Appendix: Supplementary Materials for `DieRouter+`

I. Analysis of the Heuristic Method for Solving Continuous TDM Ratios in DieRouter

A. *Heuristic Method in* `DieRouter`

`DieRouter` employs the following iterative procedure to solve for the continuous TDM ratios:

$$\bar{r}_{i,j}^{(t+1)} = r_{i,j}^{(t)} \cdot \frac{d_s^{(t)}}{c} \cdot \frac{1}{d_i^{(t)}}, \forall i \in \mathcal{D}_j^{\mathcal{T}} \tag{1}$$

$$r_{i,j}^{(t+1)} = \frac{\sum_{k \in \mathcal{D}_j^{\mathcal{T}}} \frac{1}{\bar{r}_{k,j}^{(t+1)}}}{n_j} \cdot \bar{r}_{i,j}^{(t+1)}. \tag{2}$$

In this formulation, the set $\mathcal{D}_j^{\mathcal{T}} = \{i \mid i \text{ traverses } j\}$ represents the nets routed through j. $r_{i,j}^{(t)}$ represents the continuous TDM ratio at the t-th iteration. $d_s^{(t)}$ denotes the continuous maximum net delay computed from $r_{i,j}^{(t)}$, while $d_i^{(t)}$ represents the continuous delay of net i, also derived from $r_{i,j}^{(t)}$. The term $\frac{d_s^{(t)}}{c}$ defines the compression target, where c is a hyperparameter that regulates the target threshold.

The intuition behind Equation (1) is as follows: if $d_i^{(t)}$ exceeds the compression target, the ratio $r_{i,j}^{(t)}$ is reduced to decrease $d_i^{(t)}$. Conversely, if $d_i^{(t)}$ falls below the target, $r_{i,j}^{(t)}$ can be increased to redistribute resources to more critical nets. After updating $r_{i,j}^{(t)}$ to $\bar{r}_{i,j}^{(t+1)}$, the normalization step in Equation (2) ensures that $r_{i,j}^{(t+1)}$ satisfies the resource constraint.

The iteration process terminates when either the number of iterations exceeds a preset threshold or the change in continuous maximum net delay between consecutive iterations falls below a given threshold ϵ.

B. *Analysis of the Heuristic Method in* `DieRouter`

To analyze the above update formula, we substitute (1) into (2) and simplify the formula as follows:

$$r_{i,j}^{(t+1)} = \frac{1}{n_j} \cdot \frac{r_{i,j}^{(t)}}{d_i^{(t)}} \cdot \sum_{k \in \mathcal{D}_j^{\mathcal{T}}} \frac{d_k^{(t)}}{r_{k,j}^{(t)}} \tag{3}$$

The simplified expression reveals that the iteration process is inherently independent of c.

Lemma 1. *Let i be one of the most critical nets, i.e., $d_i^{(t)} \geq d_k^{(t)}$ for all k. Define the set of TDMed edges traversed by i as $\mathcal{C}_i^{\mathcal{T}} = \{j \mid j \text{ is a TDMed edge traversed by } i\}$, and assume $\mathcal{C}_i^{\mathcal{T}} \neq \emptyset$. Then, the continuous delay of i satisfies $d_i^{(t+1)} \leq d_i^{(t)}$, with equality if and only if all nets in $\mathcal{D}_j^{\mathcal{T}}$ are equally critical for all $j \in \mathcal{C}_i^{\mathcal{T}}$.*

Proof. For each TDMed edge $j \in \mathcal{C}_i^{\mathcal{T}}$, we show that $r_{i,j}^{(t+1)} \leq r_{i,j}^{(t)}$ as follows:

$$\begin{aligned} r_{i,j}^{(t+1)} &= \frac{1}{n_j} \cdot \frac{r_{i,j}^{(t)}}{d_i^{(t)}} \cdot \sum_{k \in \mathcal{D}_j^{\mathcal{T}}} \frac{d_k^{(t)}}{r_{k,j}^{(t)}} && (4) \\ &= \frac{1}{n_j} \cdot r_{i,j}^{(t)} \cdot \sum_{k \in \mathcal{D}_j^{\mathcal{T}}} \frac{d_k^{(t)}}{d_i^{(t)}} \cdot \frac{1}{r_{k,j}^{(t)}} && (5) \\ &\leq \frac{1}{n_j} \cdot r_{i,j}^{(t)} \cdot \sum_{k \in \mathcal{D}_j^{\mathcal{T}}} \frac{1}{r_{k,j}^{(t)}} && (6) \\ &= \frac{1}{n_j} \cdot r_{i,j}^{(t)} \cdot n_j = r_{i,j}^{(t)}, && (7) \end{aligned}$$

where the second-to-last equality follows from the identity $\sum_{k \in \mathcal{D}_j^{\mathcal{T}}} \frac{1}{r_{k,j}^{(t)}} = n_j$.

Since the continuous delays of net i on all TDMed edges in $\mathcal{C}_i^{\mathcal{T}}$ do not increase, it follows that $d_i^{(t+1)} \leq d_i^{(t)}$. Furthermore, if there exists a TDMed edge $j \in \mathcal{C}_i^{\mathcal{T}}$ and a net $k \in \mathcal{D}_j^{\mathcal{T}}$ such that $d_k^{(t)} < d_i^{(t)}$, then

$$r_{i,j}^{(t+1)} < r_{i,j}^{(t)},$$

which implies $d_i^{(t+1)} < d_i^{(t)}$. This completes the proof. □

However, Lemma 1 does not necessarily imply that $d_s^{(t+1)} \leq d_s^{(t)}$. This is because the continuous delay of a non-critical net i may increase, i.e., $d_i^{(t+1)} > d_i^{(t)}$, which, in turn, may contribute to an increase in the continuous maximum net delay. Consequently, the iterative heuristic method does not guarantee a monotonic decrease in the continuous maximum net delay or convergence. As illustrated in Fig. 1, when $\epsilon = 0.05$, the heuristic method exhibits oscillatory behavior and diverges on test case 5.

II. Theoretical Results on Scheduler-Driven DP

For convenience, we summarize the key notations used in the main paper.

- N_j^+: The number of nets traversing edge j in the positive direction.
- $\{0, \ldots, N_j^+ - 1\}$: The index set of nets traversing edge j in the positive direction, ordered such that their continuous TDM ratios satisfy

$$r_{0,j} \leq r_{1,j} \leq \cdots \leq r_{N_j^+-1,j}.$$

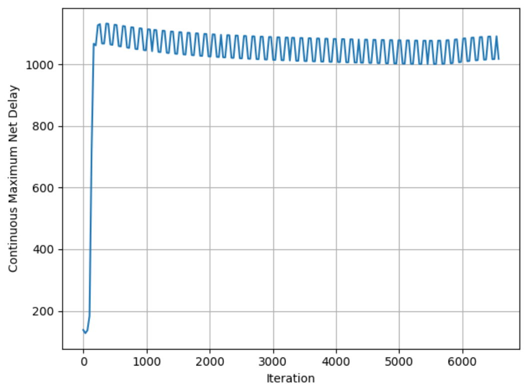

Fig. 1. Evolution of the continuous maximum net delay over iterations in Test Case 5.

- (α, β): A state representing the multiplexing of nets $\{0, \ldots, \alpha\}$ to β wires.
- $J^+(\alpha, \beta)$: The optimal state cost of (α, β), defined as the smallest maximum excess difference among all feasible TDM assignments, where each assignment multiplexes nets $0, \ldots, \alpha$ on β wires.
- $J^-(\alpha, \beta)$: The optimal state cost in the negative direction, defined analogously to the positive direction.
- $V^+(\beta)$: The optimal cost of assigning all positive-direction nets to β wires, given by
$$V^+(\beta) = J^+(N_j^+ - 1, \beta).$$
- $V^-(\beta)$: The optimal cost of assigning all negative-direction nets to β wires, given by
$$V^-(\beta) = J^-(N_j^- - 1, \beta).$$
- $g(\beta)$: The function
$$g(\beta) = \max\{V^+(\beta), V^-(n_j - \beta)\},$$
which represents the maximum cost between the two directions for a given partitioning of wires.
- β^*: The optimal β, determined by
$$\beta^* = \arg \min_{\beta \in \{1, \ldots, n_j - 1\}} g(\beta). \tag{8}$$

Lemma 2. *Let β^+ and β^- be integers satisfying $\beta^+ + \beta^- < n_j$. Then the following statements hold: (a) If $V^+(\beta^+) \geq V^-(\beta^-)$, then $g(\beta^+ + 1) \leq g(\beta^+)$; (b) If $V^+(\beta^+) \leq V^-(\beta^-)$, then $g(n_j - \beta^- - 1) \leq g(n_j - \beta^-)$.*

Proof. We provide a proof for part (a) of Lemma 2. The proof of part (b) follows a similar argument.

Given $\beta^+ + \beta^- < n_j$, it follows that $\beta^- \leq n_j - 1 - \beta^+$. Consequently, we have:
$$V^-(n_j - \beta^+ - 1) \leq V^-(\beta^-) \leq V^+(\beta^+), \tag{9}$$
where the first inequality holds since V^- is a non-increasing function.

Furthermore, since $V^+(\beta^+ + 1) \leq V^+(\beta^+)$, we can establish the following:
$$\begin{aligned} g(\beta^+ + 1) &= \max\{V^+(\beta^+ + 1), V^-(n_j - \beta^+ - 1)\} &(10)\\ &\leq V^+(\beta^+) &(11)\\ &\leq \max\{V^+(\beta^+), V^-(n_j - \beta^+)\} &(12)\\ &= g(\beta^+). &(13)\end{aligned}$$

This completes the proof of part (a). □

Theorem 1. *Once the scheduler has allocated all wires, that is, when $\beta^+ + \beta^- = n_j$, β^+ is an optimal solution to* (8).

Proof. We first consider the case $n_j = 2$. In this scenario, the scheduler terminates with $\beta^+ = \beta^- = 1$. Since each direction requires at least one resource, it is clear that $\beta^+ = 1$ is an optimal solution.

Next, we examine the case $n_j > 2$. The crux of the argument relies on the fact that there always exists an optimal solution β^* of (8) in the set $\{\beta^+, \ldots, n_j - \beta^-\}$, and this statement serves as a loop invariant.

This property holds trivially when $\beta^+ = \beta^- = 1$. Suppose it holds at some β^+, β^-. We show that it remains true after a round of resource allocation. Note that a new resource can only be allocated if $\beta^+ + \beta^- < n_j$. We consider two cases:
Case 1. $V^+(\beta^+) \geq V^-(\beta^-)$. By Lemma 2(a), we have $g(\beta^+ + 1) \leq g(\beta^+)$. Hence, there exists an optimal solution within $\{\beta^+ + 1, \ldots, n_j - \beta^-\}$.
Case 2. $V^+(\beta^+) < V^-(\beta^-)$. By Lemma 2(b), we obtain $g(n_j - \beta^- - 1) \leq g(n_j - \beta^-)$, implying that there is an optimal solution in $\{\beta^+, \ldots, n_j - \beta^- - 1\}$.

These two cases justify the scheduler's decision to allocate a resource to the direction with a higher cost while preserving the loop invariant.

Finally, when $\beta^+ + \beta^- = n_j$, the search space containing an optimal solution reduces to a single element. That element must be optimal. □

III. Ablation Study

We present the ablation study in Table I, evaluating `DieRouter+` with different initial routing algorithms. The table reports the approximate maximum net delay at intermediate steps and the exact value at the final step. For clarity, the table uses the following abbreviations: *Init* for Initial Routing, *R&R* for violation-driven and performance-driven rip-up-and-rerouting, *SOCP* for solving continuous TDM ratios via SOCP, and *DP* for legalization via scheduler-driven DP. In the following, we refer to maximum net delay simply as delay.

We compare our SPT-based initial routing with the hybrid approach in `DieRouter`, denoted as MST-SPT, where nets are classified as critical or non-critical, with critical nets routed by MSTs and non-critical nets by SPTs. We also test an approximated SMT-based approach [1] for routing critical nets, denoted as SMT-SPT. For net classification, we use the same hyperparameters as `DieRouter`.

From Table I, we conclude that SPT-based initial routing reduces average delay by **11.42%/16.23%** over MST-SPT and

TABLE I
ABLATION STUDY RESULTS.

Test Case	**DieRouter+ w/ MST-SPT**				**DieRouter+ w/ SMT-SPT**				**DieRouter+ w/ SPT**			
	Init	R& R	SOCP	DP	Init	R& R	SOCP	DP	Init	R& R	SOCP	DP
1	2.51	2.51	6.5	**6.5**	2.51	2.51	6.5	**6.5**	2.51	2.51	6.5	**6.5**
2	6.93	5.65	13	13	6.64	6.64	14	14	3.84	3.83	7.5	**7.5**
3	12.1	12.1	10.5	13.5	17.2	16.6	13	13.5	10.7	10.7	9.5	**11.5**
4	29.7	18.1	14.4	**18.5**	30	18.1	14.32	**18.5**	16.7	16.5	15.49	**18.5**
5	139.71	138.48	125.77	**133**	150.1	137.57	125.8	**133**	137.69	137.51	125.77	**133**
$\Delta\text{Delay}_{\text{avg}}$	-20.28%	-10.66%	-8.85%	**-11.42%**	-26.51%	-17.35%	-13.04%	**-12.25%**	0%	0%	0%	0%
6	279.75	621.78	255.85	264	279.87	548.19	189.27	199.5	92.27	460.99	181.05	**187.5**
7	233.39	223.99	75.43	81.50	237.64	188.39	66.82	**73.5**	106.69	98.21	66.58	**73.5**
8	360.03	428.81	159.92	169.50	361.81	310.49	97.57	109	158.03	142.72	99.47	**105.5**
9	216.01	410.19	139.23	148.50	217.48	458.92	134.31	142.50	89.96	238.94	132.92	**139.5**
10	1699.65	7274.17	4402.39	**4420**	1699.76	7323.2	4432.11	4449	1404.81	7354.09	4465.99	4485
$\Delta\text{Delay}_{\text{avg}}$	-50.62%	-37.88%	-16.37%	**-16.23%**	-50.89%	-33.06%	-0.61%	**-2.10%**	0%	0%	0%	0%

TABLE II
PEAK MEMORY (MB) FOR EACH STAGE ON 10 TESTCASES. NUMBERS OUTSIDE (INSIDE) THE PARENTHESES CORRESPOND TO DIEROUTER+ (DIEROUTER).

Test Case	**Init SPT (Hybrid)**	**R&R**	**Conti SOCP (Heuristic)**	**Leg Scheduler-driven (Origin) DP**
1	36.48 (36.95)	37.11 (37.42)	96.89 (70.78)	72.05 (72.10)
2	37.12 (36.92)	37.77 (37.92)	100.21 (71.31)	72.45 (72.78)
3	36.60 (36.74)	37.95 (38.16)	100.43 (71.07)	72.79 (72.68)
4	37.39 (37.21)	40.39 (40.14)	106.81 (73.23)	75.38 (552.68)
5	42.73 (42.76)	82.17 (83.96)	220.27 (110.46)	302.83 (727.83)
6	263.41 (269.48)	1,325.40 (1,425.38)	9,081.75 (1,621.41)	11,580.00 (13,206.20)
7	146.72 (146.77)	722.05 (766.16)	5,421.72 (836.51)	5,410.48 (5,584.68)
8	164.09 (159.45)	828.49 (944.10)	6,163.20 (977.06)	6,086.88 (6,498.38)
9	1,396.23 (1423.98)	8,938.59 (9,112.63)	79,925.20 (11,211.20)	105,484.00 (158,935.00)
10	5,312.52 (5,426.90)	41,747.20 (41,978.70)	287,766.00 (51,848.70)	489,885.00 (592,228.00)

12.25%/2.1% over SMT-SPT on easier/ challenging test cases, highlighting the impact of the initial routing topology.

Generally, a lower approximated delay after initial routing (approximated initial delay) correlates with a lower final delay. However, when approximated initial delays are similar, final performance becomes less predictable. In test case 10, for instance, despite a slightly lower approximated initial delay with SPT-based routing, its final delay exceeds that of the other two methods. Nonetheless, approximately delay after the rip-up and rerouting phase remains a useful indicator of final performance.

IV. MEMORY CONSUMPTION ANALYSIS

Table II presents the peak memory consumption across all test cases and stages for both `DieRouter` and `DieRouter+`. Memory usage is measured using the psrecord tool, with a sampling interval of 0.01s for test cases 1–5 and 0.5s for test cases 6–10.[1] We focus our analysis on the Conti and Leg stages, as the memory usage in the Init and R&R stages is comparable between `DieRouter` and `DieRouter+`, and significantly lower than that in the later stages. We have the following observations:

1) For all test cases, the scheduler-driven DP used in `DieRouter+` consumes less memory than the original DP in `DieRouter`. This improvement arises because the scheduler-driven DP only requires maintaining a cost vector of size N_j^+ and N_j^- for the positive and negative directions of each TDMed edge j, respectively. In contrast, the original DP maintains a cost matrix of size $N_j^+ \times n_j$ and $N_j^- \times n_j$, which significantly increases memory requirements. As the problem scale grows, this difference becomes more pronounced. For

[1]A finer sampling interval was used for test cases 1–5 due to their execution times being shorter than one second.

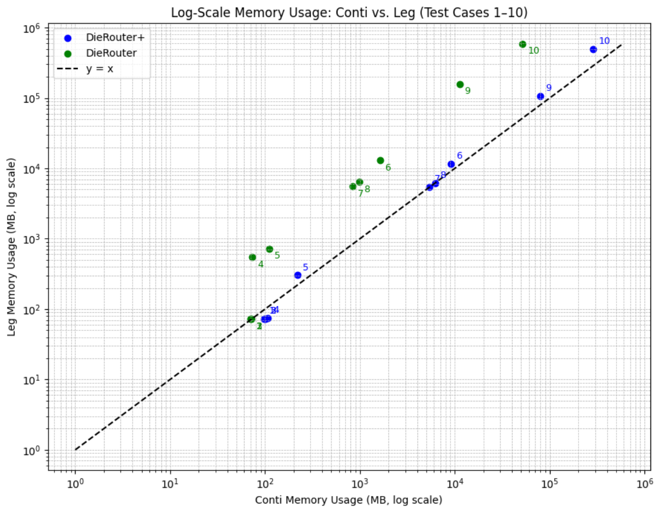

Fig. 2. `DieRouter` vs. `DieRouter+`: Memory Usage in Conti and Leg Stages.

example, in test case 10, `DieRouter+` requires approximately 478 GB of memory, while `DieRouter` consumes nearly 578 GB, exceeding the physical memory capacity. As a result, `DieRouter` fails to complete test case 10 when attempting to perform legalization in parallel using 10 sub-processes.

2) In `DieRouter+`, solving the SOCP consumes more memory than the heuristic method used in `DieRouter`. However, once the number of nets exceeds the scale of hundreds of thousands—as in test cases 6, 9, and 10—the memory bottleneck shifts to the legalization stage. In this stage, `DieRouter+` consumes less memory than `DieRouter`, as previously analyzed and further illustrated in Fig. 2.

V. Future Work and Scalability Considerations

To scale `DieRouter+` to handle larger instances, such as those with more than 10 million nets, we outline several potential directions:

1) To reduce memory usage in solving the continuous TDM ratios, the SOCP problem can be decomposed into a sequence of smaller subproblems. Specifically, we may iteratively maximize the slack ratio on a single edge while keeping the TDM ratios on all other edges fixed. This approach is analogous to the coordinate descent method and can significantly lower the memory footprint. Moreover, by controlling the number of iterations, a trade-off between solution quality and computational efficiency can be achieved.
2) In the scheduler-driven DP, computational efficiency can be improved by leveraging GPU acceleration. Since the computation of the cost vector is inherently parallel—each element being independent—it is well-suited for GPU-based parallelization. To further reduce memory consumption, more advanced memory-sharing mechanisms can be implemented, avoiding the need to duplicate the main process's memory across all child processes.

References

[1] K. Mehlhorn, "A faster approximation algorithm for the steiner problem in graphs," *Information Processing Letters*, vol. 27, no. 3, pp. 125–128, 1988.