Redistribution Layer Routing with Dynamic Via Insertion Under Irregular Via Structures

Je-Wei Chuang[1], Zong-Han Wu[1], Bo-Ying Huang[2], 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
jwchuang@eda.ee.ntu.edu.tw; zhwu@eda.ee.ntu.edu.tw; b09901089@ntu.edu.tw; ywchang@ntu.edu.tw

ABSTRACT

In modern advanced packaging, redistribution layers (RDLs) are often used for signal transmission among chips, and vias are used for communication among different layers. Most existing RDL routers perform via planning before routing. However, since vias can be placed at arbitrary locations under the irregular via structure, via planning limits the solution space and reduces layout flexibility. This paper proposes a new flow with a novel routing graph model for 90- and 135-degree routing, which allows dynamic via insertion during routing. The proposed algorithm enlarges the solution space by providing more choices during path-finding, achieving higher routing quality. The experimental results based on commonly used benchmark suites show that our router achieves over 10% better wirelength with over 29X speedup over the state-of-the-art work and even achieves 0.4% better wirelength with 55X speedup over the state-of-the-art any-angle router.

CCS CONCEPTS

• **Hardware** → **Packaging**; **Wire routing**.

KEYWORDS

Heterogeneous integration, Redistribution layer routing, X-architecture routing

1 INTRODUCTION

As the complexity of integrated system design continues to grow, heterogeneous integration technologies have emerged as promising solutions for both the design and cost aspects. Among those packaging technologies, the integrated fan-out (InFO) wafer-level chip-scale package (WLCSP) developed by Taiwan Semiconductor Manufacturing Company (TSMC) has exhibited significant advantages in connection densities, power efficiency, and thermal properties [1–6]. The InFO WLCSP is introduced for modern system-in-package (SiP) designs with many I/Os and higher interconnection densities.

In InFO packages, multiple chips with different technology nodes are integrated into one package, and the inter-chip and chip-to-board connections are completed by multiple redistribution layers (RDLs) below. The cross-sectional view of an InFO package is shown in Figure 1. The topmost part in pink is the molding compound that serves as a base for the package and provides physical protection for chips and passive devices. The bottom parts are bump pads that would connect to the printed circuit board (PCB). The layers in the middle are the RDLs, composed of alternating wire layers and via layers, which facilitate intra-layer and inter-layer connections, respectively. This work focuses on the RDL routing problem under the irregular via structure. That is, vias can be placed anywhere in the via layers.

This work was partially supported by AnaGlobe, ASUS, Delta Electronics, Google, Maxeda Technology, TSMC, NSTC of Taiwan under Grant NSTC 110-2221-E-002-177-MY3, NSTC 112-2218-E-002-033, and NSTC 112-2223-E-002-020 .

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3657371

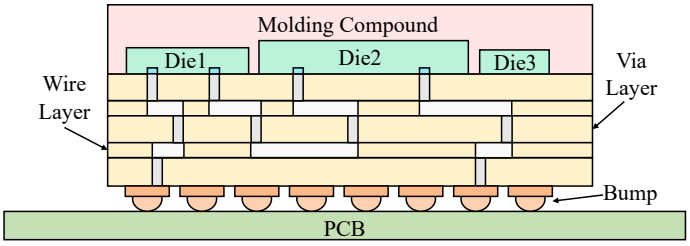

Figure 1: Side view of an InFO package.

1.1 Previous Works

The RDL routing problem has been evolving with packaging technology, from the initial single-layer structure to the more advanced multi-layer structure adopted in InFO packages. For the single-layer structure, Fang *et al.* [7] proposed the first router for flip-chip designs using an RDL. The router used the Minimum-Cost Maximum-Flow (MCMF) algorithm with channels modeled as vertices on the routing graph. Fang *et al.* [8] formulated the RDL routing problem using integer variables and applied integer linear programming (ILP) to optimize the routing solution. Fang *et al.* improved their previous network-flow-based router to handle a more general net assignment [9]. Lin *et al.* [10] guided global routing by computing the longest common subsequence (LCS) and the maximum planar subset of chords (MPSC). They also observed that the previous routing capacity model tended to over-constrain the capacity, confining the solution space. Therefore, they introduced a more accurate tile capacity constraint model for ILP. Yan and Wong mentioned a similar observation in [11]. They pointed out that previous works often failed in modeling diagonal capacity and introduced a novel network-flow tile model along with proof of the correctness of the capacity constraints. This model used five vertices representing the center and boundaries in four directions and four types of capacity constraint in a tile. Lee *et al.* [12] extended the model to handle obstacles with guaranteed routing feasibility.

As the multi-layer structure matured, multi-layer RDL routers became essential for finding better solutions. Lin *et al.* [13] was the first work to handle the multi-layer multi-chip RDL routing problem. One of the most critical tasks in the multi-layer structure is to assign the layer of each net, which was handled by a concentric circle model proposed in Lin's work. After the assignment, each net was routed only in the assigned layer by congestion-aware escape routing. Wen *et al.* [14] further took advantage of the multi-layer structure by using RDL vias. They introduced weighted MPSC to perform single-layer concurrent routing to route a portion of nets. The remaining nets were then routed using A*-search over the routing graph constructed with vias, which enabled the routes to switch to another layer through vias. Routed nets were then formulated as obstacles in the routing graph. Subsequently, Cai *et al.* [15] developed a novel simultaneous routing framework called crossing-aware A*-search. They partitioned the routing space by the Manhattan-distance-based Voronoi Diagram into tiles, then used a chord-based tile model with net-sequence lists to keep track of the order of net access points inside a tile so that illegal assignments due to wire crossings could be avoided. Their algorithm performed routing without reconstructing the entire routing graph, which was much more efficient. Chen *et al.* [16] proposed an RDL routing algorithm that can handle obstacles in the routing region. They used *Constrained Delauney Triangulation* (CDT) to partition the routing region and combine with projection to correctly estimate the routing resource. A Rubber-band-based pattern routing and a dynamic-programming-based bend minimization were proposed in detailed routing. Chung *et al.* [17] proposed the first any-angle RDL router. They also applied CDT to partition the routing region and adopted the crossing-aware A*-search to generate crossing-free

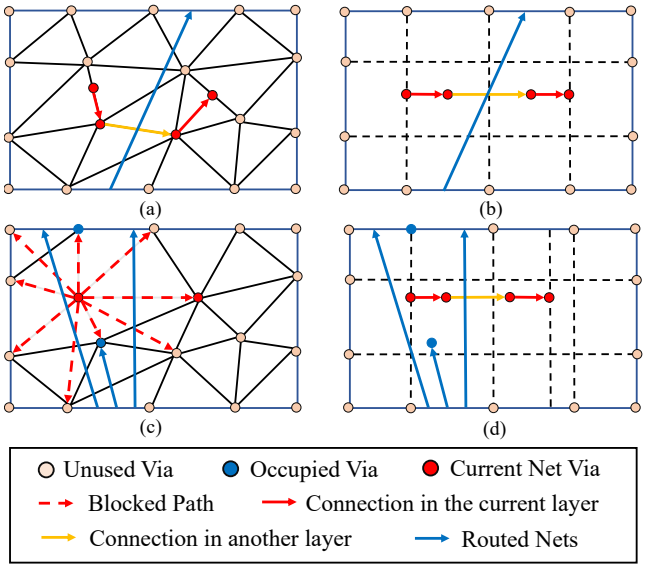

Figure 2: (a) Detour is needed for the pre-placed via structure. (b) Dynamic via insertion enables nets to bypass obstacles easily. (c) No available via is allowed for escaping. (d) Inserting via dynamically enables nets to escape easily.

guides. In addition, they also proposed a multi-net access point adjustment method based on dynamic programming to optimize the layout. To use the any-angle structure, they applied a greedy-based algorithm to connect access points in the shortest wirelength.

1.2 Motivation

In modern RDL routing, the irregular via structure provides layout flexibility by allowing vias to be placed at arbitrary locations in the routing region. However, most existing RDL routing works insert vias before routing to construct a 3D routing graph with limited layer-changing resources. Such flow cannot fully utilize the advantages of the irregular via structure. As shown in Figure 2(a), on the routing graph with fixed vias, the connection between two red points must detour to bypass the obstacle since there is no via between them. In contrast, if inserting vias dynamically during routing is allowed, as shown in Figure 2(b), we can insert a via at the location we need. Moreover, dynamic via insertion also helps improve pin accessibility. Current RDL routers spend much of their execution time finding a feasible solution by changing the net order since some I/O pins cannot be accessed after several routing iterations. As shown in Figure 2(c), routed nets trap the red point, and there is no pre-placed via located at the proper location, allowing escape. On the other hand, in the dynamic via insertion framework, as shown in Figure 2(d), we can insert a via to solve the problem. Thus, to improve the RDL routing quality, it is desirable to develop a routing algorithm compatible with dynamic via insertion.

1.3 Our Contributions

The contribution of this paper is summarized as follows:

- We propose a new routing flow in which vias can be inserted dynamically during routing. Our proposed flow exhibits better layout flexibility and is more likely to find high-quality solutions.
- We propose a 3D routing graph model allowing dynamic via insertion, which effectively estimates the routing resource consumption of vias.
- We propose a constrained A*-search and backtrace topology determination method for finding paths on the routing graph, which searches effectively in the larger solution space.
- We develop a tile routing algorithm to perform legalization and optimization in a tile.
- Experimental results show that our algorithm achieves over 10% better wirelength with over 29X speedup over the state-of-the-art work and even achieves 0.4% better wirelength with 55X speedup over the state-of-the-art any-angle router.

2 PRELIMINARIES

In this section, we first give the terminologies and notations used in this paper and then formulate the RDL routing problem.

2.1 Terminologies and Notations

We will use the following terminologies and notations:

- $N = \{n_i \mid 1 \le i \le |N|\}$: the set of all nets.
- $T = \{T_i \mid 1 \le i \le |T|\}$: the set of all routing graph tiles.
- $V = \{v_i \mid 1 \le i \le |V|\}$: the set of all routing graph nodes.
- S_v^i: the i^{th} net sequence on node v.
- w_w: the wire width.
- w_s: the minimum spacing.

2.2 RDL Routing Constraints

We consider the RDL routing problem under irregular via structure and following design rules:

- Irregular via structure: The vias can be placed anywhere in the layout.
- X-architecture: Wire segments can only be routed in vertical, horizontal, and 45°/135° diagonal directions.
- Non-crossing rule: Crossing between two nets in an RDL is not allowed.
- Minimum spacing rule: A minimum spacing between two vias or wire segments belonging to different nets is needed.
- Routing-angle constraint: Two connected segments can only have a right angle or a 135° turn. A 45° turn is not allowed because of manufacturability issues.

2.3 Problem Formulation

We define the RDL routing problem as follows:

Problem 1 (**X-Architecture RDL Routing Under Irregular Via Structure**). *Given an RDL structure, design rules, a netlist of pre-assigned nets, and a set of I/O pads and bump pads, generate a layout with maximized routability and minimized wirelength without design rule violations.*

3 PROPOSED ALGORITHMS

In this section, we first give an overview of the whole algorithm flow and then go into the details of each step. Figure 3 shows the flow of the proposed algorithm, which consists of three major stages: (1) *Preprocessing*, (2) *Global Routing*, and (3) *Detailed Routing*. In the preprocessing stage, we first partition our routing region into rectangular tiles, then construct a multi-layer routing graph without pre-placed vias based on the partitioning result. In the global routing stage, we perform a constrained 3D A*-search on the routing graph constructed in the previous stage, which has special constraints when switching between layers. Then, we backtrace through the path searched in the A*-search process to obtain a crossing-free topology. In the detailed routing stage, we modify the location of access points on tile boundaries and place vias based on the topology generated in the global routing stage to obtain routing guides. Last, we route the nets tile by tile. These stages will be detailed in the following sections.

3.1 Preprocessing

In the preprocessing stage, we first extend the boundaries of each die to form an initial partitioning result. We add extra cuts for tiles with too large dimensions until their widths and heights are smaller than a user-defined value. After partitioning the layout into rectangular tiles, we construct the 3D routing graph.

3.1.1 Routing Region Partitioning. To apply our proposed algorithm, fixed I/O pads must be on the boundaries of a tile. As shown in Figure 4, we extend die boundaries to form a set of cuts, guaranteeing that every I/O pad is on at least one cut. Moreover, to improve the resolution of the global routing algorithm, we further partition tiles with too large dimensions into smaller tiles. The process terminates until all tiles have widths and heights smaller than a user-defined value.

3.1.2 Routing Graph Construction. After partitioning the layout into rectangular tiles, we construct the routing graph based on the result. We model each tile by the graph shown in Figure 5(a). The graph is mainly

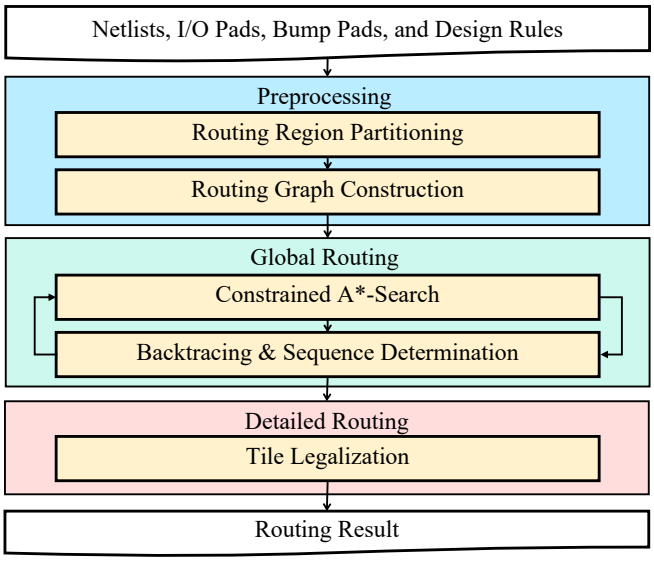

Figure 3: Our algorithm flow.

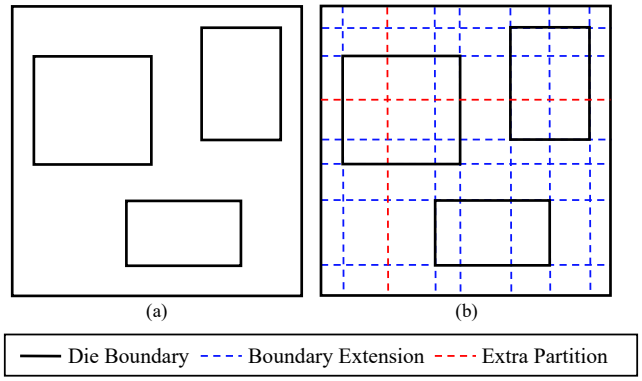

Figure 4: (a) Top view and (b) the partitioning result of the routing region.

composed of two kinds of search nodes: (1) *tile nodes* and (2) *edge nodes*, and three kinds of search edges: (1) *diagonal edges*, (2) *regular edges*, and (3) *change-layer edges*.

Tile nodes are nodes modeling center regions of tiles. A tile node is a must-pass node when a guide tries to cross the tile or switch between layers through via in the tile. Tile nodes have no capacity constraint, but extra checks will be performed during global routing to ensure the tile is not congested. Edge nodes are nodes modeling the boundaries of tiles, and their capacities are computed as follows:

$$C^k_{i,j} = \lfloor d(p_{i,k}, p_{j,k})/(w_w + w_s) \rfloor, \tag{1}$$

where $p_{i,j}$ is the i^{th} vertex of the j^{th} tile, $d(p_{i,j}, p_{i',j'})$ is the distance between $p_{i,j}$ and $p_{i',j'}$, and $C^k_{i,j}$ is the capacity of the edge between $p_{i,k}$ and $p_{j,k}$.

Diagonal edges connect adjacent edge nodes, and regular edges connect edge nodes and tile nodes. Since the utility check will be performed on tiles during global routing, these edges have no capacity constraint. Change-layer edges connect tile nodes in different layers at the same location. A change-layer edge in a routing guide is equivalent to a via in the corresponding tile. The layout may become hard to legalize if more than one via exists in a single tile. Therefore, we set the capacity of the change-layer edge to one to prevent the situation.

If there are I/O pads on the tile boundary, we need to partition the edge further to record the topology. As shown in Figure 5(b), we partition the edge with I/O pads into sub-edges and I/O vias and treat those generated components as edge nodes, and add routing graph edges between them. The capacity of a sub-edge is calculated in the same way as an edge node, and the capacity of an I/O via is one. An I/O via can only be utilized by its corresponding net.

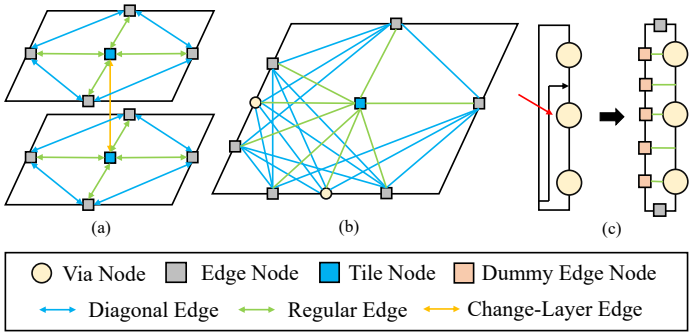

Figure 5: (a) The basic routing graph model. (b) The routing graph model when there are vias on the boundaries of a tile. (c) A constrained version of the routing graph model for tiles that are too thin and have vias on one of their edges.

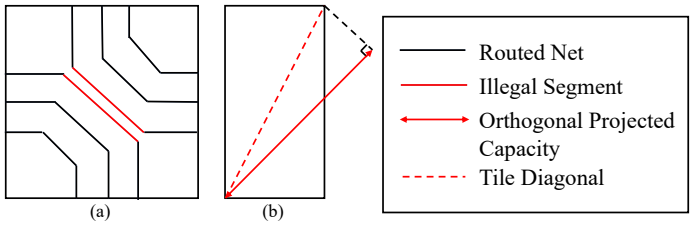

Figure 6: (a) Diagonal capacity violation, which cannot be detected by the capacity constraint on edges. (b) The diagonal capacity calculation for non-square tiles.

Moreover, since the die distribution affects the initial partition, some thin tiles might not allow a route to proceed in a certain direction. As shown in Figure 5(c), the black net results in a spacing violation with the via on the boundary, while the via limits the location where the access point can be placed. We create dummy nodes on the boundary to avoid generating a non-legalizable layout and only keep the direct connections.

Our routing model guarantees that the number of guides that pass through the boundary of a tile is acceptable. However, as stated in [11], there might be violations in the diagonal direction. As shown in Figure 6, the red segments violate the minimum spacing rule, while there is no capacity violation on each boundary edge. Thus, we additionally check the diagonal usage of each tile during global routing, and the diagonal capacity of a tile is computed as follows:

$$C_i = \lfloor \max(l^1_i, l^{-1}_i, l^0_i, l^\infty_i)/(w_w + w_s) \rfloor, \tag{2}$$

where C_i and l_i are the diagonal capacity and the diagonal of T_i, and l^1_i, l^{-1}_i, l^0_i, and l^∞_i denote the length of projection of l_i on the four X-architecture line, whose slopes are 1, −1, 0, and ∞, respectively.

3.2 Global Routing

In the global routing stage, we perform the constrained A*-search, which searches over multiple topologies in a single iteration to obtain a routing guide prototype that can be legalized. Then, in the backtracing stage, we determine the exact topology of the routing guide based on the same algorithm we use for searching and updating the routing resource usage. We will repeat the process until all routing guides are generated. The details of each step are as follows:

3.2.1 Constrained A-search.* In our search algorithm, the path cost is defined as the sum of distances under X-architecture between the geometric centers of nodes on the path, and the heuristic cost is defined as the distance under X-architecture between the geometric center of the current node and the target. Define $f(x)$ to be our cost function, and $g(x)$ and $h(x)$ are the path cost and the heuristic cost in the A*-search, respectively. We have

$$f(x) = g(x) + h(x), \tag{3}$$

$$g(x) = \sum_{i=2}^{|P_{s,x}|} d_x(v_{i-1}, v_i), \tag{4}$$

$$h(x) = d_x(x, t), \tag{5}$$

where t is the target node, and $P_{s,x} = \{v_i | 1 \le i \le |P_{s,x}|\}$ is the set consisting of nodes on the path from the source node s to the current node x, where v_1 is the source node and $v_{|P_{s,x}|}$ is the current node. The distance under X-architecture is defined as follows:

$$d_x(v_i, v_j) = d_{max}(v_i, v_j) + (\sqrt{2} - 1) \times d_{min}(v_i, v_j), \tag{6}$$

$$d_{min}(v_i, v_j) = \min(|x_{v_i} - x_{v_j}|, |y_{v_i} - y_{v_j}|), \tag{7}$$

$$d_{max}(v_i, v_j) = \max(|x_{v_i} - x_{v_j}|, |y_{v_i} - y_{v_j}|), \tag{8}$$

where x_{v_i}, y_{v_i} are the x-coordinate and the y-coordinate of v_i, respectively, and $d_x(v_i, v_j)$ is the distance between v_i and v_j under X-architecture. Since the distance under X-architecture satisfies the triangle inequality, the actual path cost from x to t must be greater than the distance between them, and the heuristic cost never overestimates the actual path cost.

Theorem 1. *Our A*-search algorithm guarantees to find a shortest path.*

Proof 1. *Consider an arbitrary path $P_{x,t}$ from x to t, $v'_i \in P_{x,t}$ are nodes on the path. By the triangle inequality, we have:*

$$h(x) = d_x(x, t) \le \sum_{i=2}^{|P_{x,t}|} d_x(v'_{i-1}, v'_i). \tag{9}$$

Thus, the heuristic cost $h(x)$ never overestimates the actual cost. As a result, our cost function satisfies the admissibility property [18], and thus our A-search algorithm guarantees to find a shortest path.*

Moreover, we impose some extra constraints in addition to those intrinsically given by the routing graph. A node is pushed into the queue for further searching only if it passes all these checks.

(1) **Topology Check:** During the constrained A*-search, we maintain a net-sequence list in each node to record the current topology. Since crossing topology on the layout always leads to incompatible net sequences on some nodes, maintaining the net sequence guarantees that our layout is topologically crossing-free. Figures 7(a) to (c) show illegal topologies resulting from an inaccessible edge, an inaccessible sub-edge, and an inaccessible tile center, respectively. As shown in Figure 7(d), except for explicit topology violations, we also forbid accessing a tile node from different directions in a single tile. Accessing a tile node from two different directions implies that two nets with different orientations are crossing the same tile or there is a net accessing via with a direction different from other nets crossing the tile. The former turns into a crossing topology, and the latter limits the physical location where via can be placed, which might result in an illegal layout.

(2) **Via Location Check:** Since vias are inserted dynamically during the guide generation process, we have yet to determine the exact location of vias before detailed routing. During the global routing stage, we must ensure that the via can be legally placed when using the change-layer edge. In other words, we must have a physical location where the via can be placed in two layers the via connects. First, we must make sure there is enough space. As shown in Figure 8, placing a via occupies routing resources on both horizontal and vertical edges and the diagonal of the tile. Thus, stricter capacity checks should be performed during layer changes, and the resources consumed should be recorded during backtracing. The net direction of tiles also needs to be considered. As shown in Figure 9(a), if two layers connected by the via have nets with the same direction, the via might not be able to be placed because of the layout topology. Thus, we constraint that the change-layer edge can be used only when the corresponding tile in both layers has nets with different directions. Under this circumstance, as shown in Figure 9(b), each layer determines a coordinate of the via location; thus, it can always be placed legally.

By applying these checks, the generated guides are guaranteed to be legalizable. Moreover, since the via location is not determined in a layer, we can search over multiple topologies when changing the layer by adjusting the via location. As shown in Figure 10, for the red net coming from another layer, we can place the via between blue and green net or between green and yellow net, which leads to entirely different topologies. Our constrained A*-search will find paths to all nodes that any topologies can access, and the final topology will be determined after backtracing. We

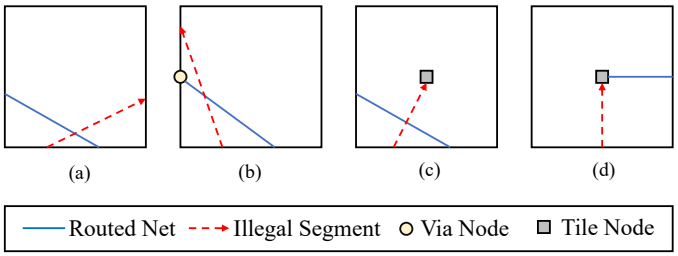

Figure 7: Four types of illegal segments. (a) Segment accessing an inaccessible edge. (b) Wrong geometric relation on sub-edge. (c) Segment accessing tile center while blocked. (d) Segments with different directions access the tile center.

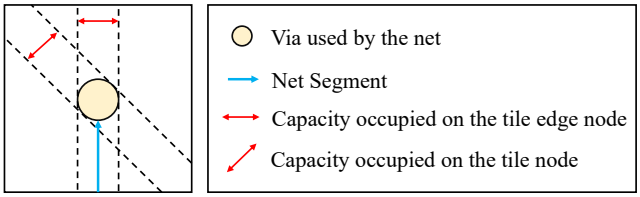

Figure 8: Capacity occupied by via.

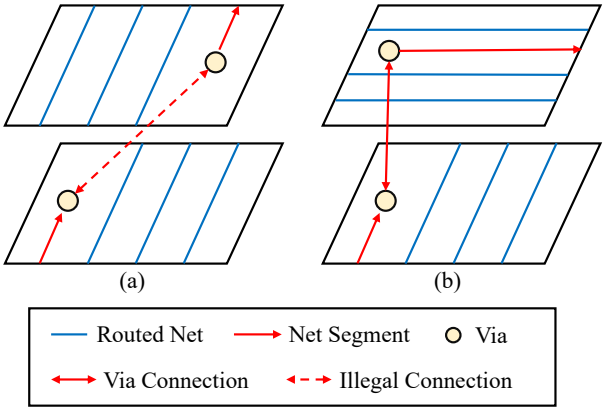

Figure 9: (a) Two layers with the same wire direction, resulting in no legal location to place the via. (b) Two layers with different wire directions, each constraint only one coordinate of the via location.

do not need to record all these topologies, but only a start and an end terminal, since all possible net sequences for a net n on a node v_i must be continuous.

Theorem 2. *For a net n on a node v_i in an A*-search iteration, if net sequences $(n, n_1, n_2 ... n_k)$ and $(n_1, n_2 ... n_k, n)$ on v_i are both feasible, $(n_1, n, n_2 ... n_k)$, $(n_1, n_2, n ... n_k)$..., $(n_1, n_2 ... n, n_k)$ are also feasible on v_i.*

Proof 2. *Suppose the statement is not true. We consider the first $1 \le s < k, s \in \mathbb{N}$ such that $(n_1, n_2 ... n_s, n ... n_k)$ is legal and $(n_1, n_2 ... n_{s+1}, n ... n_k)$ is not. Under this circumstance, net n must be blocked by net n_{s+1}. Thus, for all $t > s, t \in \mathbb{N}$, $(n_1, n_2 ... n_t, n ... n_k)$ is illegal, which contradicts with the fact that $(n_1, n_2 ... n_k, n)$ is legal. By contradiction, we prove that the all legal net sequences on a node must be continuous.*

By the theorem above, we only need to take care of the boundary net of all legal net sequences, making the path-finding process much more efficient. This topology-preserving mechanism optimizes the topological relation between the newly placed via and existing guides, helping us find the solution much more easily.

3.2.2 Backtracing and Sequence Determination. After finding a path between two terminals of a net, we need to confirm its topology. Since we search over multiple topologies in an iteration, the net sequence on some nodes might be undetermined. To find a legal solution, we start backtracing from the target node of the search process, applying the search algorithm

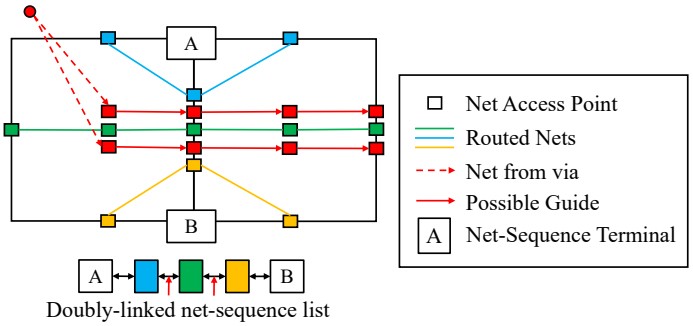

Figure 10: The two red guides in the graph represent two different legal topologies. The search algorithm will keep both of them until the path is found.

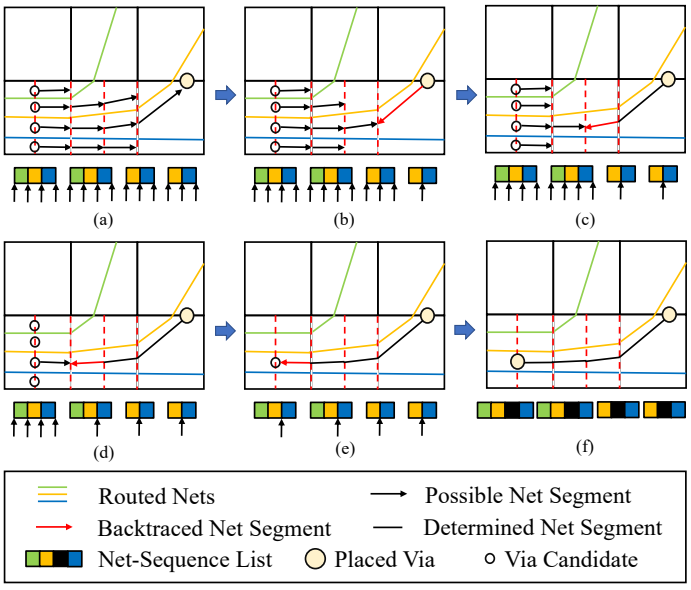

Figure 11: The backtracing process.

in the A*-search stage with the searched node constrained to its previous node and the net sequence constrained to the net sequence candidates for the node. Such a process guarantees a feasible topology, and the result is treated as the global routing guide.

Theorem 3. *The backtracing process guarantees a feasible topology.*

Proof 3. *Suppose we cannot find a legal topology, implying that for some node v, all available net sequence candidates of its previous node is not compatible with the current net sequence on v. However, this is impossible since net sequence S_v^i on v will be recorded during the searching step only if there are some net sequences $S_{v_{prev}}^j$ from its previous node v_{prev} are compatible with it. Thus by contradiction, we prove the theorem.*

As shown in Figure 11, after an A*-search iteration, the net sequences on red edges are not determined. We start backtracing from the target terminal until we reach the source terminal. In each step, the backtraced segment (the red arrow) determines the net sequence, and we can finally get a routing guide with a fixed topology.

3.3 Detailed Routing

In detailed routing, we transform the routing guides into the physical layout. First, for access points on tile boundaries, we distribute them equally on the edges. For those tiles with via in their center, as shown in Figure 12, we adjust the access point to match the coordinate to avoid getting blocked by via. After adjusting access points, we start routing the net tile by tile. We route nets in a tile from the corners to the center.

As shown in Figure 13, we first route corner nets and then the cross-tile nets. For the tile routing algorithm, we apply a similar greedy method

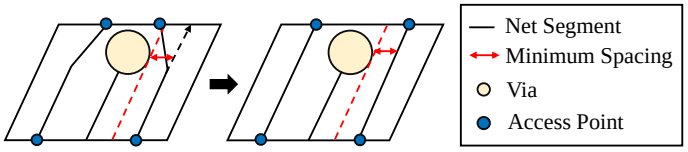

Figure 12: Adjusting access point to avoid getting blocked by via.

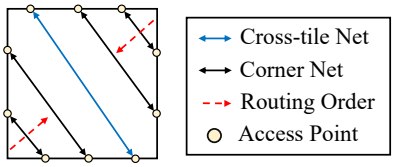

Figure 13: The routing order of tile routing. Cross-tile nets are routed after all corner nets are routed.

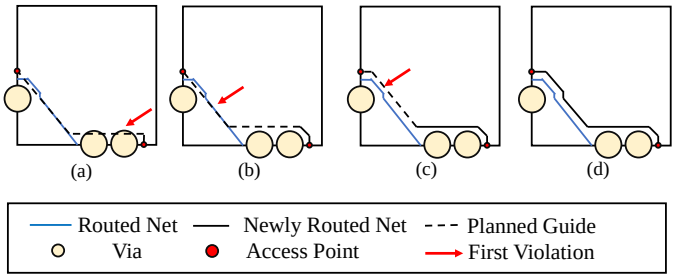

Figure 14: The detailed routing process. (a) The planned guide. (b) Adjust the guide to bypass the via, and adopt the route closer to the tile boundary. (c) Adjust the guide according to routed net to fix violation. (d) The final layout.

as in [17], pushing the current net toward the tile boundary under the constraint that no detour occurred. Under X-architecture, the shortest path connecting two points can only consist of segments with two directions. As shown in Figure 14, we first plan a guide that minimizes the wirelength and is closest to the tile boundary of these directions. We then route each segment one by one. When a spacing violation occurs, we adjust the direction of the current segment to bypass the obstacle. Since the generated net segment is confined in the rectangular region created by the two access points, we can always generate a feasible layout if the topology is accepted in the global routing stage.

4 EXPERIMENTAL RESULTS

We implemented our RDL router in the C++ programming language on a 2.9GHz AMD Ryzen 3990X Processor with 126GB memory. We performed experiments on the RDL circuit benchmarks in [17] and [14]. Both benchmarks consist of five dense designs with many flight lines; the statistics are in Table 1 and Table 2. "#*Chips*," $|IO|$, $|B|$, $|N|$, $|L_w|$, and $|W_d|$ represent the number of chips, I/O pads, bumps, nets, RDL layers, and wire spacing, respectively.

We compared our algorithm with two state-of-the-art RDL routers, named *Cai* and *Wen*, and the state-of-the-art any-angle RDL router, named *AA*, to evaluate its effectiveness. The results of these works were obtained directly from the papers[14, 15, 17]. All routers reach 100% routability, and the total wirelength and running time were reported. We also gave a lower bound on wirelength under the X-architecture obtained by directly calculating the X-shaped distance between terminals of each net, named X_{min}.

Table 3 and Table 4 show that our algorithm gives the shortest wirelength among X-shaped routers for all cases and 4 cases with wirelength shorter than the any-angle router. Comparing our result with X_{min}, we observed that our wirelength is only an average of about 4.4% longer than the wirelength lower bound, which implies that our solution is very close to the optimal value in terms of wirelength. Considering the running time, our algorithm achieves 828X acceleration with *Wen*, 29X acceleration

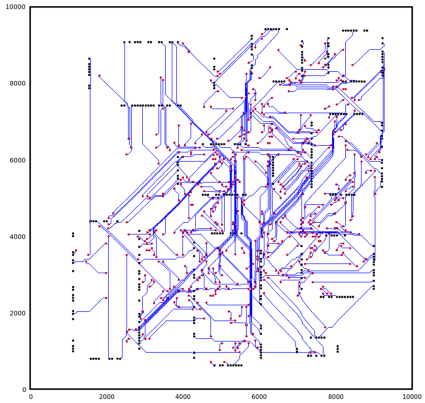

Figure 15: Layout of the second layer in dense5. The red circles are vias placed during routing, and the black circles are I/O vias.

with *Cai*, and 55X acceleration with the any-angle router. The reason that our proposed algorithm has a significant wirelength and running time improvement is analyzed as follows:

- Under the fixed-via framework, nets need detours to find a via available to bypass the routed nets. In our proposed algorithm, we can insert a via wherever we need it, which means that we do not need to detour as long as there is enough space for via insertion.
- *Wen* reconstructs the routing graph after a net is routed, which is time-consuming. By applying the constrained A*-search, our proposed algorithm only constructs the routing graph once, which is more efficient.
- Because of the limited solution space, *Cai* and *AA* require multiple iterations of rip-up and reroute to find a suitable net order to generate a feasible solution. Our proposed algorithm allows dynamic via insertion, which makes the solution space much more extensive and thus alleviates the problem.
- When switching between layers, our algorithm looks over multiple topologies in a single iteration by preserving multiple feasible net sequences. This mechanism makes the solution-finding process more efficient.

Table 1: Benchmark statistics in [14]

Circuit	#*Chips*	$\|IO\|$	$\|B\|$	$\|N\|$	$\|L_w\|$	$\|W_d\|$
mLayer1	2	44	324	22	3	4
mLayer2	3	92	784	46	3	4
mLayer3	5	158	308	80	5	4
mLayer4	6	222	684	111	5	10
mLayer5	9	522	1444	261	5	4

Table 2: Benchmark statistics in [15] and [17]

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

5 CONCLUSIONS

This paper has proposed an RDL routing algorithm that can fully utilize the irregular structure's layout flexibility. We have proposed a 3D routing graph model that allows dynamic via insertion to fully use the layout flexibility provided by the irregular via structure. We have proposed a constrained A*-search algorithm combined with a backtrace topology determination method for the path-finding algorithm to generate crossing-free guides efficiently. We have developed a legalization process consisting of access point adjustment and a greedy-based tile routing algorithm for detailed routing. Experimental results have shown that our algorithm is much more efficient than previous works and performs better in terms of wirelength.

Table 3: Wirelength and running time compared with the router in [14].

Case	Wirelength (μm)			Running time (s)	
	Wen	X_{min}	*Ours*	*Wen*	*Ours*
mLayer1	81302	75071	76429	8.6	0.07
mLayer2	206816	193551	197689	36.3	0.09
mLayer3	266288	210634	220206	56.6	0.10
mLayer4	495255	430966	462487	197.3	0.08
mLayer5	1776860	1421970	1495930	627.1	1.07
Comp.	1.115	0.960	1	828.9	1

Table 4: Wirelength and running time compared with routers in [15] and [17].

Case	Wirelength (μm)				Running time (s)		
	AA	*Cai*	X_{min}	*Ours*	*AA*	*Cai*	*Ours*
dense1	74640	86695	74755	76873	0.415	0.79	0.05
dense2	206451	236269	193224	204887	0.703	0.60	0.08
dense3	222601	258172	208745	222105	3.68	3.92	0.06
dense4	461725	512619	430533	457645	23.3	9.13	0.17
dense5	1512849	1835410	1420950	1463690	180.07	12.5	2.87
Comp.	1.004	1.163	0.953	1	55.64	29.34	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] H. Chen, H.-C. Lin, C.-N. Peng, and M.-J. Wang, "Wafer Level Chip Scale Package Copper Pillar Probing," in *Proc. of ITC*, Seattle, WA, October 2014, pp. 1–6.

[5] D. Yu, "A New Integration Technology Platform: Integrated Fan-Out Wafer-Level-Packaging for Mobile Applications," in *Proc. of Symp. VLSI Technol.*, Kyoto, June 2015, pp. T46–T47.

[6] Y.-C. Huang, B.-Y. Lin, C.-W. Wu, M. Lee, H. Chen, H.-C. Lin, C.-N. Peng, and M.-J. Wang, "Efficient Probing Schemes for Fine-Pitch Pads of InFO Wafer-Level Chip-Scale Package," in *Proc. of DAC*, Austin, TX, June 2016, pp. 1–6.

[7] 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.

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

[9] 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.

[10] 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 Tran. on CAD*, vol. 31, no. 6, pp. 878–889, 2012.

[11] T. Yan and M. D. F. Wong, "Correctly Modeling the Diagonal Capacity in Escape Routing," *IEEE Tran. on CAD*, vol. 31, no. 2, pp. 285–293, 2012.

[12] P.-W. Lee, H.-C. Lee, Y.-K. Ho, Y.-W. Chang, C.-F. Chang, I.-J. Lin, and C.-F. Shen, "Obstacle-Avoiding Free-Assignment Routing for Flip-Chip Designs," in *Proc. of DAC*, San Francisco, CA, June 2012, pp. 1088–1093.

[13] 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.

[14] H.-T. Wen, Y.-J. Cai, Y. Hsu, and Y.-W. Chang, "Via-Based Redistribution Layer Routing for InFO Packages With Irregular Pad Structures," *IEEE Tran. on CAD*, vol. 41, no. 12, pp. 5554–5567, 2022.

[15] 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.

[16] Y.-T. Chen and Y.-W. Chang, "Obstacle-Avoiding Multiple Redistribution Layer Routing with Irregular Structures," in *Proc. of ICCAD*, San Diego, CA, October/November 2022, pp. 1–6.

[17] M.-H. Chung, J.-W. Chuang, and Y.-W. Chang, "Any-Angle Routing for Redistribution Layers in 2.5D IC Packages," in *Proc. of DAC*, San Francisco, CA, July 2023, pp. 1–6.

[18] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.