

An Adaptive Cost-based Via and Congestion Co-optimization Framework for VLSI Global Routing

Zhaoyi Wu¹, Haishan Huang², Jianli Chen³, Zhifeng Lin^{4*}

¹College of Computer and Data Science, Fuzhou University, Fuzhou, China

²Shanghai LEDA Technology Co., Ltd., Shanghai, China

³School of Microelectronics, Fudan University, Shanghai, China

⁴Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, Fuzhou, China

Emails: 231027061@fzu.edu.cn, linzhifeng@fzu.edu.cn

Abstract—Global routing is a critical stage in VLSI physical design, directly affecting the final Power, Performance, and Area (PPA) metrics. In this paper, we propose a high-performance global router that optimizes via count and routing congestion simultaneously. We first generate a 2D via-aware spine tree, which incorporates bend cost to reduce via usage while minimizing wire length. Then, a fast maze routing algorithm is employed to efficiently find a blockage-free path, followed by a congestion-aware layer assignment method to generate the 3D routing solution. Finally, we present an iterative rip-up and reroute strategy to resolve remaining congestion using the 3D bidirectional A* search. The A* search is guided by an adaptive cost function that dynamically adjusts via costs based on congestion, facilitating the co-optimization of congestion and via count. Compared to an advanced commercial tool and the leading academic engine OpenROAD, our algorithm achieves the best results in both overflow and via count, while preserving almost the same wire length.

Index Terms—global routing, via count, congestion, VLSI

I. INTRODUCTION

As the feature size keeps shrinking, more and more issues like via spacing, timing, and reliability need to be considered during the routing phase. Consequently, routing has become one of the most challenging phases in the VLSI physical design flow. To manage the complexity of the routing problem, it is typically divided into three steps: global routing, track routing [1], and detailed routing [2], [3]. The global routing stage first partitions the routing region into an array of global cells (G-cells) and then determines a coarse path of G-cells for each net. Next, the track routing stage assigns each straight wire segment to a specific routing track within the G-cells it traverses. Finally, detailed routing realizes the interconnections between pins and wire segments for global nets and constructs routing trees for local nets. As a critical step in this process, global routing directly

impacts the final power, performance, and area (PPA) of the chip and thus has been extensively studied [4].

Existing global routing approaches can be broadly classified into two categories. The first category, 2D global routing followed by layer assignment, is widely adopted in both industry and academia for its favorable balance between runtime and solution quality. Several routers such as NTU-GR [5], NTHU-Route 2.0 [6], NCTU-GR 2.0 [7], SPRoute 2.0 [8] and FastRoute 4.0 [9] all belong to this class. Typically, the flow for these routers involves decomposing each multi-pin net into two-pin nets using FLUTE [10]. Then, pattern routing or monotonic routing are employed to quickly generate an initial topology, followed by iterative Rip-up and Reroute (R&R) to resolve congestion.

The second category is routing directly in 3D space. FGR [11] utilizes maze routing to directly rip-up and reroute nets based on a discrete Lagrangian cost framework. GRIP [12] employs Integer Linear Programming (ILP) for optimal path selection. Both methods demonstrate high-quality results but can suffer from long runtimes due to the vast solution space. MGR [13] uses pattern routing and layer assignment to obtain an initial 3D solution and then employs 3D maze routing within a multi-level framework for accelerated rip-up and reroute of nets in congested areas. The work CUGR [14] uses 3D pattern routing based on dynamic programming for its initial solution and multi-level 3D maze routing for R&R, simultaneously considering wire length, DRC violations, and runtime.

With the continuous increase of design complexity, via has become a crucial issue in the VLSI domain. A large number of vias can reduce manufacturing yield, degrade circuit performance, and increase the layout area required for interconnections. In addition, due to the high integration induced by the advanced manufacturing technology, there remains a significant challenge in overflow elimination. It is thus desirable to develop effective routers to address the critical via and overflow issues. In this work, we present an adaptive cost-based routing framework to optimize via count and congestion simultaneously. The main contributions of this work are summarized as follows:

- We propose a 2D via-aware spine tree generation technique and a fast maze routing algorithm to generate blockage-free

* Corresponding author.

This work was partially supported by the Natural Science Foundation of Fujian Province under Grant 2024J01363, the Young and Middle-aged Teachers Education and Research Project of Fujian Province under Grant JZ230001, the National Natural Science Foundation of China under Grant 92373207, and the National Key Research and Development Program of China under Grant 2022YFB4400500.

routing paths with minimized wire length and via count.

- We develop a congestion-aware layer assignment algorithm that maps the 2D topology into a 3D solution, with the aim of resolving initial congestion while minimizing the count of vias.
- We design an adaptive cost function to guide the bidirectional A* search. Its key innovation is a dynamic mechanism that treats vias not as a static penalty but as resource for congestion relief by continuously adjusting their cost based on real-time congestion.
- Compared with the state-of-the-art academic tool OpenROAD, the experimental results on industrial benchmarks show that our proposed router achieves 9.5% fewer via count, 2.5% smaller wire length, and significantly reduces the total routing overflow.

The remainder of this paper is organized as follows. Section II gives the preliminaries for the global routing graph model. Section III presents our global routing algorithm and co-optimization strategy. Experimental results are reported in Section IV. Finally, Section V concludes this paper.

II. PRELIMINARIES

In global routing, the chip area is partitioned into a grid of G-cells and modeled as a 3D grid graph $G(V, E)$, where each G-cell corresponds to a vertex $v \in V$. Edges in E connect adjacent G-cells and are categorized as wire edges (on the same layer) or via edges (between adjacent layers). Each edge e is characterized by its capacity $C(e)$, representing the maximum number of available tracks, and its demand $D(e)$, the total number of tracks occupied by nets routed across it. The overflow of an edge is thus defined as follows:

$$O(e) = \max(D(e) - C(e), 0). \quad (1)$$

The traditional objective of the global routing problem is to find a routing path for each net to connect all its pins such that the total overflow is minimized. In this work, we focus on the co-optimization of congestion and via count while maintaining a comparable total wire length.

III. OUR PROPOSED ALGORITHM

The proposed global router consists of two main stages: Initial Routing and Rip-up & Rerouting, as shown in Fig 1.

For the Initial Routing stage, we first generate a 2D via-aware spine tree for each net. This tree, which is built upon a wire length minimized Steiner tree, incorporates bend costs to proactively reduce via usage in the subsequent layer assignment. Subsequently, we apply a fast, double-ended queue-based maze routing algorithm to efficiently find a blockage-free path. Finally, a congestion-aware layer assignment is performed using dynamic programming to find an optimal layer for each segment, yielding a high-quality initial solution.

Then, the Rip-up & Rerouting stage iteratively applies a 3D bidirectional A* search algorithm that employs an adaptive cost function. This cost function dynamically adjusts the penalty for using vias based on local congestion, which facilitates the co-optimization of congestion and via count while avoiding

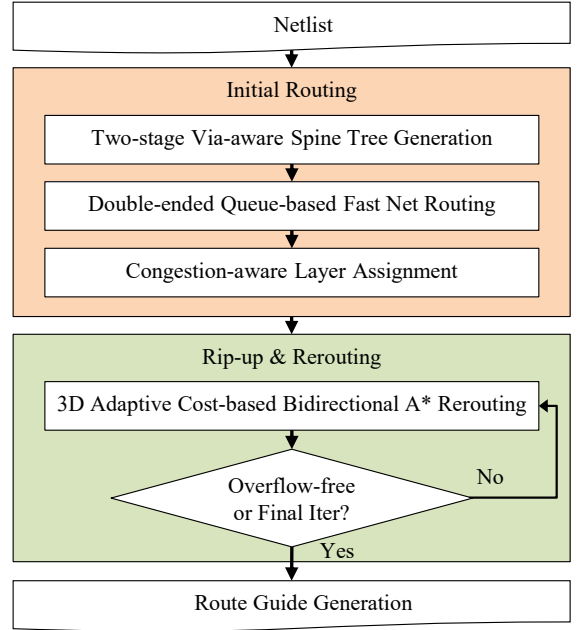


Fig. 1: Design flow of our global router.

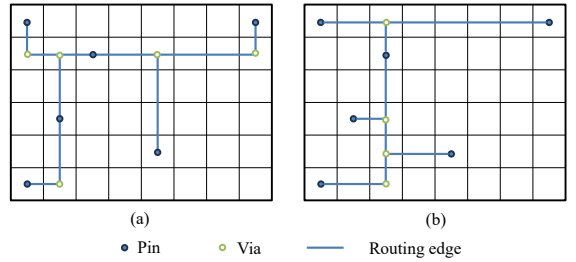


Fig. 2: Illustration of our via-aware spine tree generation. (a) An initial Steiner minimum tree with 5 vias. (b) The generated spine tree with 4 vias.

excessive wire length increase. This process repeats until the routing solution is overflow-free or a maximum iteration count is reached, at which point the final route guides are generated.

A. Two-stage Via-aware Spine Tree Generation

The number of vias generated during global routing is a critical factor affecting the final circuit yield. However, most global routers only penalize vias within the cost function of maze routing, which is a late-stage, local optimization. The effectiveness of this approach is often constrained by the routing topology fixed in earlier stages. Therefore, we propose a topology optimization method that intervenes early in the global routing flow. Through a two-stage optimization process, our method co-optimizes two key metrics: wire length and via count.

Our method's first step is to generate a rectilinear Steiner minimum tree (RSMT) with the primary objective of minimizing total wire length. The construction algorithm is based on Ho et al. [15]. Through a bottom-up traversal, this algorithm evaluates different L-shaped routing patterns for each edge of a Minimum Spanning Tree (MST) and selects the combination

that maximizes path sharing, thus theoretically approaching the shortest possible wire length. Unlike FastRoute4.0 [9], which incorporates via-awareness directly into the Steiner tree generation, our first stage strictly focuses on wire length optimization. We argue that wire length and vias are potentially conflicting objectives. Optimizing them concurrently at the initial stage may lead to a sub-optimal solution for both. Therefore, our strategy is to first generate an RSMT that is optimized purely for wire length. As an RSMT example shown in Fig 2 (a), although this topology achieves a short wire length, it may contain numerous bends, which are the primary source of vias.

The core goal of the second stage is to minimize bends through topology reconstruction, thereby indirectly yet effectively reducing the via count. We transform the RSMT into a geometrically regular spine tree. This transformation is a top-down recursive merging process. The algorithm traverses each node of the RSMT and attempts to merge the topology of its child nodes into the spine of the parent node. A strict cost function governs the merge decision: a merge is considered legal and executed only if the estimated wire length of the new spine topology does not exceed the sum of the original wire length and a bend budget. By setting an appropriate bend cost, we can compel the algorithm to generate a spine tree topology with fewer bends, with minimal to no increase in total wire length (and in some cases, even a reduction). As depicted in Fig 2 (b), the algorithm selects either a horizontal or vertical orientation to construct a main spine running through the net, creating perpendicular ribs to connect all pins. A comparison between Fig 2 (a) (5 vias) and Fig 2 (b) (4 vias) clearly demonstrates that the spine tree transformation effectively reduces the number of vias while maintaining wire length control.

B. Double-ended Queue-based Net Routing

After obtaining the spine tree, the next key step is to convert it into an actual routing path on the 2D grid. The topological blueprint from the previous stage considers wire length and bends, but it does not account for routing blockages or the actual routing resources on the chip. Therefore, the objective of this step is to find a concrete path for the topology that fully avoids blockages, preserves the low-bend advantage of the spine tree, and is also sensitive to routing congestion.

We adopt the routing cost function from CUGR [14]. It defines the cost of each path node as the sum of its base wire length and a non-linear congestion penalty based on resource usage. As shown in the (2), wl represents the wire length cost, which is set to 1 in this work. The congestion cost is described by the product of a logistic function and a proportional function, located on the right side of the addition. Additionally, the unit overflow cost, uoc , controls the congestion rate, while slope controls the global router's sensitivity to overflow. To avoid obstacles, the capacity of any edge blocked by them is reduced.

$$Cost_w = wl + uoc \times \frac{D(u,v)}{C(u,v)} \times \frac{1}{1 + e^{slope \cdot (C(u,v) - D(u,v))}} \quad (2)$$

We use an accelerated maze routing algorithm. The key to its speed is managing search node expansion with a double-ended

queue (deque) instead of relying solely on a traditional priority queue. To guide the search, the algorithm uses a composite cost function that combines wire length, congestion, and bends. Specifically, when expanding a node, the algorithm performs different enqueue operations based on its cost: 1) High-quality moves that extend straight towards the target are pushed to the front of the deque, enabling a fast, Depth-First Search (DFS)-like exploration. 2) High-cost moves, such as those that create bends, deviate from the initial routing guide, or enter highly congested regions, are moved to a separate priority queue. This queue is only processed when the deque is empty, a strategy that effectively prunes the unproductive search space. 3) Other nodes with lower congestion costs are pushed to the back of the deque to perform a robust, Breadth-First Search (BFS)-like wavefront expansion.

C. Congestion-aware Layer Assignment

Following the 2D maze routing, the algorithm proceeds to the layer assignment stage. Although the previous steps have determined a 2D net topology, the path segments have not yet been assigned to specific metal layers. A careful layer assignment strategy is critical to the final routing quality, as a hasty decision can lead to unnecessary vias, which will increase signal delay and decrease manufacturing yield, or assign paths to already congested metal layers, creating hotspots. To address this problem, we apply a congestion-aware layer assignment algorithm based on dynamic programming.

The algorithm's framework leverages the net topology generated in the preceding steps. It first promotes the 2D topology of the spine tree into a 3D topological tree for computation. Specifically, the connectivity of this tree is derived entirely from the 2D routing path's topology. Its 3D characteristic is defined as follows: all terminal nodes representing physical pins are initially constrained by their valid metal layer ranges in the z-dimension; meanwhile, all intermediate Steiner points representing path intersections are considered z-dimension unconstrained, awaiting the algorithm to find their optimal metal layers. A cost matrix with dimensions [number of nodes \times number of layers] is initialized to store the complete solution space for the subsequent DP calculations.

The core of our layer assignment algorithm is a bottom-up dynamic programming (DP) process performed on the spine tree. To clearly explain the algorithm's model, we first define the notation used. u and v_j represent nodes in the topology tree. $C(u)$ is the set of all children of node u . l and l' are different layers selected from the set of all available metal layers, L .

The core state of the algorithm is $Cost(u, l)$, which represents the minimum total cost for the sub-tree rooted at u on layer l . The calculation of $Cost(u, l)$ involves two basic cost terms: $ViaCost_{pin}(u, l)$ is the via cost required to connect the physical pin at node u to the target layer l . $RouteCost_{2pin}((u, l) \rightarrow (v_j, l'))$ is the total cost of the optimal physical path connecting a parent node (u, l) and a child node (v_j, l') . This cost already includes all necessary wire and via costs. To accurately model the wire cost and via cost, we employ the established cost model from the CUGR [14]. The wire cost has been discussed in

Section III-B. The via cost, $Cost_v$, not only includes a fixed base value but is also directly related to the congestion on the two metal layers it connects. This design allows via decisions to be congestion-aware, which is key to achieving co-optimization. The cost is calculated as follows:

$$Cost_v(u, u') = uvc \cdot (1 + lg(u) + lg(u')), \quad (3)$$

where the logistic term is:

$$lg(u) = \frac{1}{1 + e^{slope \cdot r(u)}}. \quad (4)$$

The DP cost calculation is determined by a set of state transition equations. The process starts at the leaf nodes of the tree. Since they have no downstream branches, they serve as the starting points for the recursive computation. The cost for a leaf node u on layer l , $Cost(u, l)$, is simply defined as the cost to connect its corresponding physical pin to layer l . Its state transition equation is defined as follows:

$$Cost(u, l) = ViaCost_{pin}(u, l). \quad (5)$$

After the costs for the leaf nodes are determined, the algorithm starts to compute the costs for all non-leaf nodes. The total cost for a non-leaf node u is the sum of the costs of all its downstream branches. Its state transition equation is as follows:

$$Cost(u, l) = \sum_{v_j \in C(u)} \min_{l' \in L} \left[Cost(v_j, l') + RouteCost_{2pin}((u, l) \rightarrow (v_j, l')) \right] \quad (6)$$

The logic for this formula is that for each child v_j of the parent node u , the algorithm finds the optimal connection between v_j and u . This optimization balances two costs: 1) The pre-computed total cost of the child's sub-tree, $Cost(v_j, l')$, on various candidate layers l' . 2) The two-pin routing cost, $RouteCost_{2pin}$, required to connect the parent (u, l) to the child (v_j, l') .

After finding this minimum cost for each branch, the algorithm sums them up to obtain the total cost for the non-leaf node u . By applying these two state transition equations in a bottom-up manner across the entire spine tree, we can compute the optimal cost for the root node on every layer. Finally, a top-down traceback determines the final metal layer for each path segment, generating a complete 3D routing solution that is co-optimized for congestion and vias.

D. 3D Adaptive Cost-based A* Rerouting

After the stages of via-aware spine tree generation, obstacle-avoiding routing, and layer assignment, our algorithm has already produced an initial 3D topology for each net which balances wire length and via count. However, because these preceding stages primarily focus on the local optimality of individual nets and do not fully consider the resource competition among them, the arise of local congestion hotspots is inevitable.

To resolve these residual congestion while continuing to optimize the via count, we introduce a powerful iterative Rip-up and Reroute process aimed at co-optimization. At the core of

this process is a bidirectional A* search algorithm guided by a multi-dimensional cost function we designed. The optimization objective here is to resolve resource competition among nets from a global perspective. The workflow begins with identifying congested nets. At the start of each iteration, the algorithm scans the entire routing grid to find all the segments of the nets that pass through overflowed G-cell boundaries. These segments will have their current routing paths ripped up and added to a priority queue for rerouting.

To achieve the co-optimization of routing congestion and via count, we designed a multi-dimensional, dynamically evolving cost function, which serves as the decision-making core of our R&R engine. The total cost of a routing path P , denoted as $Cost(P)$, is composed of the costs of all wire edges and vias it traverses. Its general form can be expressed as:

$$Cost(P) = \sum_{e_w \in P} C_w(e_w) + \sum_{e_v \in P} C_v(e_v). \quad (7)$$

The routing cost of a wire edge, $C_w(e_w)$, is defined by a multiplicative model that combines the path's physical properties, historical congestion memory, and an assessment of the current congestion state:

$$C_w(e_w) = (C_{phys}(e_w) + C_{hist}(e_w)) \cdot (1 + C_{cong}(e_w) + C_{spread}(e_w)) \quad (8)$$

Here, the base physical cost $C_{phys}(e_w)$ is primarily the length of the wire segment, ensuring the algorithm maintains a fundamental tendency to find the shortest path while satisfying other constraints. The historical congestion cost $C_{hist}(e_w)$ acts as the algorithm's long-term memory and records the cumulative number of times the G-cell has been overflowed in previous iterations, effectively addressing stubborn and recurring congestion regions.

This base cost is then multiplied by a penalty factor representing the current congestion state, which consists of two parts. The first is the incremental congestion cost $C_{cong}(e_w)$, which precisely quantifies the marginal contribution of the current routing action to congestion. We calculate this incremental cost using a pre-computed look-up table, $CostLookup$, where cost increases non-linearly with overflow:

$$C_{cong}(e_w) = CostLookup(overflow_{new}) - CostLookup(overflow_{old}) \quad (9)$$

A suitable $CostLookup$ function can be modeled in an exponential form:

$$CostLookup(overflow) = P_{base} \cdot (\lambda^{overflow} - 1). \quad (10)$$

The growth factor λ , typically set between 1.5 and 2, ensures that the cost of adding a segment to an already congested region is significantly higher than adding it to an uncongested region.

The second part is the wire spread cost $C_{spread}(e_w)$, a congestion prevention mechanism. We model this cost as a piecewise linear penalty function with two different and increasing slopes, which applies different levels of penalty in stages based on the utilization of capacity.

$$C_{spread}(e_w) = \begin{cases} 0 & D(e_w) \leq T_1 \\ k_1(D(e_w) - T_1) & T_1 < D(e_w) \leq T_2 \\ k_1(T_2 - T_1) + k_2(D(e_w) - T_2) & D(e_w) > T_2 \end{cases} \quad (11)$$

We set two soft utilization thresholds, $T_1 = 0.35 \cdot C(e_w)$ and $T_2 = 0.50 \cdot C(e_w)$. When the total demand of e_w is below the first threshold T_1 , no wire spread cost is applied, which encourages the algorithm to fully utilize uncongested regions. When the demand is between T_1 and T_2 , the algorithm applies a linear penalty with a slope of k_1 , acting as a gentle push to guide nets to begin spreading out. Once the demand surpasses the second threshold T_2 , the penalty slope increases to k_2 (where $k_2 > k_1$), creating a stronger push to prevent the region from overflow.

Our modeling of the via cost, $C_v(e_v)$, is the key to the entire co-optimization framework. We believe that an intelligent algorithm should be able to make a trade-off between the cost of using a via and the benefit of escaping congestion. To this end, we define the via cost as the product of a new base cost and an adaptive discount factor $\Psi(e_v)$:

$$C_v(e_v) = C_{v,base}(e_v) \cdot \Psi(e_v). \quad (12)$$

First, we define a new base via cost $C_{v,base}(e_v)$, which incorporates the overflow penalty of the target G-cell g_{tgt} :

$$C_{v,base}(e_v) = C_{phys}(e_v) + C_{hist}(e_v) + C_{cong}(g_{tgt}). \quad (13)$$

This design follows a simple principle: entering a congested region is inherently costly, for any reason. Then, we introduce the most critical component, the adaptive discount factor $\Psi(e_v)$. The value of this factor is determined entirely by the congestion level of the source G-cell, g_{src} , and it quantifies the urgency of escaping from the currently congested metal layer of g_{src} . We model $\Psi(e_v)$ using a single, continuous S-shaped function:

$$\Psi(e_v) = 1 - D_{max}(z_{src}) \cdot \frac{1}{1 + e^{-k\left(\frac{D(g_{src})}{C(g_{src})} - r(z_{src})\right)}} \quad (14)$$

This function can smoothly and dynamically generate a discount based on the g_{src} 's congestion rate. When the source G-cell has abundant resources, $\Psi(e_v)$ is approximately 1, offering almost no discount. When the source is severely congested, $\Psi(e_v)$ approaches $1 - D_{max}$, providing a significant cost discount. D_{max} is the maximum discount factor allowed by the algorithm, which we set to 0.5 in this work. Taking a step further, the maximum discount rate $D_{max}(z_{src})$ and the congestion sensitivity threshold $r(z_{src})$ are both parameters related to the metal layer z . This means we can configure this discount behavior to be activated only in the congestion-prone lower and middle layers, while it automatically deactivates on top-level metal layers, leaving those high-level resources for clock net or critical signal nets.

Algorithm 1 details the bidirectional A* search guided by the adaptive cost function. Lines 1–7 initialize priority queues (q_s, q_t) and path costs (g_s, g_t) for the forward and backward searches, while $best_cost$ tracks the optimal path cost found so far. The main loop begins at line 8 and alternates wavefront

Algorithm 1 Bidirectional A* Search Rerouting

Input: A net $N = \{s, t\}$ to be rerouted, Grid Graph G

Output: The new path p with the lowest cost from s to t

```

1:  $q_s, q_t \leftarrow$  Priority queues for forward (from  $s$ )
2:   and backward (from  $t$ ) searches
3:  $g_s[v], g_t[v] \leftarrow \infty$  for all nodes  $v \in G$ 
4:  $parent_s[v], parent_t[v] \leftarrow \mathbf{null}$  for all nodes  $v \in G$ 
5:  $g_s[s] \leftarrow 0$ ;  $q_s.push(s, heuristic(s, t))$ 
6:  $g_t[t] \leftarrow 0$ ;  $q_t.push(t, heuristic(t, s))$ 
7:  $best\_cost \leftarrow \infty$ ;  $meeting\_node \leftarrow \mathbf{null}$ 
8: while  $q_s$  is not empty and  $q_t$  is not empty do
9:    $u \leftarrow$  node with the minimum f-cost in  $q_s$ 
10:  for each neighbor  $v$  of  $u$  do
11:     $g\_cost \leftarrow g_s[u] + cost(u, v)$ 
12:    if  $g\_cost + heuristic(v, t) \geq best\_cost$  then
13:      continue;
14:    end if
15:    if  $g\_cost < g_s[v]$  then
16:       $g_s[v] \leftarrow g\_cost$ 
17:       $parent_s[v] \leftarrow u$ 
18:       $q_s.update(v, g_s[v] + heuristic(v, t))$ 
19:      if  $v$  visited by backward search then
20:         $path\_cost \leftarrow g_s[v] + g_t[v]$ 
21:        if  $path\_cost < best\_cost$  then
22:           $best\_cost \leftarrow path\_cost$ 
23:           $meeting\_node \leftarrow v$ 
24:        end if
25:      end if
26:    end if
27:  end for
28: end while
29: return  $reconstruct\_path(s, t, meeting\_node, parent_s,$ 
     $parent_t)$ 

```

TABLE I: Statistics of the benchmarks

Benchmark	#Macros	#Cells	#Nets	#Pins
ct_top_N7	92	66795	77428	257083
HopenC910_N7	272	677558	718230	2658520
RocketSystem	332	2081549	2214354	6704609
fetch_cur_luma	40	1148746	1151466	4188483
z80_core_top	102	1266789	1278422	4813006
top	22	161383	232182	639334
ddr3	70	1059643	1162363	3295332
openC910	355	2187832	2320637	7946901

expansion from the source and target to ensure the two search frontiers advance toward the middle in a balanced manner.

Lines 10–27 detail the core logic for expanding from one direction (using the source as an example). For each neighbor v of the current node u , the algorithm first calculates the path cost g_cost of reaching v through u in line 11. The cost function used here is the multi-dimensional, dynamic cost function described

TABLE II: Comparison of Solution Quality (Overflow, Via, Wire Length) and Runtime (GR, TR) with iTool and OpenRoad

Benchmark	Overflow (%)			Via Count			Wire Length			GR Time (s)			TR Time (s)		
	Ours	iTool	OR	Ours	iTool	OR	Ours	iTool	OR	Ours	iTool	OR	Ours	iTool	OR
ct_top_N7	0.11	0.29	0.30	456939	450541	469809	1286007	1333999	1378613	52	56	49	16	18	15
HopenC910_N7	0.79	1.61	1.62	1726479	2140350	2307846	3872244	3975389	4115481	347	323	281	36	35	36
RocketSystem	0.02	1.46	1.50	4462593	4520744	4602525	11538914	11408037	11831866	2010	2122	1981	112	120	122
fetch_cur_luma	0.03	1.20	1.11	3048429	3353316	3556352	10567993	10565128	10613544	1364	1363	916	164	165	163
z80_core_top	0.01	0.29	0.31	3159871	3254659	3273954	10392536	10341627	10586541	1256	1259	1042	157	159	174
top	0.33	0.58	0.60	1160666	1185877	1229183	3559256	3300104	3331206	245	226	230	33	33	29
ddr3	0.58	2.86	3.69	4366155	4628816	4545161	12167229	12313883	13504231	1020	1008	991	165	161	186
openC910	0.35	1.33	1.36	4556560	4870468	4823089	13903498	13079034	13540448	1443	1452	1412	144	153	189
Norm.Average	0.278	1.203	1.311	1.000	1.065	1.095	1.000	0.991	1.025	1.000	0.998	0.890	1.000	1.028	1.056

earlier. The algorithm’s efficiency primarily comes from the pruning step in line 12 and the meeting point check in line 19. In the pruning step, if the sum of the new path’s known cost, g_cost , and the heuristic cost from v to the final target t , $heuristic(v, t)$, is no better than the current best complete path cost, $best_cost$, then this path is considered unpromising and is immediately pruned. This strategy significantly reduces the search space. If a path passes the pruning check, the algorithm then checks in line 18 if the backward search wavefront has already reached its destination node v . If so, this means the two search frontiers have met, forming a complete path. At this point, the algorithm calculates the total cost of this complete path (line 20) and uses it to update $best_cost$ and the $meeting_node$ (lines 21–23). After the main loop terminates, the algorithm reconstructs the lowest-cost routing path that achieves the best trade-off between congestion and vias. This is done by backtracking from the final recorded $meeting_node$ using the parent pointers from both search directions.

IV. EXPERIMENTAL RESULTS

The proposed via-aware global router was implemented in C++, and all experiments were performed on a Linux workstation with an AMD EPYC 7763 1.5 GHz CPU and 512GB of memory. To evaluate the performance of our proposed router, we conducted experiments on a series of industrial designs under the 7nm technology node. Our experimental flow strictly followed industrial flow: first, we used a leading commercial tool with the default settings (represented by iTool) to complete the design’s placement and clock tree synthesis. The resulting DEF file then served as a unified input for our algorithm, iTool, and the academic open-source tool OpenROAD. After each tool completed its global routing, all output DEF files were imported back into iTool to run track routing and then reported the final quality metrics.

Our benchmarks contain eight industrial designs as shown in Table I, where the numbers of macros, standard cells, routable nets, and pins of all routable nets are denoted by “#Macros”, “#Cells”, “#Nets”, and “#Pins”, respectively. Table II lists the comparisons of the solution quality among “Ours”, “iTool”, and “OpenROAD”. “Overflow(%)” represents the total routing overflow in the horizontal and vertical directions, and “Norm.Average” shows the average normalized via count and

wire length ratios based on our results. Since the values of overflow are relatively small, we compared the average value instead of the average ratio.

As shown in Table II, our algorithm outperforms iTool and OpenROAD significantly in overflow control, achieving near-zero overflow (0.278%) on average. On challenging test cases with severe congestion, such as HopenC910_N7 and ddr3, our algorithm almost completely eliminates the large overflow produced by the competing tools. While achieving extremely low overflow, our algorithm also shows a consistent advantage in the number of vias, while reducing via counts by 6.5% and 9.5% respectively. These results collectively validate the effectiveness of our proposed adaptive cost-based framework.

Additionally, our total wire length is comparable to iTool (only 0.9% increase) and 2.5% shorter than OpenROAD, demonstrating a favorable trade-off between congestion/via reduction and wire length. Finally, regarding runtime, although our GR runtime is higher due to the iterative strategy, the high-quality output accelerates the subsequent Track Routing (TR) stage by 2.8% and 5.6% compared to iTool and OpenROAD, respectively, benefiting the overall design cycle.

V. CONCLUSIONS

We have presented a global router that co-optimizes routing congestion and the number of vias in modern VLSI design. In the initial routing stage, we first generated a 2D via-aware spine tree for each net. Then, a fast, deque-based maze routing algorithm was employed to efficiently find a blockage-free path. This is followed by a congestion-aware dynamic programming for layer assignment, which yields a high-quality initial 3D routing solution. Based upon this initial solution, an iterative rip-up and reroute stage refined the result using a 3D bidirectional A* search algorithm. This search was guided by an adaptive cost function that dynamically adjusts the cost of using a via based on the severity of congestion. Compared with state-of-the-art tools, experimental results on challenging industrial circuits demonstrate that our router achieves significant reductions in both total overflow and the count of vias, while maintaining almost the same wire length.

REFERENCES

- [1] M.-P. Wong, W.-H. Liu, and T.-C. Wang, "Negotiation-based track assignment considering local nets," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 378–383.
- [2] G. Chen, C.-W. Pui, H. Li, and E. F. Y. 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*, vol. 39, no. 9, pp. 1902–1915, 2020.
- [3] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Y. 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)*, 2019, pp. 1–7.
- [4] P. Zhang, P. Yao, X. Li, B. Yu, and W. Zhu, "V-gr: 3d global routing with via minimization and multi-strategy rip-up and rerouting," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 963–968.
- [5] 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*, 2009, pp. 582–587.
- [6] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang, "Nthu-route 2.0: A fast and stable global router," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 338–343.
- [7] 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.
- [8] J. He, U. Agarwal, Y. Yang, R. Manohar, and K. Pingali, "Sproute 2.0: A detailed-routability-driven deterministic parallel global router with soft capacity," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 586–591.
- [9] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: Global router with efficient via minimization," in *2009 Asia and South Pacific Design Automation Conference*, 2009, pp. 576–581.
- [10] 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, 2008.
- [11] 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.
- [12] T.-H. Wu, A. Davoodi, and J. T. Linderorth, "Grip: Scalable 3d global routing using integer programming," in *2009 46th ACM/IEEE Design Automation Conference*, 2009, pp. 320–325.
- [13] Y. Xu and C. Chu, "Mgr: Multi-level global router," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 250–255.
- [14] J. Liu, C.-W. Pui, F. Wang, and E. F. Y. Young, "Cugr: Detailed-routability-driven 3d global routing with probabilistic resource model," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
- [15] J.-M. Ho, G. Vijayan, and C. Wong, "New algorithms for the rectilinear steiner tree problem," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 2, pp. 185–193, 1990.