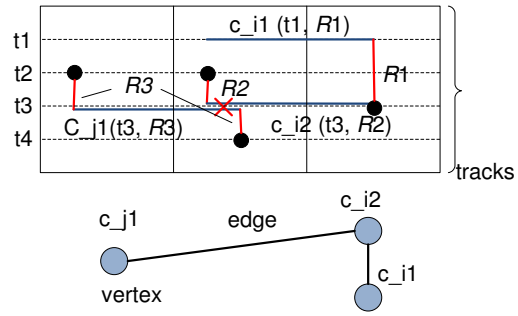

Figure 6: Conflicting choices and conflict graph.

We formulate the global segment assignment problem for one panel as a Maximum-Weighted Independent Set (MWIS) problem. We can represent the conflicts among the choices by a conflict graph as shown in Figure 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 benefit of the assignment. Then the problem is to select a set of independent vertices to maximize the total weight.

The weight calculation for each vertex is important in the MWIS problem. It contains several components for differentiating choices and leveraging various objectives. In general, it includes both the *segment differentiation* as well as the *choice differentiation*. The first one differentiates segments and all choices derived from the same segment will share the common weight of this part. The second one differentiates the choices derived from the same segment. We use the following function for weighting a vertex (choice).

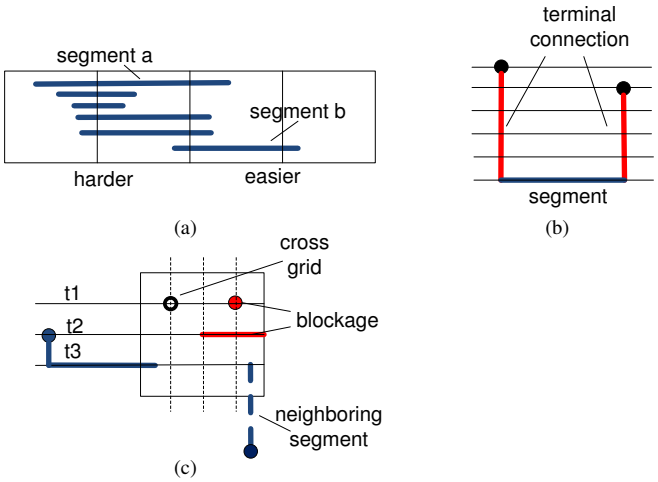

Figure 7: (a) G-Cell boundary density. (b) Terminal connection. (c) Flexibility component

$$W(v) = L + \alpha_1 \times \|R\| + \alpha_2 \times (\frac{\sum_{b \in B} (D_b)^2}{\|B\|}) + \alpha_3 \times (F1 + F2) \quad (1)$$

We have four major components for determining the weight of a choice.

1. *Segment Length*
   L is the number of global routing grids that the segment spans. It reflects length of the segment. In the weight calculation, we encourage packing more usage to current layer. This is a component for segment differentiation.

2. *G-Cell boundary density*
   This component is used to increment weight for the segments that cross dense G-Cell boundaries. We use the number of crossing segments to represent the boundary density. Intuitively, the segment passing through denser G-Cell boundaries is harder to assign. They can easily incur conflicts with other segments. We use the average quadratic G-Cell boundary density for this component. In Figure 7(a), segment a is harder to assign than segment b as it passes through denser G-Cell boundaries. In Equation 1, B is the set of G-Cell boundaries the segment passes through. D_b is the density of boundary b. We sum the quadratic value of density and divide it by the number of boundaries. This component is also proposed for differentiating segments.

3. *Terminal Connection*
   This component increases the weight for the choices with longer terminal connection route. The longer the terminal connection, the more likely the segment incurs potential conflicts with other segments. We divide the terminal connection usage to be three parts: preferred usage, via count and non-preferred usage. They are adjusted by different coefficients for leveraging their importance. For instance, if via count is critical, then we charge a higher cost for the number of vias in the terminal connection. For the sake of simplifying pool of choices, we generate terminal connection route by maze routing [1]. This component works for differentiating choices for the same segment.

[1] We could save the trouble of maze routing by terminal promotion, which will be discussed later

4. *Flexibility Component*
   The flexibility component is used to differentiate choices for the segment with one or more ends that are incident to neighboring segment which has not been assigned. The choice of current segment which offers more flexibility for assigning the neighboring segment will be better of routability. Since the direction of the segment and its neighboring segments will be perpendicular, the cross grid is the intersection of tracks between current layer and next layer. we define the flexibility count to be the number of cross grids that have access to upper layer inside the ending G-Cell. It indicates how much freedom of assignment for the neighboring segment. In Equation 1, we use $F1$ and $F2$ to represent the count for both ends. As in Figure 7(c), the flexibility count for track $t1$, $t2$ and $t3$ are 2, 1 and 3 respectively. Thus $t3$ is better in terms of the flexibility for assignment.

In the equation, there are three coefficients (α_1, α_2 and α_3) for tuning the importance of each component. Their exact value are determined by experiment.

The MWIS problem is NP-Complete [14]. Solving it optimally is time-consuming. (Typically there are hundreds of panels for each layer and each panel contains thousands of segments). Instead we develop an efficient and effective heuristic. We first define the cost for each vertex:

$$\text{Cost } C(v) = W(v) - \beta \times SumW_i(v) - \gamma \times SumW_o(v) \quad (2)$$

where W represents the weight of vertex v, $SumW_i$ is the inner sum weight, $SumW_o(v)$ is the outer sum weight, and β and γ are parameters. The inner and outer sum weight are defined with respect to the clique containing all choices for a segment in the conflict graph. The cliques for two segments are illustrated in Figure 8. Vertices A, B, C and D are choices for one segment and they form $clique1$. Similarly, vertices E, F and G are choices for another segment and they form $clique2$. For each vertex, inner (outer) sum weight is the total weight of its neighboring vertices inside (outside) its corresponding clique. The heuristic to find MWIS selects vertices in the order of decreasing cost. When a vertex is selected, we update the graph by removing all of its neighboring vertices (i.e., vertices conflicting with the selected vertex). The inner/outer sum weight of all vertices incident to the removed vertices are also updated. In order to improve the efficiency of extracting vertex with largest cost and cost update, we utilize a heap to organize the cost for each vertex.

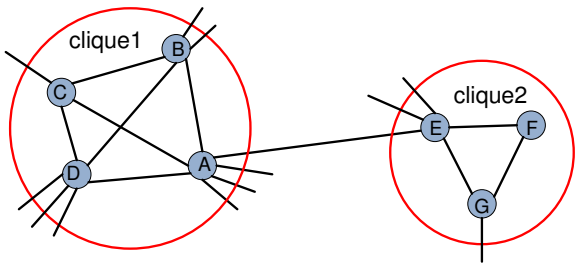

Figure 8: Cliques for two segments in the conflict graph.

Basically, vertex with larger weight and smaller inner/outer sum weight is prioritized. First, the vertex with higher weight is preferred since our algorithm tries to optimize the objective defined in Equation (1). Second, larger inner sum weight means more choices for the segment. Its assignment can be deferred so that routing resources are reserved for less flexible segments. Thirdly, the vertex with larger outer sum weight means more conflicts with other segments. Such choice is discouraged. The parameters β and γ are experimentally determined and the same values are used for all testcases.

4.2 Partial Assignment

After solving the MWIS problem by our algorithm, there are potentially large number of remaining unassigned segments in the congested panels. In order to better utilize the routing resources of current layer, we need explore more possibility beyond the choices defined in the MWIS problem. We implement a partial assignment for increasing the utilization of the panel.

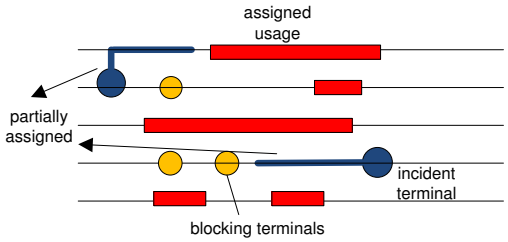

Figure 9: Partial assignment technique for unassigned segments.

For each unassigned segment, we will try assigning part of the segment starting from the incident terminals. We rank the unassigned segment based on the grid length (i.e., number of G-Cells passed) and process them in a decreasing order. There are potentially different options to perform partial assignment starting from one terminal by using different tracks. We use the same evaluating function as in Equation 1 to evaluate each option and select the best one. The idea is roughly illustrated in Figure 9. Both terminals of the unassigned segment are on the current layer. They are highlighted with bigger round shape than other blocking terminals. The assigned usage is treated as blockage as well. The partial assignment of the segment will be the route starting from the terminal to the first blockage met. Please note that in this example we have assumed the terminals are in the current layer. When we discuss the definition of terminal in the beginning of this section, it is not necessary to be true. But it is valid in our algorithm, the reason will be clear after we discuss the next subsection.

We also observe that it is possible to fully assign the segment using two partial assignment originating from each terminal and non-preferred connection to connect the two partial route. Actually the situation is not rare, the routing resources are better utilized and we will have more opportunity to assign more segments. Hence the partial assignment largely improve the efficiency of our algorithm. Besides our technique, an alternative idea is to incorporate the partial assignment into the MWIS problem. In particular, we could enumerate some partial assignment choices associated with each segment. However, it introduces much more vertices and edges and the resulting runtime can be significantly increased. Moreover, it is likely that the routing solution might no be regular routing oriented. So we only apply the partial assignment as postprocessing after solving MWIS problem.

4.3 Terminal Promotion

For the unassigned segments after applying partial assignment, we need defer their assignment to upper layers. However, there will be terminal connection issue when the segment is assigned in upper layer while its terminals are located in lower layers. Let's suppose a

horizontal segment is finally assigned to metal5 and one of its terminals is pin which is on metal1. In this case, the terminal connection might not be successful when the routing resource is limited (i.e., nearby routing tracks are taken up), realizing this connection can be headache and may finally results in big effort in rip-up and reroute.

After promoting terminals up, in the assignment in upper layer, we could always treat the terminals as in current layer. The idea could be highly effective for congested panels where the routing resources are limited. Hence the main difference of our algorithm and the track assignment [9] is that ours could guarantee a valid solution after all segments are successfully assigned. Yet track routing may spend a lot of rip-up and reroute effort for correcting failed terminal connection and segments between different layers.

We count the number of tracks that the terminal can access in the upper layer. If there is only one track that the terminal can access. We select the track and promote the terminal accordingly. If the terminal is capable of accessing multiple tracks, we pick the track that is closest to the terminal. If there is no access to upper layer, we rip-up the usage that is small and close to the terminal to guarantee it has at least one access. We use a terminal connection route to promote the terminal. In the upper layer, we create a virtual stripe covering all the tracks in the upper layer that the original terminal can access (virtual means we do not assign actual wire). If the segment is eventually assigned to a track that is not the same as the track we promote the original terminal to, we just rip-up the original terminal connection and correct it.

In Figure 10, we show how we promote the old terminal to upper layer. The horizontal tracks are current layer routing tracks and the dotted tracks are tracks on the upper layer. The vertical short lines in light color are potential via location. The original terminal is located on track $t1$. To promote the old terminal, we extend it with a short wire on track $t1$ and then add the via. The new terminal is shown on the tracks of upper layer.

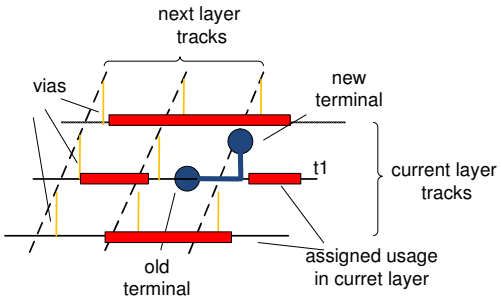

Figure 10: Terminal promotion to avoid terminal connection failure.

Overall, the terminal promotion is a highly effective technique in RegularRoute. A valid detailed routing solution is generated after global segment assignment.

4.4 Unassigned Segments on Top Layers

For unassigned segments on the top two layers (top two layers with horizontal and vertical tracks respectively), there are no upper layers where we can defer the assignment to. For better routability, we apply panel merging and eventually maze routing if the testcase is too hard.

All of our discussion has been based on the assumption that each segment respects the global routing solution. More specifically, the input global routing solution determines which panel each segment must be assigned. This assumption is restrictive in the fact that it forbids the option of trying alternative panel for better routability.

More specifically, for the panel with unassigned segments, we try to merge it with neighboring few panels and redo the assignment. In the case when the neighboring panel has space for holding more segments, it is likely the problem can be resolved. In the experiment, we merge the neighboring one panel (total three panels). This number can be modified depends on the level of hardness and runtime. The panel merging technique is effective for the segments with preferred tracks near the panel boundary. The merging of panels eliminate the boundary and the segment becomes more flexible.

For the very hard testcases, we could apply the panel merging in lower layers instead of waiting till the top layer. Actually, we could run RegularRoute initially for estimation and record the panels that are congested (with unassigned segments). We then run RegularRoute again starting from scratch with the precaution of congested panels. The panels that are predicted congested could be merged with neighboring panels since the lower layers. Again, it is a trade-off between solution quality and runtime and we adopt this idea in the case when testcase is too difficult to handle.

If there are still unassigned segment left, we eventually resort to a line probe maze routing technique or a full 3-D maze routing technique. The maze routing is most flexible technique but detour-prone and time-consuming. We adopt it as the last effort in RegularRoute.

Besides maze routing technique, we could also try academic MWIS solver in order to better solve the problem. However, the near-optimal solver such as [15] will be slow in nature. In order to maintain the fast runtime, we keep the fast heuristic as the main solver.

5 Experimental Results

All our experiments are performed on a machine with 2.67GHz Intel Xeon CPU and 32G memory. We derive two sets of detailed routing testcases from ISPD98 [1] and ISPD05 [2] respectively. In original ISPD98 placement benchmarks, pins are set to be at the center of each standard cell, we develop a program to randomly set the pin coordinate and make sure they satisfy the spacing requirement at the bottom layer. The size of each module in the derived testcases is the same to that of the IBMv2 [16] placement benchmarks. We use Dragon [17] to generate the placed testcases for ISPD98 derived testcases and FastPlace 3.1 [18] for ISPD05 derived testcases. We derive the global routing testcases similar to the format defined by ISPD07/08 global routing benchmarks [19, 20]. We then use FastRoute 4.0 [21] to route the global routing testcases and generate the 2-D global routing solution. Both the global routing testcase and the 2-D solution are imported into RegularRoute. Due to the lack of available academic detailed routers, we compared our results with an industrial router - WROUTE. However, WROUTE does not recognize bookshelf placement format or the global routing testcase format we use. We therefore convert the placed testcases by publicly available conversion tool[2] to LEF/DEF format testcases which we can import into WROUTE. Although the testcases are in different formats, we make sure the basic information such as pitch size, module size, routing region, routing layers etc. are identical for both testcases.

5.1 Results of Local Net Routing

We first show RegularRoute's performance on handling local nets based on the single trunk V-Tree topology. We report the final unas-

[2]We use PlaceUtil executable developed by Umich, http://vlsicad.eecs.umich.edu/BK/PlaceUtils/bin/Sol64

signed local nets, total CPU time, final metal2 usage and unassigned global segment count. Here we only use metal1 and metal2. We compare our results with RSMT topology.

In Table 1, the first column lists all experimental testcases. Due to limited space, we only show the results for the ISPD98 derived testcases. The next column shows the total number of local nets for each testcase. The following six columns show results of single trunk V-Tree and RSMT respectively. The RSMT is generated by FLUTE [22] using default settings. First, $\#un.L.$ is the final unassigned local nets. Single trunk V-Tree has no unassigned local nets. But RSMT tree incurs some unassigned nets. Second, CPU is the runtime in seconds. FLUTE runs faster than our algorithm. But the local nets routing runtime is trivial compared with global segment assignment. So the runtime advantage is not important. Third, $metal2\ usage$ is the total usage on metal2 after routing local nets. The single trunk V-Tree introduces 20% to 30% less metal2 usage, which saves more resources on metal2. $\#un.G.$ is the final unassigned global segment if either topology is applied. RSMT may incur some unassigned global segments and it further suggests RSMT is inferior in preserving routing resources.

		Single Trunk V-Tree				RSMT			
Name	#Loc. Nets	#un. L.	CPU (sec.)	metal2 usage	#un. G.	#un. L.	CPU (sec.)	metal2 usage	#un. G.
ibm01	1081	0	0.04	6300	0	0	0.02	9600	0
ibm02	1750	0	0.09	12800	0	0	0.04	15300	0
ibm07	4479	0	0.18	22300	0	7	0.05	32600	5
ibm08	5539	0	0.23	27800	0	0	0.11	39600	0
ibm09	5429	0	0.20	28200	0	9	0.08	37900	0
ibm10	2984	0	0.27	17400	0	0	0.12	29400	1
ibm11	6983	0	0.26	38900	0	4	0.07	50100	7
ibm12	2433	0	0.32	14500	0	0	0.12	26800	0

Table 1: Results for Local Net Routing on ISPD98 Testcases

5.2 Global Segment Assignment Results for ISPD98 Testcases

In Table 2, we show the results for global segment assignment of RegularRoute on the eight ISPD98 testcases. We compare the results with WROUTE (version 3.0.61). The testcase 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 segment ($MaxSeg.$) in one panel and maximum number of pin in one panel ($MaxPin$) respectively. These statistics provide an overall idea about the complexity of these testcases. The next column shows the runtime for FastRoute 4.0 [21]. The global routing runtime gives the rough idea of how fast our detailed router compared with 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. We develop an internal checker to make sure the assigned segments respect the design rules we have specified (spacing rule). CPU is the runtime in seconds. The WROUTE results are reported with similar metrics except "$viol.$", the number of design rule violations.

First, RegularRoute is capable of routing through all the eight testcases. WROUTE, nevertheless, can route four testcases without violation. Here the number of violation is the number of spacing rule violation caused by inability to allocate the nets. Second, in terms of runtime, RegularRoute is better compared with WROUTE, which spends a lot of runtime on rip-up and reroute. Please note that it is likely that WROUTE incorporates more design objectives than ours, though we strived to turn off all non-relevant design objectives. However, the point we want to demonstrate is the restrictive regular routing is doable for good routing completion. As we have mentioned in earlier part, we could incorporate more design metrics into our framework. The weight function or cost function for solving MWIS problem can be thus extended to incorporate other design objectives. And we also see better chance for satisfying various design rules, as more and more complicated design rules are triggered by non-trivial routing patterns. The good routing completion rate could also save additional effort during the design rule clean-up stage and thus better manufacturability. Third, we achieve comparable results in terms of wirelength and via count. It is because RegularRoute tries to restrict the usage of detoured routing path. During the routing we also charge additional penalty for the regular routing choices with larger via count.

5.3 Global Segment Assignment Results for ISPD05 Testcases

We show the complete results on six testcases derived from ISPD05 [2] placement benchmarks. They are much bigger in problem size and more challenging in complexity than the ISPD98 derived testcases. We only show the results for six testcases (eight benchmarks in total) because the other two testcases cannot be routed by WROUTE (crashes during execution). As mentioned earlier, these testcases are made following the similar procedure of ISPD98 testcases. We use FastPlace 3.1 [18] to place all the placement benchmarks with default setting. Likewise, the results are also compared with WROUTE. 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 five testcases without unassigned segments, and WROTUE can route three testcases. WROUTE is likely to incur a number of design violations. Like the experiments for ISPD98 testcases, we tried our best to switch off unrelated design objectives. Based on the results, we show RegularRoute is also doing well for larger designs.

6 Acknowledgment

The authors would like to thank Dr. Hardy Leung for valuable discussions which inspire our work, and the University of Michigan CAD group for the helpful placement utility tools to convert bookshelf placement format to LEF/DEF format.

7 Conclusion

In this paper, we propose a detailed router which seeks to route global segments with regular routing patterns. The whole algorithm is based on a bottom-up layer-by-layer processing. The problem for each layer is partitioned into sub-problems by panels. Inside each panel, the global segment assignment is formulated as a MWIS problem. An effective heuristic and a few postprocessing techniques are developed. We have shown RegularRoute's performance on detailed routing testcases derived from real circuits. In the future, we would like to further improve RegularRoute's performance and incorporate more design objectives to make our tool more suitable for industrial applications. In addition, we are interested in making a parallel version of our tool for further runtime reduction.

References

[1] IBM-Place 1.0 benchmark suites. `http://er.cs.ucla.edu/benchmarks/ibm-place/`.

	Testcase Statistics							FR4.0	RegularRoute				WROUTE (Encounter)			
Name	#Nets	Grid	#Seg.	#Loc. Nets	Avg. Deg.	Max Seg.	Max Pin	CPU (sec.)	#unass-igned	CPU (sec.)	via ×10e5	wlen ×10e5	viol.	CPU (sec.)	via ×10e5	wlen ×10e5
ibm01	11507	133×132	42307	1118	3.85	238	463	0.47	0	3.17	0.84	6.9	0	47	0.84	7.1
ibm02	18427	152×151	80891	1616	4.23	366	616	2.71	0	14.4	2.9	15.9	3	155	3.0	16.1
ibm07	44394	229×228	162009	5691	3.7	507	877	8.51	0	34.3	3.8	39.9	12	190	3.8	40.6
ibm08	47944	239×238	198188	6905	4.13	568	1042	10.1	0	54.6	4.4	44.5	0	193	4.4	44.1
ibm09	50393	243×242	179942	6590	3.73	509	1016	6.11	0	43.1	3.9	37.0	0	184	3.9	37.4
ibm10	64227	316×315	282041	4329	4.19	640	1045	8.97	0	66.9	6.0	68.5	0	290	6.2	69.5
ibm11	66994	276×275	230365	8486	3.54	637	1140	15.7	0	68.1	4.8	53.2	23	287	5.1	53.8
ibm12	67739	341×340	336106	3810	4.34	736	1151	25.4	0	112.1	7.0	97.4	9	422	7.2	98.3

Table 2: Results of RegularRoute and WROUTE for ISPD98 Testcases

	Testcase Statistics							FR4.0	RegularRoute				WROUTE (Encounter)			
Name	#Nets	Grid	#Seg.	#Loc. Nets	Avg. Deg.	Max Seg.	Max Pin	CPU (sec.)	#unass-igned	CPU (sec.)	via ×10e6	wlen ×10e7	viol.	CPU (sec.)	via ×10e6	wlen ×10e7
adaptec1	219243	893×892	988418	54374	4.28	1424	2594	141	0	622	1.5	8.4	0	1201	1.5	8.5
adaptec2	257659	1174×1172	1040019	44356	4.09	1533	3065	189	0	558	1.9	10.2	221	1344	2.0	10.4
adaptec3	466293	1935×1946	1887820	44356	4.01	2142	4950	342	0	1176	3.5	21.8	0	3939	3.6	22.1
adaptec4	515300	1933×1945	1812333	85000	3.70	1884	3820	289	4	1330	3.0	19.8	324	4424	3.2	20.4
bigblue1	282399	893×892	1182506	25288	4.02	1410	2534	134	0	911	2.2	9.8	0	1802	2.2	9.7
bigblue2	576618	1560×1568	1826150	92945	3.60	1648	3804	249	0	1177	3.7	21.2	54	2856	3.9	22.0

Table 3: Results of RegularRoute and WROUTE for ISPD05 Testcases

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

[3] A. Hashimoto and J. Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proc. ACM/IEEE Design Automation Conf.*, pages 155–169, 1971.

[4] T. Yoshimura and E. Kuh. Efficient algorithms for channel routing. *IEEE Trans. on Computer-Aided Design*, 1(1):633–647, Jan 1982.

[5] H. Shin and A. Vincentelli. A detailed router based on incremental routing modifications: Mighty. *IEEE Trans. on Computer-Aided Design*, 6(6):942–955, Nov 1987.

[6] J. Cong, J. Fang, and K. Khoo. DUNE-a multilayer gridless routing system. *IEEE Trans. on Computer-Aided Design*, 20(5):633–647, May 2001.

[7] Y. Chang and S. Lin. MR: A new framework for multilevel full-chip routing. *IEEE Trans. on Computer-Aided Design*, 23(5):793–800, May 2004.

[8] G. Nam, K. Sakallah, and R. Rutenbar. A new FPGA detailed routing approach via search-based boolean satisfiability. *IEEE Trans. on Computer-Aided Design*, 21(6):674–684, Jun 2002.

[9] S. Batterywala, N. Shenoy, W. Nicholls, and H. Zhou. Track assignment: A desirable intermediate step between global routing and detailed routing. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 59–66, 2002.

[10] M. Ozdal. Detailed-routing algorithms for dense pin clusters in integrated circuits. *IEEE Trans. on Computer-Aided Design*, 28(3):340–349, March 2009.

[11] Danny Rittman. Nanometer DFM – the tip of the ice. Intelligence: From Science to Industry, Tayden Design newsletter, March 2008.

[12] Luigi Capodieci. Layout printability verification and physical design regularity: Roadmap enablers for the next decade. In *edp*, 2006.

[13] H. Chen, C. Qiao, F. Zhou, and C. Cheng. Refined single trunk tree: A rectilinear Steiner tree generator for interconnect prediction. In *Proc. ACM Intl. Workshop on System Level Interconnect Prediction*, pages 85–89, 2002.

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

[15] M. Halldorsson. Approximations of weighted independent set and hereditary subset problems. *Journal of Graph Algorithms and Applications*, 1(4):1–16, Apr 2000.

[16] IBM-Place 2.0 benchmark suites. `http://er.cs.ucla.edu/benchmarks/ibm-place2/`.

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

[18] N. Visvanathan, 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.

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

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

[21] 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.

[22] C. Chu and Y. Wong. FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design. *IEEE Trans. on Computer-Aided Design*, 27(1):70–83, Jan 2008.