CUGR: Detailed-Routability-Driven 3D Global Routing with Probabilistic Resource Model

Jinwei Liu, Chak-Wa Pui, Fangzhou Wang and Evangeline F. Y. Young
Department of Computer Science and Engineering, The Chinese University of Hong Kong
{jwliu,cwpui,fzwang,fyyoung}@cse.cuhk.edu.hk

***Abstract*—Many competitive global routers adopt the technique of compressing the 3D routing space into 2D in order to handle today's massive circuit scales. It has been shown as an effective way to shorten the routing time, however, quality will inevitably be sacrificed to different extents. In this paper, we propose two routing techniques that directly operate on the 3D routing space and can maximally utilize the 3D structure of a grid graph. The first technique is called 3D pattern routing, by which we combine pattern routing and layer assignment, and we are able to produce optimal solutions with respect to the patterns under consideration in terms of a cost function in wire length and routability. The second technique is called multi-level 3D maze routing. Two levels of maze routing with different cost functions and objectives are designed to maximize the routability and to search for the minimum cost path efficiently. Besides, we also designed a cost function that is sensitive to resources changes and a post-processing technique called patching that gives the detailed router more flexibility in escaping congested regions. Finally, the experimental results show that our global router outperforms all the contestants in the ICCAD'19 global routing contest.**

I. Introduction

Due to the complexity of the routing problem in the VLSI domain, it has been divided into two sub-problems - global routing and detailed routing. In global routing, the 3D routing region is compressed into a 3D grid graph, of which each edge represents multiple tracks or multiple vias. The capacities of the edges are the resources that a global router can make use of, or in another word, the number of nets that can go through the edge without causing overflow. Conventionally, a global router will produce a path for each net to connect all its pins without overflow, while minimizing the total wire length. The ICCAD'19 global routing contest [1] has introduced a new objective that a global router is required to produce a set of connected rectangular route guides, instead of a single path, which makes the task even more complex and challenging. To better bridge the gap between global routing and detailed routing, the state-of-the-art detailed router, Dr. CU [2]–[4] was adopted to use the generated route guides to perform detailed routing to evaluate the quality of the global router.

A common strategy of doing 3D global routing, as adopted by NCTU-GR 2.0 [5], NTHU-Route 2.0 [6], NTUgr [7] and FastRoute 4.0 [8], is to first compress the 3D grid graph into a 2D grid graph and perform 2D global routing. The obtained overflow-free 2D global routing solution is then projected back into the 3D space by layer assignment. This strategy has been shown by many high performance routers to be good at balancing runtime and routing quality. During the 2D global routing, routers often resort to pattern routing and/or monotonic routing [9] first to quickly generate an initial routing solution. Nets that are hard to route will then be handled by maze routing and may go through multiple iterations of rip-up and reroute. Various kinds of history costs are devised to enable the routers to avoid routing through historically congested regions. After all nets are routed, the segments of the 2D paths are assigned to different layers by a dynamic programming based layer assignment.

Another less common but powerful strategy is to directly route the nets in a 3D grid graph. For example, FGR [10] applies maze routing directly in the 3D routing space with a cost scheme based on discrete Lagrange multipliers, and GRIP [11] utilizes integer linear programming to select a good route for each net. These two approaches have exhibited good quality, but take prohibitive long time to finish due to the large solution space. On the other hand, MGR [12] adopts a multi-level technique, in which the grid graph is coarsened so as to perform 3D maze routing in a smaller space. Their method is demonstrated to achieve competitive routing quality within reasonable runtime.

In this work, we propose a detailed-routability-driven 3D global router with probabilistic resource model. The proposed router utilizes a probability-based cost scheme, and routes the nets directly in the 3D space with two main phases, 3D pattern routing and multi-level 3D maze routing. In 3D pattern routing, the nets are broken down into two-pin nets, and a dynamic programming based algorithm will route the two pin nets sequentially using L-shape patterns and stacking vias at the turns. In the multi-level 3D maze routing phase, the grid graph is coarsened to shrink the routing space, and maze routing is first performed in the coarsened space with an objective to find a routing region with the highest routability. A fine-grained maze routing will then search for a lowest cost path within the region. Besides, we also propose a new technique called patching to add some stand-alone route guides or patches onto the routed path to further improve the detailed routability. Note that, the source code of this work is released on Github[1].

Our main contributions in this work include:

1) A sophisticated probability-based cost scheme minimizing the possibility of overflow after detailed routing.
2) An optimal 3D pattern routing technique that combines 2D pattern routing and layer assignment, and is able to route a majority of the nets without overflow, even if only L-shapes are used.
3) A multi-level maze routing utilizes two levels of routing - a coarsened level that searches for a region with the best routability, and a finer level that searches for a lowest cost solution within the region.
4) A patching technique that adds useful route guides to further improve the detailed routability.

The rest of the paper is organized as follows: Section II introduces the problem formulation, evaluation metrics and terminologies. Our proposed algorithm will be discussed in Section III.

[1] https://github.com/cuhk-eda/cu-gr

Section IV presents the experiments and results. Finally, Section V will give the conclusion.

II. Preliminary

A. Problem Formulation

In traditional global routing problem, the 3D routing region is represented by a set of global routing cells (G-cells) obtained by a set of evenly distributed horizontal and vertical grid lines. A grid graph $G(V, E)$ can then be defined by treating each G-cell as a vertex ($v \in V$) and creating an edge ($e \in E$) between every two adjacent G-cells. We call the edge between two G-cells on the same layer *wire edge*, and the capacity of a wire edge is equal to the number of tracks that can go through the edge. The edge between two G-cells with the same 2D coordinates but on different layers is called *via edge*, and its capacity is put as infinity by many 2D routers (but not in this work).

Traditional global routing problem can be defined as, given a grid graph and a set of nets to be routed, find a path for each net such that all the pins of a net are connected without overflow and the overall wire length is minimized. However, in the ICCAD'19 global routing contest, the problem is formulated in a slightly different way. Rather than merely path finding, the global routers are required to produce a set of connected rectangle guides, each containing an integral number of G-cells, so that the detailed router can find a path within the guides to connect all pins of each net and minimize a give cost function.

B. Evaluation Metrics

In the past, the quality of a global routing algorithm is usually measured by the total wire length, the number of vias used and the number of overflows on the global routing edges. Although this evaluation method is fast and straightforward, it cannot reflect the detailed routability of the solution accurately. The discrepancy between the global routing solution evaluation and the detailed routing performance is large in many cases. In this work, we adopted the same evaluation method in the ICCAD'19 global routing contest. The output of the global router is a set of connected rectangular route guides which will be fed into an academic detailed router called Dr.CU, and the score is calculated based purely on the quality of the detailed routing solution according to the metrics in Table I.

Metric	Weight
length of wire	0.5
num of vias	4
length of wrong-way wire	1
num of off-track vias	1
length of off-track wires	0.5
length of out-of-guide wires	1
num of out-of-guide vias	1
num of min-area violations	500
num of spacing violations	500
num of short violations	500
short metal area / metal2 pitch	500

TABLE I: Evaluation Metrics for Detailed Routed Solution.

C. Terminologies

Our algorithm is designed based on three major concepts - capacity, demand and resource, all of which can be used to describe a wire edge or a GCell. For example, we have both the capacity of an edge and the capacity of a G-cell. Since all the following discussion will revolve around these three concepts, a brief explanation for each is given below:

1) Capacity: The capacity of a wire edge $e(u, v)$ denoted by $c(u, v)$ is the number of tracks going through that edge, which is also roughly the maximum number of nets that can utilize the edge. The capacity of a G-cell u, $c(u)$, is then defined as the average capacity of its two abutting wire edges.

2) Demand: The demand of an edge, $d(u, v)$, is the part of the capacity that is already used by the routed nets. It is the sum of two parts - demand by wires and demand by vias. The demand by wires is simply the number of wires going through the edge, while the demand by vias is an estimated number of tracks affected by the vias in the edge region. For instance, the demand of edge $e(u, v)$ is calculated using the the first equation in Equation (1a),

$$d(u, v) = wire(u, v) + 1.5 \times \sqrt{\frac{via(u) + via(v)}{2}}, \tag{1a}$$

$$d(v) = \frac{wire(u, v) + wire(v, w)}{2} + 1.5 \times \sqrt{via(v)}, \tag{1b}$$

where $wire(u, v)$ is the number of wires going through the edge and $vias(u)$ is the number of vias that have metals inside G-cell u. Note that a stacking via that enters a G-cell from below (above) and exit from above (below) is counted as two.

The demand of a G-cell, $d(u)$, is defined similarly. It is also the sum of the demand by wires and the demand by vias. However, the demand by wires is replaced by the average demand by wires of its two abutting edges, and the demand by vias only considers the vias in the G-cell, as shown in Equation (1b). Note that u and w are the two G-cells adjacent to v in the preferred routing direction.

3) Resource: The resource of an edge or a G-cell is the part of the capacity that can still be utilized by the nets to be routed. Its relationship with the capacity and demand is shown in Equation (2), in which the resource of an edge $e(u, v)$ is denoted as $r(u, v)$ and the resource of a G-cell u is denoted as $r(u)$.

$$r(u, v) = c(u, v) - d(u, v) \tag{2a}$$

$$r(u) = c(u) - d(u) \tag{2b}$$

Neither capacity nor demand can well describe the degree of congestion, but the remaining resource will be a good indicator of congestion and how big the capacity margin is.

III. Proposed Algorithm

A. Overview

Figure 1 presents the overall flow of the proposed algorithm, which can be divided into three parts: initial routing, multi-level maze routing and route guide generation. Note that, both intial routing and mutli-level maze routing operate in the 3D space.

B. Cost Scheme

Our carefully crafted cost scheme, taking wire length, possibility of overflow and ability of detailed router to solve overflow into consideration, is capable of minimizing wire length and via number, while avoiding overflow and reserving enough capacity for later use. To discuss the proposed cost scheme, note that a path of a net is formed by a set P of wire edges and vias, and the total cost of the

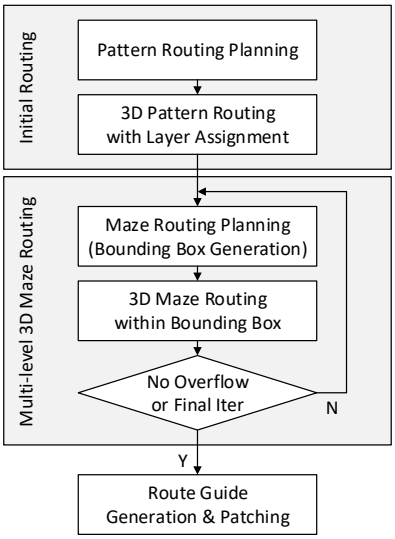


Fig. 1: Overall Flow of the Proposed Algorithm.

path will be the sum of the costs of all those edges as computed by Equation (3).

$$cost(P) = \sum_{e(u,v)\in P} cost_w(u,v) + \sum_{via(u,u')\in P} cost_v(u,u'), \quad (3)$$

where $cost_w(u,v)$ is the cost of a wire edge and $cost_v(u,u')$ is the cost of a via edge. The wire edge cost is comprised of two parts - wire length cost and congestion cost, and is computed using the formulae in Equation (4),

$$cost_w(u,v) = wl(u,v) + eo(u,v) \times lg(u,v), \quad (4a)$$

$$eo(u,v) = wl(u,v) \times \frac{d(u,v)}{c(u,v)} \times uoc, \quad (4b)$$

$$lg(u,v) = (1.0 + exp(slope \times r(u,v)))^{-1}. \quad (4c)$$

$wl(u,v)$ is the wire length of the wire edge (which is also its wire length cost, since the coefficient for wire length cost is one). The product of $eo(u,v)$ and $lg(u,v)$ forms the congestion cost, where $eo(u,v)$ is the expected overflow cost and $lg(u,v)$ is a logistic function of $r(u,v)$. The expected overflow cost is calculated by multiplying the wire length of the edge being considered by the possibility of overflow, $d(u,v)/c(u,v)$, and the unit length overflow cost uoc (a given constant). Note that the possibility of overflow, computed as the ratio between demand and capacity, is accurate if the detailed router adopts the simplest strategy of picking a track randomly to route. However, most well designed detailed routers will do much better than random selection. To handle this, we use a logistic function to model the ability of a detailed router to avoid overflow. The variable $slope$ of the logistic function is an adjustable parameter that determines the global router's sensitivity to overflow. The formation of the congestion cost of a wire edge is illustrated in Figure 2. When the resources are abundant, there is almost no congestion cost, but the cost will increase rapidly as the resources are being used up and will keep increasing almost linearly after all the resources are used. This is because when the resources are used up, the expected overflow cost will dominate the congestion cost.

Many global routers ignore the impact of vias before the layer assignment stage, thanks to our 3D pattern routing strategy, a via

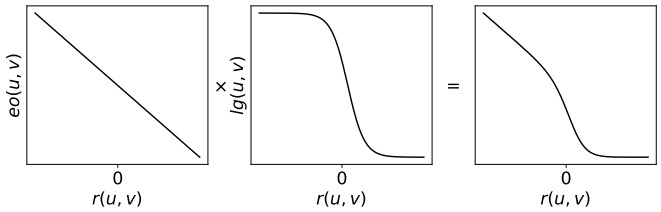


Fig. 2: Congestion Cost Composition.

cost scheme can be embedded to reflect the impact. Suppose u and u' are the lower and upper G-cells connected by a via, the via cost of $via(u,u')$ can be computed using Equation (5), where uvc represents the given unit via cost. The logistic function with the same $slope$ value is similar to that for wire edge, and can model the ability of the detailed router to avoid overflow in either G-cell u or u'. Note that for simplicity, no area overflow cost for vias is included.

$$cost_v(u,u') = uvc \times (1 + lg(u) + lg(u')), \quad (5a)$$

$$lg(u) = (1.0 + exp(slope \times r(u)))^{-1}. \quad (5b)$$

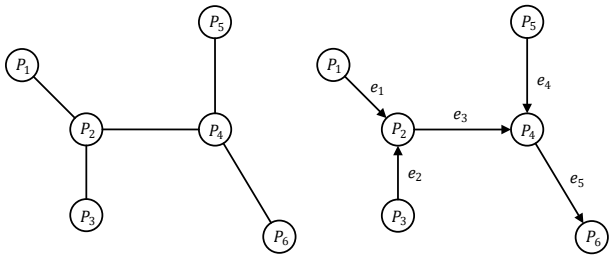
C. Initial Routing / 3D Pattern Routing

Traditional pattern routing generates 2D topologies only, while our proposed 3D pattern routing directly generates 3D topologies without the need of an extra layer assignment stage. Take L-shape routing as an example, traditional pattern routing will only choose between 2 possible paths for each two-pin net, and some of the paths may turn out to be impossible to be routed after layer assignment even if they did not cause any overflow in the 2D grid graph. However, the proposed 3D pattern routing combines 2D pattern routing and layer assignment. It chooses a path among $2 \times L \times L$ possible options for every two-pin net, where L is the number of layers. With the help of dynamic programming, an optimal solution with respect to our cost scheme on wire length and congestion is guaranteed to be found if it exists.

In order to perform 3D pattern routing, each multi-pin net will first be broken down into a set of two-pin nets in a step called pattern routing planning. In the same step, the order in which they are routed will also be determined. A dynamic programming algorithm is then conducted to pattern route the two-pin nets and minimize the overall cost.

1) Pattern Routing Planning: During the planning stage, we will first utilize FLUTE [13] to generate a rectilinear Steiner minimum tree (RSMT) for each multi-pin net so as to guide the pattern routing to produce shorter wire length. FLUTE is an RSMT construction algorithm adopting a look-up table approach, which is both fast and optimal for low-degree nets. However, FLUTE is unaware of routing congestion. We used a technique called edge shifting described in [14] to alleviate the problem. The multi-pin net will then be broken down into a set of two-pin nets in such a way that every two points sharing an edge in the RSMT will form a two-pin net.

Next, we will determine the order in which the two-pin nets will be routed. A degree-one node in the RSMT will be randomly picked as the root, and a depth first search (DFS) traversal will be performed to visit all the other nodes. The two-pin nets will be routed in the reverse order in which they are visited in this traversal. For example, suppose we pick P_6 of the RSMT in Figure 3a as the root, and the DFS traversal visits the nodes in the order of



(a) (b)

Fig. 3: Two-Pin Nets Ordering.

P_6, P_4, P_5, P_2, P_3 and P_1. We then label the two-pin nets in the reverse order as e_1, e_2, e_3, e_4, e_5 and e_6, which is also the order in which they will be routed. Note that each two-pin net will be routed from destination to source, and the arrows of the edges indicate the directions that they will be routed. We call the point P_i pointed to by an out-going edge from P_j the parent of P_j, and we say P_j is a child of P_i or $P_j \in ch(P_i)$.

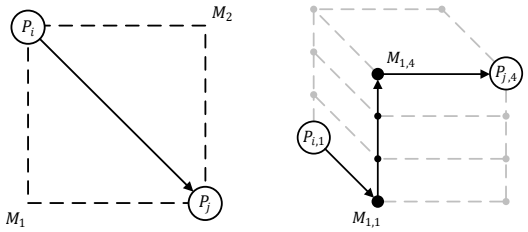
2) Dynamic Programming: Having the nets been broken down into two-pin nets and their order decided, a dynamic programming algorithm is designed to perform pattern routing and layer assignment simultaneously. Our dynamic programming algorithm is similar to that introduced in [15]. However, their algorithm only performs layer assignment after a net is pattern routed, while our algorithm combines the two steps so as to avoid loss of accuracy caused by compressing 3D grid graph to 2D.

First, we define $S(P_i)$ as the sub-tree including P_i and all its descendants, and $S(P_i, P_j)$ as the sub-tree including P_i, P_i's child P_j and all the descendants of P_j. Take the net in Figure 3 as an example, $S(P_2)$ refers to the sub-tree comprised of $\{P_2, P_1, P_3\}$ and $S(P_4, P_2)$ refers to the sub-tree comprised of $\{P_4, P_2, P_1, P_3\}$. Most importantly, as the building blocks of our dynamic programming algorithm, we define minimum sub-tree cost, $msc(P_i, l)$, as the minimum cost of routing the sub-tree $S(P_i)$ rooted at $P_{i,l}$, the G-cell on the l^{th} layer and at the position of P_i, and define $msc(P_i, P_j, l)$ as the minimum cost of routing the sub-tree $S(P_i, P_j)$ rooted at $P_{i,l}$.

For each two-pin net $P_i \rightarrow P_j$, we'll first calculate $msc(P_i, l)$ for $l = 1, ..., L$ using Equation (6), assuming there are totally L layers, and P_i has k children $\{P_{c(1)}, ..., P_{c(k)}\}$.

$$msc(P_i, l) = \min_{\substack{1 \le l_1 \le L \\ \dots \\ 1 \le l_k \le L}} (cost(V(P_i, l_1, ..., l_k, l)) + \sum_{j=1}^{j \le k} msc(P_i, P_{c(j)}, l_j)). \tag{6}$$

Note that in Equation (6), $V(P_i, l_1, ..., l_k, l)$ is the set of vias needed to connect all the k children sub-trees and pins to $P_{i,l}$ if they exist, and the cost function is simply the one described in Equation (3). For example, if a node P_i has k children and the pin itself is on the l_p^{th} layer, to connect them all to $P_{i,l}$ will need vias at P_i from the $min(l_1, ..., l_k, l_p, l)^{th}$ layer to the $max(l_1, ..., l_k, l_p, l)^{th}$ layer. If a node P_i has no children, $msc(P_i, l)$ will only consist of the via cost. After that, all possible 3D L-shapes to connect $P_i \rightarrow P_j$ will be considered to update



(a) (b)

Fig. 4: Two-Pin Net 3D L-Shape Routing.

$msc(P_j, P_i, l)$ for $l = 1, ..., L$ by,

$$msc(P_j, P_i, l) = \min_{\substack{1 \le l_i \le L \\ x=1,2}} (cost(Path(P_{i,l_i}, M_{x,l_i}, M_{x,l}, P_{j,l})) + msc(P_i, l_i)). \tag{7}$$

$Path(P_{i,l_i}, M_{x,l_i}, M_{x,l}, P_{j,l})$ is an L-shape path with two wire segments, $P_{i,l_i} \rightarrow M_{x,l_i}$ and $M_{x,l} \rightarrow P_{j,l}$, and a stack of vias, $M_{x,l_i} \rightarrow M_{x,l}$. The variable x specifies the two possible options of the L-shape route, and the turning position of the L-shape path is denoted by $M_{x,\_}$. Figure 4a shows the 2D structure of the two-pin net, and Figure 4b shows the 3D structure of one possible path. In the example, M_1 is chosen as the turning position. There are wires connecting $P_{i,1}$ and $M_{1,1}$ on the first layer, and wires connecting $M_{1,4}$ and $P_{j,4}$ on the fourth layer. The two layers are connected by three vias from $M_{1,1}$ to $M_{1,4}$.

After the minimum costs of all sub-trees are computed by Equations (6) and (7), the minimum sub-tree cost of the root is the final cost ($\min_{0 \le l \le L} msc(P_6, l)$ in our example) and we can then trace back the solution to find the path achieving the minimum cost.

D. Multi-level 3D Maze Routing

After initial routing, those nets with violations will be ripped up and may go through multiple iterations of rip-up and reroute (RRR) by maze routing. However, maze routing on the whole 3D grid graph G will be very time consuming due to the large search space. One alternative is to restrict the searching area to the bounding box of the net. Although this approach makes rerouting much faster, the routing quality will be compromised. Considering the above pros and cons, we propose a multi-level 3D maze routing to enable whole-graph searching, while achieving a good trade-off between runtime and routing quality. The proposed routing technique has two levels, maze route planning and fine-grained maze routing within guides, each of which serves a different purpose. Maze route planning aims at finding a smaller but highly routable search space, while fine-grained maze routing seeks to find a path with minimum actual cost within the search space.

First, we introduce an idea called grid graph coarsening, which means to compress a block of G-cells (5x5 in our implementation) into a coarsened cell, like what is shown in Figure 5. The resource $R(A)$ of a coarsened cell A is computed as the average resource of the G-cells in A. We then use the following formula to

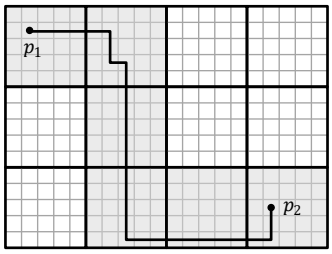


Fig. 5: Multi-level 3D Maze Routing.

model the edge cost between two adjacent coarsened cells.

$$C_W(A,B) = s \times \left(\frac{1}{\max(R(A),0.1)} + \frac{1}{\max(R(B),0.1)}\right), \quad (8a)$$

$$C_V(A,B) = \frac{1}{\max(R(A),0.1)} + \frac{1}{\max(R(B),0.1)}. \quad (8b)$$

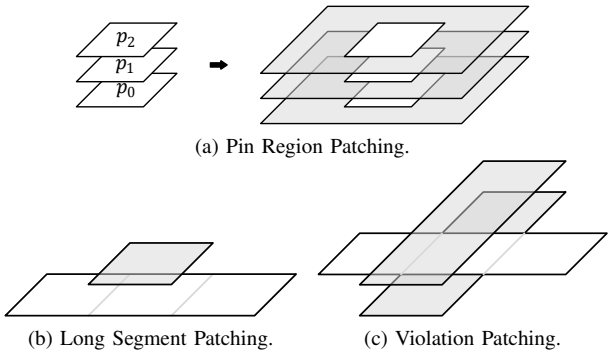
In Equation (8), $C_W(A,B)$ represents the edge cost between coarsened cells A and B when they are on the same layer, while $C_V(A,B)$ represents the edge cost when two cells are on adjacent layers. The parameter s refers to the coarsening scale, i.e., the number of G-cells in each row (column) of the coarsened cell. In our implementation, s is set to be 5. We multiply the cost by the parameter s because moving from the center of a coarsened cell A to that of its neighboring coarsened cell B on the same layer, the wire will pass through s fine-grained edges. However, moving from the center of A to that of B on an adjacent layer, only one via is needed. This cost scheme for maze routing planning is highly sensitive to resource changes, and the cost will double each time the resource halves.

In this way, a coarsened grid graph G_c with a much smaller size than the original graph G can be constructed, with a different cost scheme to enable the global router to navigate in G_c. The grey area in Figure 5 shows an example solution of this maze route planning step. The solution is then transformed into bounding boxes and fed into the fine-grained maze router. The fine-grained maze router will then perform another level of maze routing with the cost scheme discussed in Section III-B within these bounding boxes, aiming at finding a minimum cost path. An example solution of the fine-grained maze routing step is shown as the dark black path in Figure 5.

E. Postprocessing / Guide Patching

Since the output of the global router is connected rectangular route guides, not restricted to paths of single G-cell width, we can add new guides to improve detailed routability, so long as the guides still forms a connected path. For this reason, we develop a technique called patching - adding new stand-alone guides to alleviate routing hot spots. We propose 3 kinds of patching to address hot spots that occur for different reasons, namely pin region patching, long segment patching and violation patching.

1) Pin Region Patching: Pin region patching is the most effective patching among the three. It is needed due to the fact that pin accessibility is crucial for a net to be successfully routed, while it also involves too many details for global routing to handle. Therefore, the ideal way of improving pin accessibility is to identify those hard-to-access pins and assign more resources to them so that the detailed router can have the flexibility to find a way to access the pins. Our global router will check the upper (or lower) two layers of a pin, which are vital for accessing the pin. If the resource on either of them is below a specified threshold T, a larger guides will be patched. Take Figure 6a as an example, G-cell p_0 contains a pin, so the resources of p_1 and p_2 will be checked. If either $r(p_1) < T$ or $r(p_2) < T$, three 3×3 patching guides will be added to all the three layers.



(a) Pin Region Patching.

(b) Long Segment Patching. (c) Violation Patching.

Fig. 6: Different Kinds of Patching.

2) Long Segment Patching: A completely vacant long track is often hard to find, so a longer routing segment often means more wrong way wires and causing more congestion. We will thus identify the bottlenecks for long segments and add patches above or below to help with track switching, so as to reduce wrong ways. If a guide is longer than a specified length I, we'll consider long segment patching. Starting from one end of the segment and walking towards the other, if a G-cell with resource below a threshold T is encountered, a single G-cell route guide will be patched above or below it, depending on which of them has sufficient resource as shown in Figure 6b. Besides, after a G-cell is patched we will skip next I G-cells before we patch another one.

3) Violation Patching: For G-cell with inevitable violations, patching will be used again to enable the detailed router to search with more flexibility. Figure 6c shows an example of violation patching, in which the G-cells on the two sides of the G-cell with violation, along with the three G-cells above are patched.

IV. Experimental Results

We used C++ with the boost geometry library [16] to implement our global router, and we used Rsyn [17] to parse LEF/DEF formats. All our experiments are conducted on a 64-bit Linux workstation with Intel Xeon 3.4 GHz CPU and 32 GB memory. The benchmarks are all from ICCAD'19 global routing contest [1]. We global route all the benchmarks using 8 threads in accordance with the contest, and Dr. CU 2.0 [4] is used to generate the detailed routing solution. Lastly, the final results are reported by Cadence Innovus 18.1 [18]. Table II first shows the detailed decomposition of the overall detailed routing scores. Our global router produces solutions with similar wire length and via score with the first place router, but with $1-2\%$ fewer shorts and spacing violations. We then compare both our scores and runtime with the first and second places in the ICCAD'19 contest, which are shown in Table III. Note that the average runtimes and scores are scaled by those of ours. Our router is around 9% faster than the first place and 15 times faster than the second place. The guides generated by our global router takes 5753 seconds on average for the detailed router to route, which is similar to that of the first place ($< 1\%$ difference) and 9% slower than the second place. However, our router produces detailed routing result comparable to the first place and around 11%

TABLE II: Scores Decompositions Compared with the First and Second Places in ICCAD'19 Global Routing Contest.

Design	Wire Length & Via			Non-Preferred Usage			Short			Min-Area and Spacing		
	1st	2nd	ours	1st	2nd	ours	1st	2nd	ours	1st	2nd	ours
18test5	15614801	15728314	15613663	155898	457247	166994	318500	505340	330425	271000	381500	288500
18test5m*	15810136	15426511	15807997	126947	213345	135293	273220	270765	261150	232000	222500	224000
18test8	37434586	37385956	37441058	261785	592927	269993	212445	333670	209470	143500	255500	144000
18test8m	36743453	35912143	36746610	339927	575688	336768	210580	408540	194510	132000	184500	129500
18test10	39037528	39291497	39061258	891219	1488538	882371	671755	1705070	669965	483000	948000	471000
18test10m	40233544	40209785	40246090	1433956	2957377	1413120	4633110	71940390	4021620	705500	4058000	685500
19test7	77294941	78534037	77286072	1428490	1916508	1428396	9800300	13820375	9680620	6987000	9213000	6883000
19test7m	70845164	70994659	70848996	1528759	2643684	1535876	9735870	18154405	9943260	6634000	8349500	6686000
19test8	119200121	120443016	119199593	1339678	1789897	1338449	7826500	9472060	7780220	6053000	6525000	6103000
19test8m	116107932	117059329	116062781	1466713	2194550	1493314	8815065	11847405	8561400	6158500	6496500	6089000
19test9	184218174	186350610	184246497	2185321	2955375	2181774	14783160	18772785	14765850	10865500	11769000	10847000
19test9m	179227079	180119959	179242111	2334544	3350841	2323850	16289715	21250500	16020280	11124500	12105500	10948000
Avg.	1.00	1.01	1.00	1.00	1.56	1.00	1.02	2.33	1.00	1.01	1.22	1.00

* Note that 18test5m is the abbreviation for ispd18_test5_metal5. The same rule applies to other designs.

TABLE III: Runtime and Scores Compared with the First and Second Places in ICCAD'19 Global Routing Contest.

Design	GR Runtime (s)			DR Score		
	1st	2nd	ours	1st	2nd	ours
18test5	83	56	68	16111082	16690901	16089196
18test5m	80	403	85	16204442	15910624	16210303
18test8	260	229	236	37920522	38312552	37908815
18test8m	240	621	300	37277889	36896370	37293962
18test10	349	510	334	40613594	42485106	40600501
18test10m	350	21119	373	45680831	115107553	46300610
19test7	564	1297	506	88453086	94332917	88577731
19test7m	335	1780	377	82380132	91849749	82169293
19test8	562	966	365	128356260	131762471	128412302
19test8m	499	32257	588	126172995	131164782	126429212
19test9	896	1207	528	201262621	208180772	201270655
19test9m	593	6755	658	197686238	204815803	197937335
Avg.	1.09	15.21	1.00	1.00	1.11	1.00

better than the second place. Besides, our algorithm's peak memory is close to the first place and 1.83 times of that of the second place on average (ours is 8.22 GB on average and 19.8 GB for the biggest design).

V. Conclusion

In this work, we propose a 3D global router that can well balance routing quality and efficiency. A sophisticated cost scheme for wires and vias is designed to optimize wire length, via number and congestion. A technique of 3D pattern routing combines 2D pattern routing and layer assignment, and is able to route most of the nets fast and optimal. A multi-level maze routing strategy first narrows the search space to a smaller region with sufficient resource, and then performs bounded maze routing to search for a minimum cost path. Besides, we also demonstrate that the patching technique helps to improve detailed routability. Compared with the top participants of ICCAD'19 global routing contest, our global router exhibited competitive performance and runtime.

Acknowledgment

The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK2150992).

References

[1] S. Dolgov, A. Volkov, L. Wang, and B. Xu, "2019 cad contest: Lef/def based global routing," 2019.

[2] G. Chen, C.-W. Pui, H. Li, J. Chen, B. Jiang, and E. F. Young, "Detailed routing by sparse grid graph and minimum-area-captured path search," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 754–760, ACM, 2019.

[3] G. Chen, C.-W. Pui, H. Li, and E. F. Young, "Dr. cu: Detailed routing by sparse grid graph and minimum-area-captured path search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2019.

[4] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Young, "Dr. cu 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2019.

[5] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "Nctu-gr 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 32, no. 5, pp. 709–722, 2013.

[6] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang, "Nthu-route 2.0: a fast and stable global router," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 338–343, IEEE Press, 2008.

[7] H.-Y. Chen, C.-H. Hsu, and Y.-W. Chang, "High-performance global routing with fast overflow reduction," in *2009 Asia and South Pacific Design Automation Conference*, pp. 582–587, IEEE, 2009.

[8] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: global router with efficient via minimization," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pp. 576–581, IEEE Press, 2009.

[9] M. Pan and C. Chu, "Fastroute 2.0: A high-quality and efficient global router," in *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pp. 250–255, IEEE Computer Society, 2007.

[10] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1066–1077, 2008.

[11] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "Grip: scalable 3d global routing using integer programming," in *Proceedings of the 46th Annual Design Automation Conference*, pp. 320–325, ACM, 2009.

[12] Y. Xu and C. Chu, "Mgr: Multi-level global router," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 250–255, IEEE Press, 2011.

[13] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2007.

[14] M. Pan and C. Chu, "Fastroute: A step to integrate global routing into placement," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pp. 464–471, ACM, 2006.

[15] T.-H. Lee and T.-C. Wang, "Congestion-constrained layer assignment for via minimization in global routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 9, pp. 1643–1656, 2008.

[16] "Boost geometry library." https://www.boost.org/doc/.

[17] G. Flach, M. Fogaça, J. Monteiro, M. Johann, and R. Reis, "Rsyn: An extensible physical synthesis framework," in *Proceedings of the 2017 ACM on International Symposium on Physical Design*, pp. 33–40, ACM, 2017.

[18] "Cadence innovus implementation system." https://www.cadence.com.