2024 CAD Contest Problem D: Chip Level Global Router

B11901029 Jhih-Jie Lee
Department of Electrical Engineering of National Taiwan University

B10901192 Chun-Che Lin
Department of Electrical Engineering of National Taiwan University

B09901137 Tsz Wun Fok
Department of Electrical Engineering of National Taiwan University

***Abstract*—In this paper, we present a solution to the ICCAD 2024 CAD Contest Problem D: Chip Level Global Router. As integrated circuits become increasingly complex, effective global routing at the chip level is essential for managing communication between various blocks. Our approach incorporates innovative algorithms designed to efficiently route connections, aiming to minimize congestion and reduce detours. Experimental results on provided test cases demonstrate the effectiveness of our method in achieving significant improvements in routing quality and performance.**

I. Introduction

The advancement of technology has led to circuits containing billions of transistors. To manage this complexity, modern physical design typically adopts a hierarchical design approach. Once floorplanning is completed, planning the routing area at the chip level becomes crucial. The global router must be well-designed to minimize detours and avoid congestion.

In ICCAD 2024 CAD Contest Problem D [1], a multi-objective global router is required to improve congestion, via counts, and wirelength while providing an obstacle-avoiding path. In previous works, GRIP [2] applies linear programming to minimize wirelength and via costs simultaneously but consumes excessive runtime. Meanwhile, FLUTE [3] breaks multi-pin nets into sets of two-pin nets to provide an initial solution at high speed. Maize-Router [4] applies dynamic cost deflation and interdependent net decomposition to achieve higher quality.However, these methods often focus on a single aspect and lack a more comprehensive consideration.

This report addresses these challenges by proposing a lightweight global router based on an efficient obstacle-avoiding Steiner tree constructor proposed by Lin [5]. Our solution focuses on minimizing wire length while effectively managing the irregular shapes and densities. Through our approach, we aim to enhance the overall efficiency and effectiveness of the routing process in high-complexity chip designs.

II. Problem Formulation

In ICCAD 2024 CAD Contest Problem D [1], the proposed global router needs to satisfy several constraints, including connecting specified pins, accommodating blocks that nets must feed through, and avoiding non-feedthroughable blocks. The quality of the global router is evaluated based on the following key cost factors:

1) Channel Overflow: The ratio of occupied routing tracks to available routing tracks, crucial for avoiding DRC violations.
2) Wirelength: The total length of all segments in a net. Shorter lengths uses fewer resources and reduces timing issues.
3) Edge Pin Density: Density of pins on a block edge, which must be controlled to prevent DRC violations and timing delays.
4) Turn Cost: The number of turns a net makes. Each turn increases routing delay and the likelihood of timing violations.

Minimizing these costs is essential to ensure that the chip meets specifications and can be successfully manufactured.

III. The First Stage Algorithm

A. Blocks Removal and Partitioning

We aim to adapt the algorithm from [5] to solve Problem D, which involves some differences in the problem formulation. First, in [5], pins must lie outside all obstacles, whereas in Problem D, pins are located within blocks. Therefore, when routing each net, we must remove all blocks containing the net's pins. Additionally, we remove all feedthroughable blocks since they do not obstruct any nets. In [5], obstacles are rectangular and non-overlapping. In Problem D, however, obstacles are rectilinear polygons which can overlap. To accommodate the requirement of the algorithm for rectangular blocks, the rectilinear polygon blocks are partitioned into rectangles. Edges between adjacent rectangles are preserved for later deletion, as they lie within the rectilinear polygons and should be removed during the subsequent graph construction.

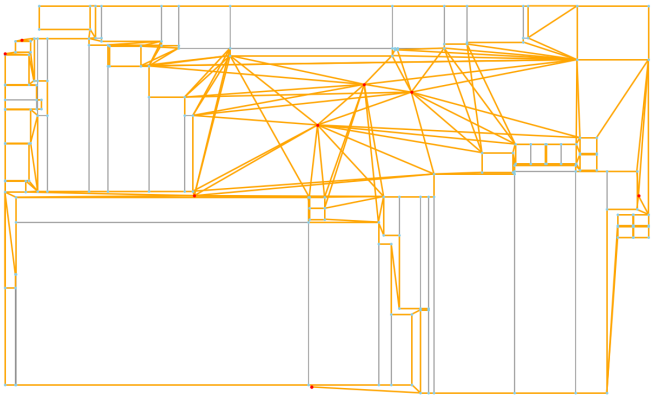
The partitioning algorithm proceeded as follows: process the vertices by increasing x-coordinates. For vertices with the same x-coordinate, process them by increasing y-coordinate. When processing a vertex, let the coordinates of this vertex be (x_1, y_1). This vertex serves as the bottom-left corner of a rectangle. Identify the next vertex with coordinates (x_2, y_1), where $x_2 > x_1$, to serve as the bottom-right corner. Identify the next vertex with coordinates (x_2, y_2), where $y_2 > y_1$, to serve as the top-right corner. The last, top-left corner of the rectangle is (x_1, y_2), which must exist since all blocks are rectilinear polygons. Repeat this process until all vertices have been processed.

B. Routing tree construction

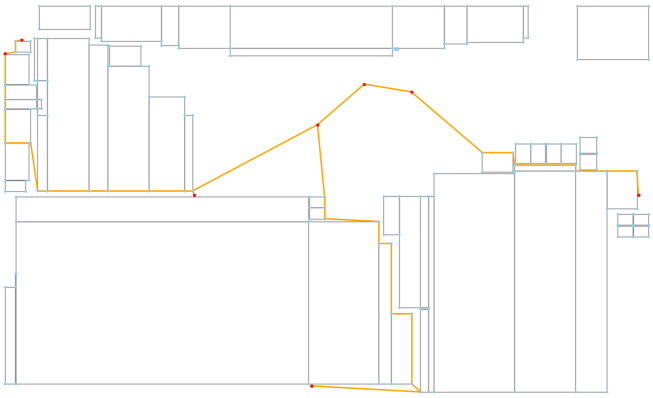
Our procedure for constructing the rectilinear Steiner routing tree basically follows the steps outlined in [5]:

1) Obstacle-Avoiding Spanning Graph (OASG): an undirected graph that connects all pins and all 4 vertices of the rectangular obstacles, with no edge intersecting with any obstacles. This step allows us to disregard the obstacles without violating the obstacle-avoiding property. A slight modification involves removing edges that lie within the original rectilinear polygons to preserve this property.
2) Obstacle-Avoiding Spanning Tree (OAST):
   The OAST is built by selecting edges from the OASG, using the following steps:
   a) **Shortest path Computation**: the shortest Manhattan distance path between each pair of pins is computed using Dijkstra's algorithm.
   b) **Initial OAST Construction**: A minimum spanning tree of the shortest paths is obtained using Prim's algorithm, which is then mapped back to the paths of the OASG.
   c) **Local Refinement**: Additional edges from the OASG are added to the OAST, and unnecessary cycles are removed to reduce wirelength.
3) Obstacle-Avoiding Rectilinear Spanning Tree (OARST) Heuristics are applied to convert slant edges to vertical ones by merging adjacent edges while reducing wirelength in [5]. However, we simply process slant edges by adding a single Steiner point to the less congested side due to overflow problems.
4) Obstacle-Avoiding Rectilinear Steiner Tree (OARSMT): remove overlapping edges and redundant vertices from OARST. In [5], U-shape pattern refinement is used to further reduce wavelength. However, we have chosen not to apply this technique here due to concerns that it may exacerbate congestion.

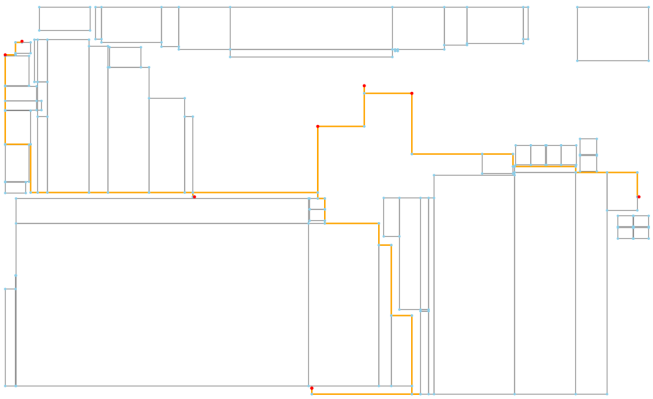
Fig. 1a to 1c shows our routing result for Case4, net 314, in the order of OASG, OAST, and OARST, respectively. (OARSMT looks the same as OARST in graph.) Gray lines represent block edges after partitioning, red spots indicate net pins, and orange lines depict graph edges.



(a) OASG



(b) OAST



(c) OARST

Fig. 1: Construction result for Case4, net 314.

IV. Modification for Congestion Optimization

A. Congestion-Aware Edge Weight

We monitor the density by adjusting the routing cost based on a continually updated congestion map. By applying a linearly increase edge weight function $w(e) = \alpha \times \text{overflow} + \text{length}$ when constructing OAST & OARST, both wire length and congestion are considered.

The reason we choose a linear weight function over an abrupt step function that blocks routes through dense areas is to optimize resource utilization throughout the process. With a abrupt step weight function, early routes utilize resources without considering later demands. Therefore, later routes will be forced to detour when encountering congested regions. A linear weight function ensures a more balanced distribution of routing resources.

B. Additional Vertices Insertion

The first stage algorithm struggles with congestion due to limitations in edge selection: in the OASG construction, it can only choose edges along the sides of obstacles and between obstacle vertices. This limitation is not significant if the obstacles fill the chip compactly without wide spaces. However, in both cases, removing all feedthroughable blocks

creates a large empty space in the middle of the chip, as shown in Fig. 2.

To address this, we place a sequence of continuation points at intervals of gcell width, forming an "X" shape in the empty spaces to distribute congestion evenly across each gcell. The diagonal line from the lower-left to the upper-right provides edges from the upper-left to the lower-right vertices, while the other diagonal line from the upper-left to the lower-right provides connections in the opposite direction. We can generalize this method for future cases by detecting wide empty spaces and directly adding a sequence of "X"-shaped vertices.

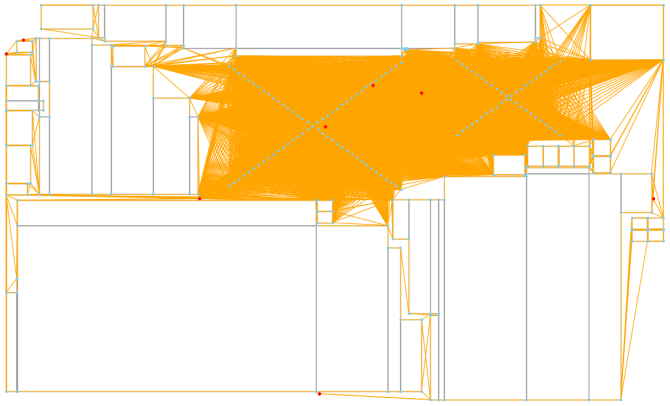


Fig. 2: OASG after placing diagonal vertices

C. No bounding box candidate removal in OASG

At the end of the OASG construction, we originally removed some candidate vertices in the candidate vertex set A to ensure they satisfy the neighbor definition in [5]: "a vertex $f \in A$ is a neighbor of vertex $v \in A$ if no other vertices $\in A$ or obstacles is inside or on the boundary of the bounding box of v and f." Since the aim of [5] is only optimizing wirelength, this steps reduce the number of vertices and edges to consider at no cost. However, if we want to account for congestion, this step reduce the number of edge choice when constructing OAST and may lead to higher overflow, especially after we add additional vertices. Therefore, we decided not to remove any candidates by the bounding box according to the neighbor definition.

D. Rip-Up and Reroute (RRR)

Net order can significantly affect the routing pattern. Therefore, after all nets are routed, we check the congestion map for congested regions, i.e. gcell with congestion $>$ `rrr_threshold`. Nets that pass through these regions are "ripped-up" and rerouted. Among these nets, We randomly reroute a number (`rrr_num`) of them and see if it yields better results with less overflow. Our experiments show that this generally leads to improved outcomes.

V. Experimental Results

We implemented our router using C++, and conducted experiments on the provided test case, Case4 (set tracks/μm $= 20$), to evaluate how our modifications improve overflow. All of the programs run within 2 minute. We compare the following 6 stages of modification:

1) Plain OARSMT method
2) (1) + Congestion-aware edge weight ($\alpha = 35$)
3) (2) + Additional vertices insertion
4) (3) + No bounding box candidate removal OASG
5) (4) + Reroute all nets through congested region once
6) (4) + Reroute all nets through congested region twice (`rrr_num` = number of nets)

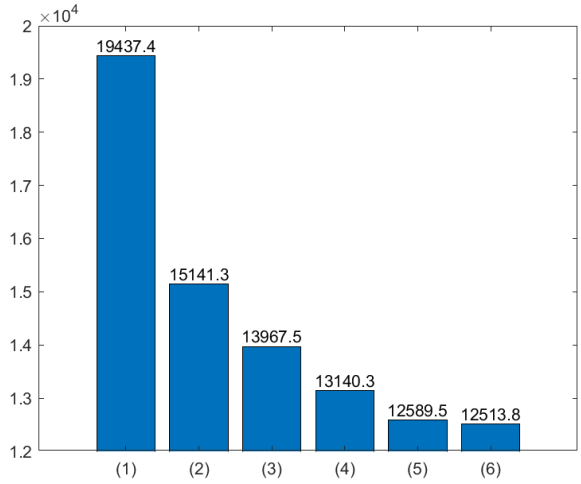


Fig. 3: Comparison of different stage of modification

Fig. 3 shows the comparison in terms of $\text{cost}_{\text{overflowlength}}$. Since the overflow at each stage is lower than at the previous stage, we can conclude that each of our modifications indeed decreases congestion. The original OARSMT is the worst, with a cost of 19437.4. Simply adapting a congestion-aware edge weight function significantly reduces congestion. Compared to the original OARSMT method, using congestion-aware edge wieght function reduces the cost by 22% to 15141.3.

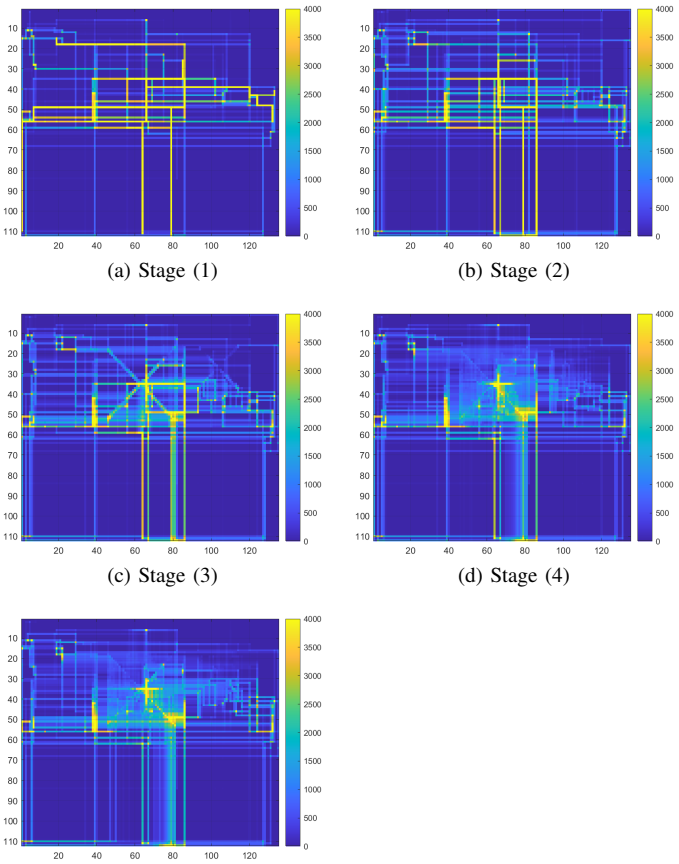
Fig. 4a to 4e show the congestion maps for stages (1), (2), (3), (4), and (5). The yellow parts represent the most congested regions, while the deep blue parts indicate the sparsest regions (usually inside the non-feedthroughable blocks). In the original OARSMT construction, although it correctly constructs a feasible routing solution, the nets all use similar regions on the chip, resulting in high congestion. After several stages of modification, only a few regions remain congested (usually where the nets' pins are located). Most regions on the chip exhibit only slight overflow.

VI. Future Works

A. Via Minimization

In Problem D, the score formula actually contains

$$\text{penalty}_{\,\#\,\text{netTurn}} = \sum_{\text{net}} \left(e^{\#\,\text{netTurn}} \quad \text{if \# of netTurn} > 1\right)$$



(a) Stage (1) (b) Stage (2)

(c) Stage (3) (d) Stage (4)

(e) Stage (5)

Fig. 4: Congestion map of Stage (1) to Stage (5)

which grows exponentially. This poses a significant challenge when dealing with many pins and complex routing trees, as seen with net 1538 in Case4, which has 35 distinct pins. If net 1538 requires ≥ 15 vias, as is likely, its turning cost would exceed $0.01 \cdot 3.269 \times 10^9 = 32690$. In Case4, our worst-case scenario for total $(0.55 \cdot \text{cost}_{\text{overflowlength}})$ is approximately 1×10^4. Thus, the turning cost of a single net can surpass the cumulative effect of overflow, not to mention the total turning cost for all 2307 nets in Case4. While we are uncertain if this discrepancy is an oversight by the contest organizers, addressing this issue is crucial for further progress in the competition.

B. Must Through Constraint

Until June 27th, 2024, the two available public test cases, Case4 and Case5, do not impose any restrictions on must through blocks or must through block edges, thus these situations are not currently addressed. We plan to deal with mush through constraints by adding pseudo pins on the block edges, as show in Fig. 5. Once the starting pin TX and the ending pin RX of the net have connection with the pseudo pins on the block edges, we can directly connect the 2 pseudo pins, then the net must exactly feedthrough the block.

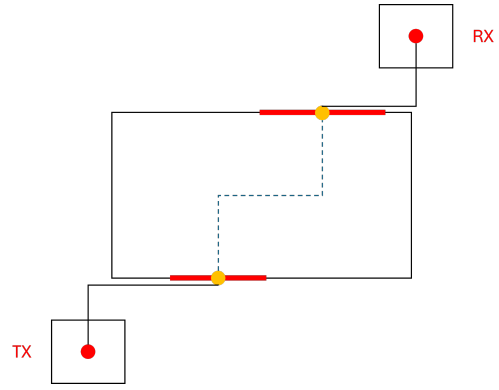


Fig. 5: Must through method (yellow points are pseudo pins)

C. Feedthrough Limit

The problem description mentions that exceeding the feedthrough limit of block `through_block_net_num` incurs penalties, but it does not provide a quantified formula for this. Therefore, we decided to remove all feedthroughable blocks without considering the limit. To address this, we may consider placing a block back after the total tracks feedthroughing this block have exceeded the limit by a specific factor. This decision involves a trade-off between this penalty and other costs such as overflow and turning costs, so the factor may be determined once the score function is provided.

VII. Conclusion

In this report, we presented our approach to the ICCAD 2024 CAD Contest Problem D, focusing on efficient obstacle-avoiding Steiner tree construction to minimize wire length and manage irregular chip densities.

Our multi-stage modifications, including congestion-aware edge weights, additional vertex insertion, and rip-up and reroute mechanisms, significantly improved congestion management and routing efficiency. Experimental results showed a marked reduction in both wire length and channel overflow.

Future work will explore further optimizations, such as advanced via minimization and better handling of must-through constraints. The methodologies developed in this study provide valuable insights for enhancing global routing algorithms and improving the design of high-complexity integrated circuits.

References

[1] CAD Contest at ICCAD, [Online]. Available: http://iccad-contest.org/
[2] T. -H. Wu, A. Davoodi and J. T. Linderoth, "GRIP: Scalable 3D global routing using Integer Programming," 2009 46th ACM/IEEE Design Automation Conference, San Francisco, CA, USA, 2009, pp. 320-325.
[3] Chu, C.; Wong, Y.C. Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2007, 27, 70–83.
[4] M. D. Moffitt, "MaizeRouter: Engineering an Effective Global Router," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 11, pp. 2017-2026, Nov. 2008,
[5] Lin, Chung-Wei, et al. "Efficient obstacle-avoiding rectilinear Steiner tree construction." Proceedings of the 2007 international symposium on Physical design. 2007.

JOB DIVISION

- B11901029 Jhih-Jie Lee:
  - **Code**: OASG, OARSMT, congestion map, reroute function, *.json parsers, *.rpt writer.
  - **Report**: First Stage Algorithm (OASG & OARSMT), Modification (no bounding box, additional vertices insertion, rip-up and reroute), Experiment Result, Future Works
- B10901192 Chun-Che Lin:
  - **Code**: OAST, OARST, Result statistics & visualization
  - **Report**: Abstract, Introduction, Problem formulation, First Stage Algorithm (OAST & OARST), Modification (Congestion-Aware Edge Weight)
- B09901137 Tsz Wun Fok:
  - **Code**: *.def parser, block positioning, blocks removal and partitioning
  - **Report**: First Stage Algorithm (blocks removal and partitioning)