Any-Angle Routing for Redistribution Layers in 2.5D IC Packages

Min-Hsuan Chung[1], Je-Wei Chuang[1], and Yao-Wen Chang[1,2]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106319, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 106319, Taiwan
mhchung@eda.ee.ntu.edu.tw; jwchuang@eda.ee.ntu.edu.tw; ywchang@ntu.edu.tw

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10247899

***Abstract*—Redistribution layers (RDLs) are widely applied for signal transmissions in advanced packages. Traditional redistribution layer (RDL) routers use only 90- and 135-degree turns for routing. With technological advances, routing in RDLs can be any obtuse angle, leading to larger routing solution spaces and shorter total wirelength. This paper proposes the first any-angle routing algorithm in the literature for multiple RDLs. We first give a novel global routing algorithm with accurate routing resource estimation. A multi-net access point adjustment method is then proposed based on dynamic programming and our partial net separation scheme. Finally, we develop an efficient tile routing algorithm to obtain valid routes with fixed access points. Experimental results show that our algorithm can achieve a 15.7% shorter wirelength compared with a traditional RDL router.**

I. Introduction

As the advanced process cost increases dramatically, heterogeneous integration by integrating multiple chips with mature technologies into a single package becomes promising to achieve the cost-performance optimization goal. Among existing packaging technologies, the integrated fan-out (InFO) structure adopted by TSMC shows significant advantages, including high-density interconnections, better power efficiency, and better thermal properties [1], [2], [3]. In an InFO package, interconnections between different chips are realized in the redistribution layers (RDLs).

Figure 1 shows the side view of an InFO package. The top part is the molding compound for the physical protection of the chips. I/O and bump pads are attached for the I/O signals between the chips and PCB. Interconnections between chips are implemented in RDLs. The RDLs are staggered via and wire layers, which are used to transmit the signals vertically and horizontally. Unlike traditional fixed-angle routing, TSMC can now support any obtuse angle connections in an RDL with its advanced manufacturing technology for high-end designs. As a result, developing any-angle routers at the package level attracts much attention recently [4], [5], [6], [7] because of the increasing needs in advanced heterogeneous integration manufacturing, due to its advantages with higher routability and shorter wirelength.

A. Previous Works

To the best of our knowledge, no existing any-angle routing algorithms are proposed for the multiple RDLs packages. Nevertheless, this problem can be viewed as a combination of traditional RDL routing problems and any-angle path-finding problems. Traditional RDL routing algorithms performs routing with multiple nets. However, underestimating routing resource and over constraining solution space could reduce the solution quality of any-angle routing. On the other hand, the any-angle path-finding algorithms could provide desired solutions for a single net but might not perform well for multiple nets.

1) Traditional RDL Routing: For RDL routing, previous work tended to consider the routing process under the X-architecture, in which wire connections in an RDL have only four feasible orientations: (1) vertical, (2) horizontal, and (3) 135-degree [8], [9], [10], [11], [12], [13], [14]. For single-layer structures, Fang *et al.* [8] first dealt with the problem with *Max-Flow Min-Cut* (MCMF) algorithm by modeling channels as vertices on the routing graph. Fang *et al.* [9] used variables to model the positions of a wire crossing each channel. They then formulated integer linear programming (ILP) with a variable reduction technique to obtain the solution. Fang *et al.* [10] extended the network-flow-based algorithm to provide solutions for more general cases. After performing *Delauney Triangulation* (DT) and obtaining *Voronoi Diagram* (VD) on the given routing region, they applied the MCMF algorithm on the VD to estimate congestion for routing. Lin *et al.* [11] solved the problem by performing *Maximum Planer Subset of Cords* (MPSC) and weighted *Longest Common Subsequence* (LCS) algorithms to find the desired routing topologies, which makes it more efficient than previous works.

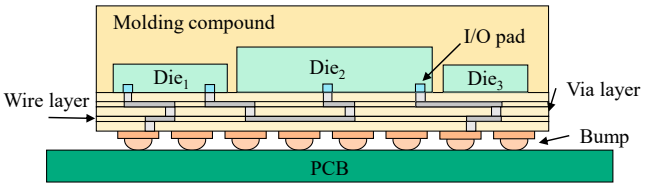

Figure 1. Side view of an InFO package.

As technology improved, multi-layer structures were proposed. Finding a better topology to minimize the layer used becomes important and challenging. Taking the advantages of the regular pad structure, Lin and Lin [12] proposed a concentric circle model and solved the layer assignment problem with LCS.

Modern technology allows switching between different layers for a single net through via, which makes the solution space even larger. Wen *et al.* [13] used MPSC to perform layer assignment and concurrent routing. They then used A*-search with an octagonal tile model to route the remaining nets. Cai *et al.* [14] performed global routing by crossing-aware A*-search. First, they partitioned a routing region into convex tiles, and then modeled those tiles as vertices on the routing graph. After preprocessing, A*-search was performed on the routing graph, and the net sequence on the edges of the routing graph was used to maintain crossing information.

These previous works provide some insights into routing guide generation. However, their formulation is not for the any-angle problem.

2) Any-Angle Routing: Yang *et al.* [15] proposed an any-angle routing algorithm for single-layer biochips. This work partitions a routing region into tiles through constrained *Delauney Triangulation* (DT). Next, a routing graph is constructed by modeling tile edges as vertices. The nets are then routed sequentially with *Dijkstra's algorithm* and optimized by a dynamic programming method. After routing a net, the routing graph is reconstructed with routed nets treated as constraints. This algorithm considers nets sequentially, incurring a waste of routing resources because of the spacing issue, and the triangulation process after finishing each net is time-consuming. Kohira *et al.* [16], [17] formulated the any-angle routing problem as a quasi-Newton optimization problem by transforming crossing checking and the maximum/minimum functions into differentiable ones. However, this analytical formulation cannot scale well for package routing because of its high time complexity.

B. Motivation

Any-angle routing provides a much larger solution space for routing considerations. With any-angle routing, solutions of shorter wirelength and higher routability can be obtained because of its larger solution space. Figure 2 shows a routing instance that is routable for any-angle routing but not feasible for traditional RDL routing. In the figure, the top wire layer is shown, where the vias pass signals to the I/O pads. Let m_i^j denote the j^{th} pin of net i, and v_i denote the i^{th} via. As shown in Figure 2(a), the channel between v_i and v_j cannot be fully utilized because of the fixed orientation constraint. As shown in the blue dotted line, the effective channel length projected to the specific orientation would be shorter than the original channel length. On the other hand, the routes in any-angle routing can be perpendicular to the channel. As shown in Figure 2(b), the perpendicular routes lead to shorter lengths on the channel and give a feasible routing solution. This example reveals the potential of any-angle

This work was partially supported by AnaGlobe, ASUS, Delta Electronics, Google, Maxeda Technology, TSMC, MOST/NSTC of Taiwan under Grant NSTC 110-2224-E-002-012, NSTC 111-2218-E-002-021, MOST 110-2221-E-002-177-MY3, and MOST 111-2622-8-002-023-SB.

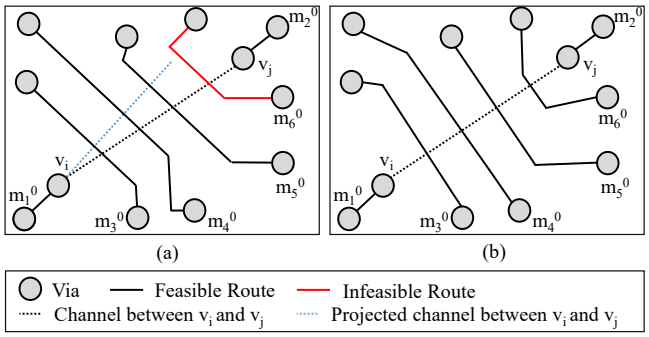

Figure 2. (a) Routing result of an RDL router in a single layer. (b) Routing result of any-angle routing in a single layer.

routing. However, as discussed, current any-angle routers only work for small problem sizes because of their time-consuming and routing-resource-wasting operations. Thus, it is desirable to develop an efficient any-angle routing algorithm for modern advanced designs.

C. Our Contributions

We summarize our main contributions as follows:

- To the best of our knowledge, this paper is the first work in the literature to solve the any-angle RDL routing problem with multiple RDLs.
- We propose a capacity estimation method and a refinement process for routing guide generation after DT, which gives a higher utilization rate in the congestion region.
- We extend the dynamic programming-based access point adjustment in the previous work, making it work for multiple nets simultaneously.
- We develop a tile routing algorithm to perform legalization and optimization in a tile.
- Experimental results show that our algorithm can achieve higher routability than the state-of-the-art any-angle router [15] and an average of 15.7% shorter wirelength than the state-of-the-art RDL router [14].

The remainder of this paper is organized as follows. Section II introduces the constraints and problem formulation for the any-angle RDL routing problem. Section III presents our global and detailed routing flow. Section IV reports the experimental results. Finally, Section V concludes this paper.

II. Preliminaries

In this section, we first give the terminologies used in this paper and then formulate our any-angle RDL routing problem.

A. Terminologies and Notations

We will use the following terminologies and notations:

- L_v: the via layer, which transmits signals between two adjacent wire layers.
- L_w: the wire layer, which contains metal wires.
- $B = \{B_i \mid 1 \leq i \leq |B|\}$: the set of all bump pads.
- $Q = \{q_i \mid 1 \leq i \leq |Q|\}$: the set of all I/O pads.
- $V = \{v_i \mid 1 \leq i \leq |V|\}$: the set of all vias.
- $M = \{m_i \mid 1 \leq i \leq |M|\}$: the set of all nets.
- $T = \{t_i \mid 1 \leq i \leq |T|\}$: the set of tiles constructed by the DT.
- $\gamma = \{\gamma_i \mid 1 \leq i \leq |\Gamma|\}$: the set of all access points. Access points are the locations where nets intersect with the boundaries of tiles.
- $P = \{p_i(x_i, y_i) \mid 1 \leq i \leq |P|\}$: the set of all points located at (x_i, y_i) in the 2D plane.
- $s(p_i, p_j)$: the segment that ends at p_i and p_j.
- $d(v_i, v_j)$: the distance between v_i and v_j.
- $C(p_i, rad)$: the circle with the center located at p_i and radius rad.
- m_i^j: the j^{th} i/o pin of m_i.
- $\kappa(i, j, k)$: a tile with three vertices related to v_i, v_j, and v_k.
- $r(\gamma_i, \gamma_j)$: the detailed route connecting γ_i and γ_j, which is a list of segments connecting the two access points.
- w_w: the wire width.
- w_v: the via width.
- w_s: the minimum spacing.
- w_x: the minimum turn-to-turn distance.

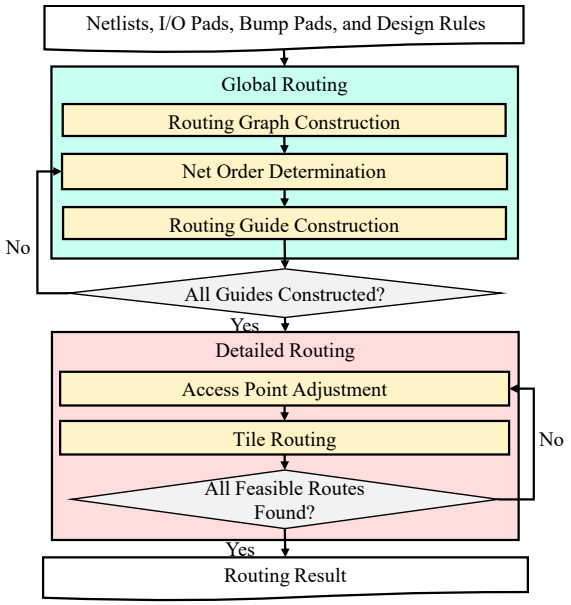

Figure 3. Our algorithm flow.

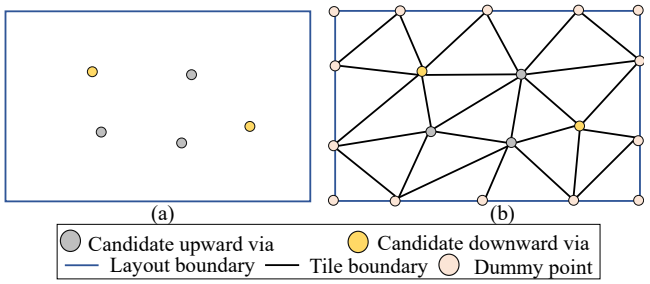

Figure 4. (a) Top view of via planning in a wire layer. (b) Tile partitioning result of (a).

B. Any-Angle RDL Routing Constraints

We consider the any-angle RDL routing problem with flexible vias and the following design rules:

- Flexible via structure: The vias in each via layer can be placed anywhere.
- Minimum spacing rule: A minimum spacing is needed between any two vias or wire segments belonging to different nets.
- Minimum angle constraint: Two connected segments can turn at any angle greater than or equal to 90-degree.
- Minimum turn-to-turn distance: The distance between two successive turns should be larger than a fixed number for better manufacturability.

C. Problem Formulation

Our any-angle RDL routing problem is formally defined as follows:

*Problem 1 (**Any-Angle RDL Routing**):* Given design rules, a set of I/O pads and bump pads, and a pre-assignment netlist, connect all nets to maximize the routability and minimize the total wirelength with no design rule violation.

III. Proposed Algorithms

In this section, we first give an overview of the whole algorithm flow and then detail our methods. Figure 3 shows our algorithm flow, which consists of two major stages: (1) *Global Routing* and (2) *Detailed Routing*. In Global Routing, we first construct a multi-layer routing graph based on the via planning [14] and the DT, which is a commonly used partitioning method in the RDL routing [10], [15]. We estimate the routing resources

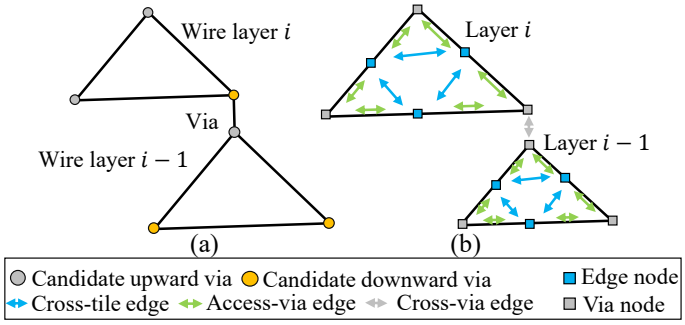

Figure 5. (a) The triangular tiles in two adjacent wire layers. (b) The corresponding routing graph representation of (a).

of the triangular tiles to determine the initial net order based on the routing graph. When some routing guides fail to be generated, the net order is adjusted until all guides are generated successfully. In Detailed Routing, we modify the dynamic programming-based single-net access point adjustment [15] to determine the initial access point location. After the access points are located, we route the nets tile by tile. These two stages will be detailed in the following.

A. Global Routing

In this stage, we first construct a routing graph, then determine the initial net order by the routing resource from the triangular tiles. Finally, non-crossing guides are obtained by maintaining the net-sequence lists [14] and blocking the infeasible searing nodes.

1) Routing Graph Construction: We first adopt the via planning algorithm in [14] to generate the candidate vias for connections in different wire layers. As shown in Figure 4, we perform DT with all candidate vias connected to the wire layer as the input vertices. In addition, to obtain more balanced triangulation results for estimating resources around the outline [10], a fixed number of dummy points are uniformly inserted into the boundary of the outline.

With the triangular tiles, we construct a routing graph, as shown in Figure 5. Our routing graph is composed of two kinds of search nodes: (1) *via nodes* ($N_v{}^i$) and (2) *edge nodes* ($N_e{}^{i,j}$), and three kinds of edges: (1) *cross-via edges* ($E_v(i,j)$), (2) *access-via edges* ($E_a(i,j,k)$), and (3) *cross-tile edges* ($E_t(i,j,k)$).

A via node is a node with capacity one, which is used to model a candidate via. An edge node is a node used to model the edge segment between two candidate vias (v_i,v_j) in a tile, and its capacity is computed as follows:

$$C_{i,j} = \lfloor d(v_i, v_j)/(w_w + w_s) \rfloor \tag{1}$$

where $C_{i,j}$ is the capacity of $N_e{}^{i,j}$.

A cross-via edge is an edge with capacity one, representing the connection between two via nodes in different layers. An access-via edge is an edge also with capacity one, which connects a via node and an edge node. A cross-tile edge connects two edge nodes, which models the route along one corner of a tile. Spacing violations would occur when too many guides pass through an edge node pair (N_e^i, N_e^j), even when both capacities of N_e^i and N_e^j are not violated. As shown in Figure 6(a), a spacing violation occurs because the edge node capacity of a corner overestimates the guide number passing through the corner. For the balance between w_x and the capacity of a tile corner j, we use a 3-segment routing pattern to estimate the capacity. As in Figure 2(b), to better utilize the resource in a tile edge, the pattern uses two segments perpendicular to the tile edge first. Then we separate the corner j into two corners, j_1 and j_2, by the bisector of j. As shown in Figure 6(b), the shorter bisector length between j_1 and j_2 is defined as the effective length l_j of corner j. Then, we define the capacity of a cross-tile edge e with corner j as follows:

$$C^e = \lfloor cos(\frac{ang(j)}{4}) \times l(j)/(w_w + w_s) \rfloor \tag{2}$$

where $ang(j)$ is the angle of corner j.

Because the capacities of a tile edge and a tile corner are treated separately, the graph model estimates the capacity with only little resource overhead. By implementing edge nodes and cross-tile edges, the tile

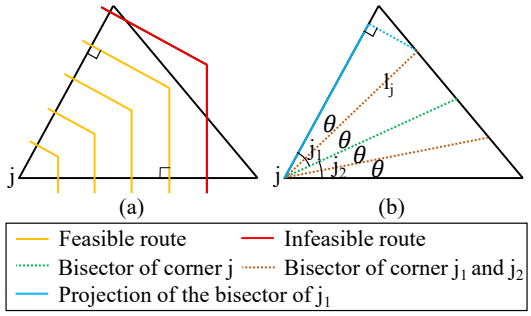

Figure 6. (a) Capacity violation in corner j. (b) Effective length l_j of corner j.

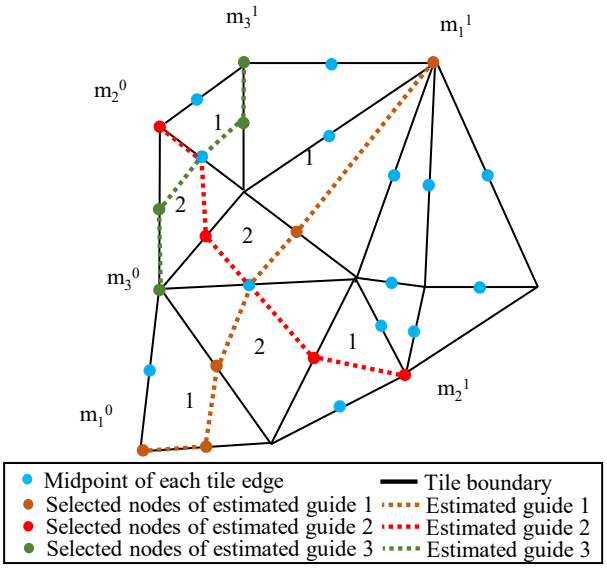

Figure 7. Initial net ordering determination.

with balanced usage of the three cross-tile edges can be fully utilized approximately, which is desirable for the congested region.

2) Net Order Determination: To avoid the routing guide congestion, we determine the initial net order by the following estimation method. First, we generate the routing guides for each net without considering others. When a guide is generated, a *Rectangular Uniform Wire Density (RUDY)*-like wire density estimation method [18] is applied, and the estimated wire cost is distributed to the tiles nearby. After all guides are generated, the congestion cost of each tile is computed by the sum of wire density caused by all nets. As shown in Figure 7, a simple example of the congestion cost in each tile is given. We then trace all guides and record the number of tiles with congestion costs higher than a user-defined value.

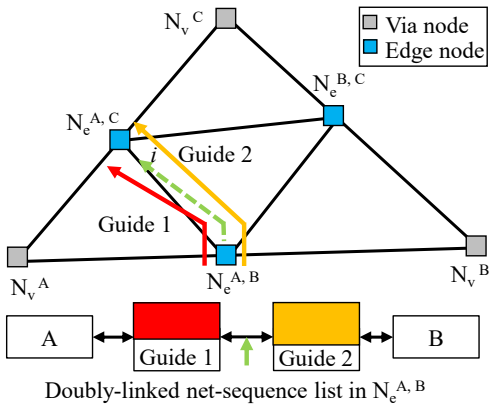

Figure 8. Infeasible searching node blockage.

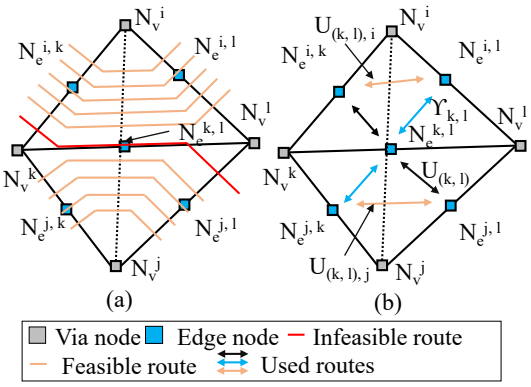

Figure 9. (a) Overflow of the diagonal capacity. (b) Utility to be checked for the refinement.

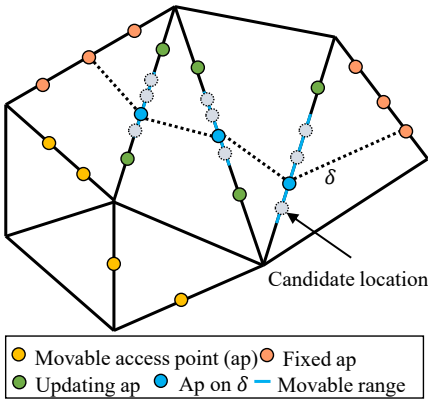

Figure 10. Update after the adjustment of δ.

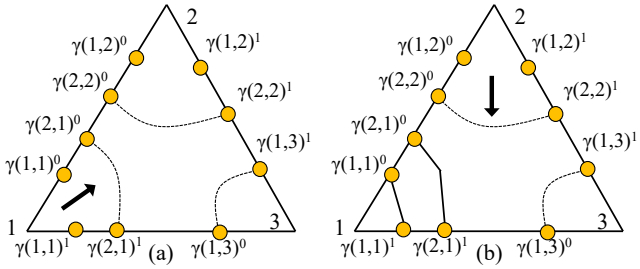

Figure 11. (a) Routing order in the beginning. (b) Routing order after $\Gamma(1)$ is routed.

When the user-defined threshold is given, the congestion cost of each guide can be obtained by the number of passing tiles with a congestion cost higher than the threshold. Take $guide1$ as an example; when we start from m_1^0 to m_1^1, the congestion cost of tiles $guide1$ passed are 1, 2, 2, and 1 separately. Hence, when the user-defined threshold is 1.5, the number of congested tiles of $guide1$ is 2. Then, the overall congested cost of each guide is computed by the number of congested tiles and the number of total tiles it passes. After all guides are traversed, we adjust the net order by the number of congested tiles and the *Euclidean distance* of the two pins. The net with more congested tiles and shorter Euclidean distance would be routed first.

3) Routing Guide Construction: We generate the routing guides by crossing-aware A*-search with capacity checking. After all guides are generated, we apply diagonal utility refinement to ensure the validity of the generated guides. If some guides fail to be generated, we adjust the net order and regenerate the guides with the adjusted order. The three steps are detailed as follows:

a) **Crossing-aware A*-search:** We maintain a net-sequence list in each edge node to avoid the topological crossing between any two guides. Since crossing between two guides always leads to a net sequence with the wrong order on the boundary of some tile, maintaining the guides with the correct order of a net sequence on the boundary of all tiles gives a non-crossing guide topology. We record the left and right guides next to the processing guide in each searched $N_e^{i,j}$ and block the invalid nodes by the two nearby guides. As shown in Figure 8, when searching for the next candidate node of guide i, the valid searching node remaining is only $N_e^{A,C}$ because both $guide1$ and $guide2$ connect to $N_e^{A,C}$. By tracing the destination of the adjacent guides, the searching nodes with crossing are blocked.

b) **Diagonal Utility Refinement:** Since the number of guides between N_v^i and N_v^j is bounded by $d(v_i, v_j)$, the diagonal utility refinement is performed to remove infeasible guides. As shown in Figure 9(a), the number of routes passing N_v^i and N_v^j is bounded by $d(v_i, v_j)$. The red route is infeasible even when the capacities of $E_t(i,k,l)$ and $E_t(j,k,l)$ are not violated. To eliminate the violation, we examine the usage of the cross-tile edges around the edge nodes. As shown in Figure 9(b), the diagonal utility constraint $D_{k,l}$ of $N_e^{k,l}$ is defined as follows:

$$(U_{(k,l),i} + U_{(k,l),j} + \Upsilon_{k,l} + 1) \times (w_w + w_s) < d(v_i, v_j) \qquad (3)$$

where both $\kappa(k,l,i) \in T$ and $\kappa(k,l,j) \in T$, $U_{(k,l),i}$ is the number of guides between N_v^i and $N_e^{k,l}$, and $\Upsilon_{k,l}$ is the number of guides passing through $N_e^{k,l}$. When the diagonal capacity constraint $D_{k,l}$ is violated, we reduce the capacity of $N_e^{k,l}$ and reroute the nets passing through $N_e^{k,l}$ until there is no diagonal utility violation.

c) **Net Order Adjustment:** When some guides fail to be generated, we rip up all routed guides and update the failure count of each net. Based on the failure count in each net, we update the net order by moving the nets with larger failure counts to the front. After the net order adjustment, we construct the routing guides with the updated order until all guides are generated.

Since the guides in our routing algorithm should be non-crossing, our adjustment procedure could be more effective than the negotiation-based rip-up and reroute (*NRR*) mechanism in [19] in some cases. Consider a pin surrounded by routes from other nets. If no overflow occurs in the surrounding tiles, the NRR gives the same result in every iteration. In contrast, our adjustment procedure would change the order according to the failure count of the nets, providing the desired routes under the circumstance.

B. Detailed Routing

After the feasible guides are generated, we evenly distribute the access points onto the belonging tile edges. Starting from the initial location, we move the access points to further reduce the total wirelength. Finally, the tile routing is performed to finish the route after the locations of all access points are determined.

1) Access Point Adjustment: To allocate the routing resources of the tile edges, we compute the movable range of each access point. The access points are further classified as (1) *fixed* and (2) *movable* by the length of the movable range. To speed up the adjustment process, we separate the original nets into partial nets. We define the partial net (δ_i^j) as the j^{th} longest access point list of n_i with consecutive movable access points, and the length of a partial net δ is the number of access points in it, denoted by $leng(\delta)$. We adjust the locations of the access points by the dynamic programming-based method [15]. Since the length of the partial net with a larger number of access points is more likely to be reduced, we adjust the longest partial net (δ^*) first. In each adjustment procedure, a user-defined number of candidate positions are evenly distributed into the movable range of all access points in δ^*. The dynamic programming algorithm gives the shortest solution of the partial net δ^*. After the locations of access points in a δ^* are determined, the movable range of every access point adjacent to δ^* could be updated by a single traversal through δ^*. With a maximum heap recording all partial nets, we can incrementally update the partial nets in the adjustment procedure. As shown in Figure 10, when partial net δ is processed, only five adjacent access points need to be updated. The access point adjustment iterates until the movable range of all access points are small enough, or all access points are moved.

The adjustment of δ takes $O(leng(\delta))$ time in the dynamic programming stage. After adjusting δ, at most $leng(\delta)$ heap keys of partial nets need to be modified. Since changing a key in the partial net heap requires less than $O(|\Gamma|)$ time, where Γ is the total number of access points, the key update after each adjustment is in $O(leng(\delta)(\lg(|\Gamma|))$ time. Since

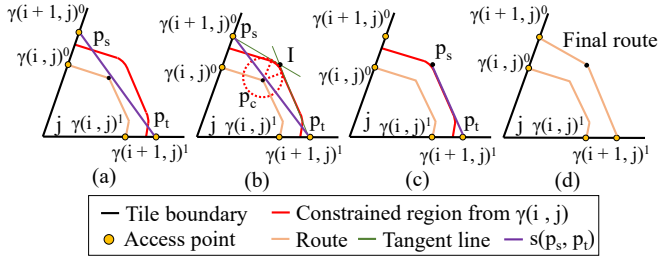

Figure 12. (a) Violation between $s(p_s, p_t)$ and $r(\gamma(i,j)^0, \gamma(i,j)^1)$ is found. (b) Intersected point p_i is found. (c) Replace p_s with p_i. (d) No spacing violation exists.

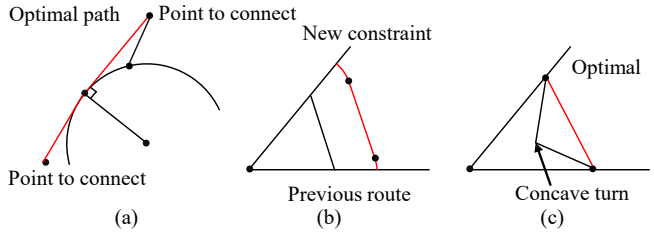

Figure 13. (a) Tangent for a smooth boundary. (b) Constraint from the previous route. (c) Convex optimal curve.

each access point is moved only once, the total length of the modified partial nets is $O(|\Gamma|)$. Thus, the overall complexity of the access point adjustment is $O(|\Gamma| \lg(|\Gamma|))$. We thus have the following theorem.

Theorem 1: The time complexity of the access point adjustment is $O(|\Gamma| \lg(|\Gamma|))$, where Γ is the total number of access points.

2) Tile Routing: After the locations of all access points are determined, we finish the remaining detailed routing tile by tile. In this stage, the access points to be connected are all paired and located on the processing tile boundaries. For every pair of access points on the same tile, the connected corner via v_j is defined as the corner node (N_j^c) of the access point pair. We define $\gamma(i,j) = (\gamma(i,j)^0, \gamma(i,j)^1)$ as the i^{th} access point pair from N_j^c, where an access point pair is a pair of two access points belonging to the same net, and $\Gamma(j)$ is the set of all $\gamma(i,j)$. We introduce the two-stage tile routing process as follows:

a) **Routing Order Determination:** We determine the net order in the clockwise order of the tile corners. For routing nets in corner j, we route $\gamma(i,j)$ after $\gamma(k,j)$ if $k < i$. As shown in Figure 11(a), the $\gamma(1,1)$ is routed before $\gamma(2,1)$. After all access point pairs in $\Gamma(j)$ are routed, we route the nets in another corner until all routes are finished. As shown in Figure 11(b), we route the nets in corner 2 after all routes in corner 1 are finished.
In this routing order, when some spacing violations exists between $r(\gamma(i,j)^0, \gamma(i,j)^1)$ and $r(\gamma(i+k,j)^0, \gamma(i+k,j)^1)$, some spacing violations must exist between $r(\gamma(i+k-1,j)^0, \gamma(i+k-1,j)^1)$ and $r(\gamma(i+k,j)^0, \gamma(i+k,j)^1)$. Thus, we record the outermost route of each corner, and then the subsequent spacing constraint checking would be more efficient than the unordered cases.

b) **Fit Routing:** In this stage, we first give a process to obtain a legal solution. Then we show that the result is an good approximation of the optimal solution with no minimum turn-to-turn constraint *S*. To route the access point pair $\gamma(i+1,j)$, we initialize two points, *source* (p_s), and *target* (p_t), by the locations of $\gamma(i+1,j)^0$ and $\gamma(i+1,j)^1$, respectively. Once $s(p_s,p_t)$ violates any spacing constraint with $r(\gamma(i,j)^0, \gamma(i,j)^1)$, as shown in Figure 12(a), the constrained circle, which is the circle centered at a point in the previous route that causes the violation, $O(p_c, w_w + w_s)$ can be obtained by a traversal in $r(\gamma(i,j)^0, \gamma(i,j)^1)$ To solve the violation, we find the intersected point I of the following two tangent lines of $O(p_c, w_w + w_s)$ that pass through: (1) p_s and (2) p_t, as shown in Figure 12(b). Then, p_s is replaced by I, and the process iterates until $s(p_s,p_t)$ incurs no spacing violation with $r(\gamma(i,j)^0, \gamma(i,j)^1)$. As shown in Figures 12(c) and (d), the routing result is obtained when there is no spacing violation. We need some lemmas to show the relationship between the solution obtained and *S*.

Lemma 1: Given a smooth (with derivatives continuous everywhere) constraint boundary and a pair of points to be connected, connections not on the boundary should be segments and connections on the boundary should be its tangent lines for an optimal solution.

Proof 1: As shown in Figure 13(a), if the connection not on the boundary is not straight, we can always use a segment to make the wirelength shorter. The case that the connection is on the boundary works similarly.

Lemma 2: If routed from the corner to the middle of the tile sequentially, the constraints generated in each stage can only consist of segments and arcs.

Proof 2: First, from Lemma 1, as shown in Figure 13(b), if the constraint is smooth and contains only segments and arcs, the route constructed will be smooth and contain only segments and arcs. Next, the only constraint for the innermost net is generated from the corner via, which is an arc. Thus, the resulting corner pattern will contain only segments and arcs. Finally, since the constraints generated by segments are segments and the constraints generated by arcs are arcs, the constraint generated will also be smooth and contain only segments and arcs. From the discussion above and mathematical induction, the lemma holds.

Theorem 2: Constructing nets from the corner to the middle of the tile greedily gives *S* for the corner.

Proof 2: To prove the theorem, we only need to show that our solution for each net is the same as the solution when considering only constraints generated by access points and vias. First, it is clear that all constraint arcs are generated by access points or vias. Thus the constraint segments generated by existing routes can only be segments with both ends being part of arcs. Next, consider the optimal solution of a net only dealing with constraints generated by access points and vias. The region formed by the solution curve and tile edges must be convex. If not, we can always replace the concave turn with a segment, as shown in Figure 13 (c). For a segment with both ends in a convex region, the whole line must be in the region by definition. From the discussion above, the constraint generated by existing routes will not affect the optimal solution since both of its ends are in the region formed by the solution curve and tile edges. Thus the theorem holds.

Back to the construction process, we observe that the process tries to approximate *S* by replacing arcs with segments, which leads to the desired solution quality.

Since each spacing violation to a point on the previous route can be solved by the intersected point finding procedure, the increment of segment number between adjacent routes is at most one. Let r_i be the i^{th} route from the corner. From the discussion above, r_i contains at most $(i+1)$ segments. Constructing route r_i only needs a single traversal of route r_{i-1}. Thus, constructing r_i need at most i checks, and the overall time complexity of the tile routing for a single tile is $O(\sum_{i=1}^{k} i) = O(k^2)$, where k is the number of access points in the tile. When some routes fail to be generated, we enlarge the distance that needs to be kept and iterate the detailed routing.

Theorem 3: The time complexity of the tile routing for a single tile is $O(k^2)$, where k is the number of access points in the tile.

IV. Experimental Results

We implemented our any-angle RDL router in the C++ programming language on a 2.9GHz AMD Ryzen 3990X Processor with 126GB memory. In the global routing, we used the open-source CDT [20] package for the triangulation. We performed experiments on the RDL circuit benchmark used in [14]. The benchmark contains five dense designs, and the statistics are given in Table I, where "#*Chips*," $|IO|$, $|B|$, $|N|$, and $|L_w|$ denote the number of chips, I/O pads, bumps, nets, and wire layers, respectively. For fair comparisons, we limited the runtime of all cases to an hour. The unfinished cases were stopped, and the results with the best routability were reported as the routing results.

To evaluate the effectiveness of our algorithm, we compared it with the state-of-the-art any-angle router [15] and a recent traditional RDL router [14]. Since AARF only works for the single-layer structure, we re-implemented the algorithm with some extensions for the multi-layer RDL structure, namely $AARF^*$. The result of the traditional RDL, named Cai,

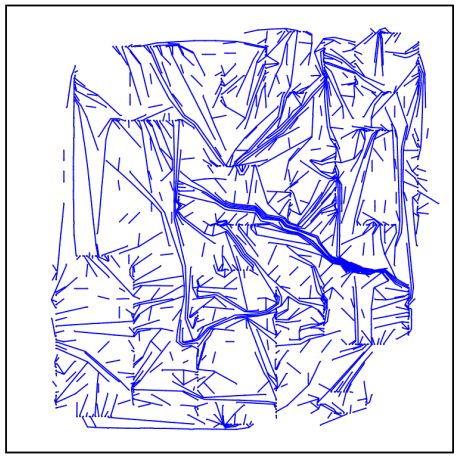

Figure 14. Layout of the first layer in dense5.

is derived from the paper [14] directly. The experimental results are listed in Table II and Table III for comparison with the traditional router [14] and the any-angle router [15], respectively. The routability, wirelength, and runtime are reported. For the cases with routability smaller than 100%, the summation of the wirelength in the successfully routed nets is reported, and the notation '$>$' means just a lower bound.

As shown in Table III, our algorithm achieves 100% routability in all benchmarks, while $AARF^*$ only achieves 100% routability in dense1. For all cases, our algorithm gives the shortest total wirelength compared with other algorithms. On average, our algorithm gives a 15.7% wirelength reduction than Cai. This shows the effectiveness of our work and the potential of any-angle routing.

The reasons why our proposed any-angle routing flow can achieve significantly better routability and wirelength with a reasonable runtime overhead are analyzed as follows:

- AARF greedily uses the total routing resource without reserving for the subsequent routes. The greedy strategy reduces the routability in the congestion region. In contrast, our algorithm results in better routability in the congestion region, as shown in the right part of Figure 14.
- In a sparse region, our any-angle router adjusts the access points and connects the routes close to long segments, unlike the traditional RDL routers that tend to generate fragmented detoured segments.

Table I
BENCHMARK STATISTICS

Circuit	$\#Chips$	$\|IO\|$	$\|B\|$	$\|N\|$	$\|L_w\|$
dense1	2	44	324	22	2
dense2	3	92	784	46	2
dense3	5	158	308	79	3
dense4	6	222	684	111	3
dense5	9	522	1444	261	4

Table II
EXPERIMENTAL RESULTS COMPARED WITH A TRADITIONAL RDL ROUTER.

Case	Routability (%)		Wirelength (μm)		Runtime (s)	
	Cai	Ours	Cai	Ours	Cai	Ours
dense1	100	100	86695	74640	0.79	0.415
dense2	100	100	236269	206451	0.60	0.703
dense3	100	100	258172	222601	3.92	3.68
dense4	100	100	512619	461725	9.13	23.3
dense5	100	100	1835410	1512849	12.5	180.07
Comp.	1	1	1.157	1	0.85	1

V. CONCLUSIONS

To the best of our knowledge, this paper is the first in the literature to consider any-angle RDL routing for multiple RDLs. Our algorithm consists of two stages: (1) Global Routing, which generates the guides for any-angle RDL routing, and (2) Detailed Routing, which adjusts the access point for shorter wirelength and finishes the routing. Experimental results have shown the effectiveness of our any-angle RDL routing algorithm.

Table III
COMPARISON WITH THE RE-IMPLEMENTATION OF THE STATE-OF-THE-ART ANY-ANGLE ROUTER.

Case	Routability (%)		Wirelength (μm)		Runtime (s)	
	$AARF^*$	Ours	$AARF^*$	Ours	$AARF^*$	Ours
dense1	100	100	79525	74640	6.29	0.415
dense2	86.9565	100	$>$ 195880	206451	3600	0.703
dense3	88.6076	100	$>$ 210798	222601	3600	3.68
dense4	83.7838	100	$>$ 408987	461725	3600	23.3
dense5	75.8621	100	$>$ 1081270	1512849	3600	180.07
Comp.	0.87042	1	–	1	1257.77	1

REFERENCES

[1] C. C. Liu, S.-M. Chen, F.-W. Kuo, H.-N. Chen, E.-H. Yeh, C.-C. Hsieh, L.-H. Huang, M.-Y. Chiu, J. Yeh, T.-S. Lin, T.-J. Yeh, S.-Y. Hou, J.-P. Hung, J.-C. Lin, C.-P. Jou, C.-T. Wang, S.-P. Jeng, and D. C. H. Yu, "High-Performance Integrated Fan-Out Wafer Level Packaging (InFO-WLP): Technology and System Integration," in *Proc. of IEEE IEDM*, San Francisco, CA, December 2012, pp. 14.1.1–14.1.4.

[2] H.-P. Pu, H. Kuo, C. Liu, and C. Douglas, "A Novel Submicron Polymer Re-Distribution Layer Technology for Advanced InFO Packaging," in *Proc. of IEEE ECTC*, San Diego, CA, June 2018, pp. 45–51.

[3] C.-F. Tseng, C.-S. Liu, C.-H. We, and D. Yu, "InFO (Wafer Level Integrated Fan-Out) Technology," in *Proc. of IEEE ECTC*, Las Vegas, NV, June 2016, pp. 1–6.

[4] GUC Tapes Out AI/HPC/Networking Platform on TSMC CoWoS® Technology. [Online]. Available: https://www.eetimes.com/guc-tapes-out-ai-hpc-networking-platform-on-tsmc-cowos-technology/

[5] W.-H. Liu, B. Chen, H.-Y. Chang, G. Lin, and Z.-S. Lin, "Challenges for Automating Package Routing," in *Proc. of ISPD*, Canada, March 2022, pp. 193–194.

[6] Synopsys and TSMC Accelerate 2.5D/3DIC Designs with Chip-on-Wafer-on-Substrate and Integrated Fan-Out Certified Design Flows. [Online]. Available: https://news.synopsys.com/2020-08-25-Synopsys-and-TSMC-Accelerate-2-5D-3DIC-Designs-with-Chip-on-Wafer-on-Substrate-and-Integrated-Fan-Out-Certified-Design-Flows

[7] The Chronicle of CoWoS. [Online]. Available: https://3dfabric.tsmc.com/english/dedicatedFoundry/technology/cowos.htm

[8] J.-W. Fang, I.-J. Lin, P.-H. Yuh, Y.-W. Chang, and J.-H. Wang, "A Routing Algorithm for Flip-Chip Design," in *Proc. of ICCAD*, San Jose, CA, November 2005, pp. 752–757.

[9] J.-W. Fang, C.-H. Hsu, and Y.-W. Chang, "An Integer-Linear-Programming-Based Routing Algorithm for Flip-Chip Designs," *IEEE TCAD*, vol. 28, no. 1, pp. 98–110, 2009.

[10] J.-W. Fang, M. D. F. Wong, and Y.-W. Chang, "Flip-Chip Routing with Unified Area-I/O Pad Assignments for Package-Board Co-Design," in *Proc. of DAC*, San Francisco, CA, July 2009, pp. 336–339.

[11] C.-W. Lin, P.-W. Lee, Y.-W. Chang, C.-F. Shen, and W.-C. Tseng, "An Efficient Pre-Assignment Routing Algorithm for Flip-Chip Designs," *IEEE TCAD*, vol. 31, no. 6, pp. 878–889, 2012.

[12] B.-Q. Lin, T.-C. Lin, and Y.-W. Chang, "Redistribution Layer Routing for Integrated Fan-Out Wafer-Level Chip-Scale Packages," in *Proc. of ICCAD*, Austin, TX, November 2016, pp. 1–8.

[13] H.-T. Wen, Y.-J. Cai, Y. Hsu, and Y.-W. Chang, "Via-Based Redistribution Layer Routing for InFO Packages with Irregular Pad Structures," in *Proc. of DAC*, San Francisco, CA, July 2020, pp. 1–6.

[14] Y.-J. Cai, Y. Hsu, and Y.-W. Chang, "Simultaneous Pre- and Free-Assignment Routing for Multiple Redistribution Layers with Irregular Vias," in *Proc. of DAC*, San Francisco, CA, December 2021, pp. 1147–1152.

[15] K. Yang, H. Yao, T.-Y. Ho, K. Xin, and Y. Cai, "AARF: Any-Angle Routing for Flow-Based Microfluidic Biochips," *IEEE TCAD*, vol. 37, no. 12, pp. 3042–3055, 2018.

[16] T. Honda and Y. Kohira, "An Acceleration for Any-Angle Routing Using Quasi-Newton Method on GPGPU," in *Proc. of IEEE MCSoC*, Aizu-Wakamatsu, Japan, September 2014, pp. 281–288.

[17] Y. Kohira and A. Takahashi, "An Any-Angle Routing Method Using Quasi-Newton Method," in *Proc. of ASPDAC*, Sydney, Australia, February 2012, pp. 145–150.

[18] P. Spindler and F. M. Johannes, "Fast and Accurate Routing Demand Estimation for Efficient Routability-Driven Placement," in *Proc. of DATE*, Nice, France, April 2007, pp. 1–6.

[19] M. A. Zapletina, D. A. Zheleznikov, and S. V. Gavrilov, "Improving Pathfinder Algorithm Perfomance for FPGA Routing," in *Proc. of IEEE EIConRusNW*, Moscow, Russia, January 2021, pp. 2054–2057.

[20] *C++ Library for Generating Constraint or Conforming Delaunay Triangulations*, 2018. [Online]. Available: https://www.github.com/artemogre/CDT