A High Performance Detailed Router Based on Integer Programming with Adaptive Route Guides

Zhongdong Qi, Shizhe Hu, Qi Peng, Hailong You, Chao Han, Zhangming Zhu
School of Microelectronics, Xidian University
zdqi@xidian.edu.cn

2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) | 979-8-3503-9354-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ASP-DAC58780.2024.10473934

***Abstract*—Detailed routing is a crucial and time-consuming stage for ASIC design. As the number and complexity of design rules increase, it is challenging to achieve high solution quality and fast speed at the same time in detailed routing. In this work, a high performance detailed routing algorithm named IPAG with integer programming (IP) is proposed. The IP formulation uses the selection of candidate routes as decision variables. High quality candidate routes are generated by queue-based rip-up and reroute with adaptive global route guidance. A design rule checking engine which can simultaneously process nets with multiple routes is designed, to efficiently construct penalty parameters in the IP formulation. Experimental results on ISPD 2018 detailed routing benchmark show that IPAG achieves better solution quality in shorter or comparable runtime, as compared to the state-of-the-art academic detailed router.**

I. Introduction

Detailed routing is a critical stage in VLSI physical design. The interconnections including wires and vias need to be constructed satisfying various design rules. Its quality determines key metrics of a design. In addition, as the feature size shrinks, complicated design rules and increasing design sizes make detailed routing very time-consuming. It is challenging to achieve high solution quality and short runtime simultaneously.

Facing the high complexity of the detailed routing problem, there are two main approaches to solve it, including sequential approach and concurrent approach.

Sequential approach uses iterative rip-up and reroute (R&R) to reduce the number of design rule violations (DRVs) [1]–[7]. The ordering of nets during routing is critical for the runtime and the solution quality. The issue is considerably mitigated by queue-based R&R approach [7] and adaptive ordering with reinforcement learning [8]. Besides the net ordering issue, another issue in the sequential R&R approach is that net routes ripped-up are completely discarded after rerouting. Considering that net routes typically become longer in R&R procedure to avoid design rule violations, discarding routes in early iterations may lead to sub-optimality.

The concurrent routing approach formulates detailed routing as integer linear programming (ILP), Boolean satisfiability (SAT), or satisfiability modulo theories (SMT) problems [9]–[11]. In this approach, route construction of multiple nets are performed concurrently to optimize the solution quality. A well-studied formulation is to treat the assignment of each routing grid point to a net as a decision variable, and model connectivity and design rules as constraints. It generates high quality solutions, but often requires long runtime. As a result, it is generally applied in small routing regions (e.g., 10 tracks $\times$ 10 tracks). Furthermore, as the the number and complexity of design rules increases, it is difficult to explicitly represent all the design rules in the formulation. As studied in FPGA detailed routing [12], another formulation is to use choices in possible net routes as decision variables. It is more scalable than the routing-grid-based formulation [12]. The formulation with candidate route selection is also used in global routing [13]–[16]. When providing flexible candidate route generation, prominent solution quality improvement can be achieved by using this formulation compared to sequential routing approaches [16]. In global routing and FPGA detailed routing, without complicated design rules, the constraints related to routing resources are easy to construct. For the detailed routing problem of ASIC designs, the challenges with this formulation are in high-quality candidate route generation, and in efficient construction of constraints (or parameters) related to design rule violations.

In this work, we propose an effective detailed routing algorithm for ASIC designs with a scalable integer programming (IP) formulation. It leverage fast queue-based R&R as a subroutine for candidate route generation. The IP formulation uses the selection of candidate route as decision variables, which overcome the issue of discarding routes in early R&R procedure. Effective techniques are proposed to achieve high efficiency in IP instance construction and high solution quality. The main contributions of our work are summarized as follows.

- An effective detailed routing algorithm with a scalable IP formulation is proposed. It has better solution quality and similar runtime compared to sequential routing approach.
- High quality candidate routes in the IP formulation are generated using fast queue-based R&R procedure with adaptive global route guides following.
- An efficient design rule checking engine which supports examining nets with multiple routes simultaneously is designed, to quickly construct penalty parameters in the IP formulation.
- Evaluated using ISPD 2018 benchmarks [19], the proposed detailed routing algorithm achieves better solution quality with comparable or shorter runtime than the state-of-the-art (SOTA) results.

The rest of the paper is organized as follows. Section II describes our detailed routing algorithm. Section III presents

experimental results, followed by conclusion in Section IV.

II. Method

In this section, we present our detailed routing algorithm, including the overall flow, the IP formulation, and detailed techniques.

A. Overall Flow

Due to the large scale of circuit designs and complex design rules, routing is generally tackled in global routing and detailed routing stages. With the aid of global routing results, detailed routing is carried out in local regions to reduce the problem size. The routing regions are typically adjusted (shifted or enlarged) throughout the detailed routing procedure, to progressively remove or reduce design rule check violations. We follow this way in our detailed routing algorithm, which comprises some number of iterations of local region-wise routing. The main procedure of detailed routing is illustrated in Algorithm 1. In each iteration, the entire layout is partitioned into non-overlapped rectangular regions, where local detailed routing is conducted.

Algorithm 1: Detailed Routing Main Procedure

```
1 iter ← 0;
2 while #DRV > 0 and iter < maxIter do
3     local region-wise detailed routing;
4     iter ← iter + 1;
```

B. IP Formulation

In each local region for detailed routing, we are given available routing resources along with blockages, a set of design rules (including same-net rules and different-net rules), a set of N nets $\mathcal{N} = \{n_1, n_2, ..., n_N\}$, each net n_i may have global route guides. Fig. 1 shows a simple example of detailed routing problem instance in a local region. Let $\mathcal{T}_i$ be the collection of possible routing trees (or candidate routes) of net n_i. We define the binary decision variable x_{it} that is equal to 1 if and only if net n_i is routed using the routing tree t in $\mathcal{T}_i$.

To optimize wirelength and via count with the aim of reducing design rule violations, the IP formulation is designed as follows:

$$\min_x \sum_{i=1}^{N} \sum_{t \in \mathcal{T}_i} c_{it} x_{it} + \sum_{i=1}^{N} \sum_{u \in \mathcal{T}_i} \sum_{j=i+1}^{N} \sum_{v \in \mathcal{T}_j} p_{iu,jv} x_{iu} x_{jv} \quad (1)$$

$$s.t. \quad \begin{cases} \sum_{k \in \mathcal{T}_i} x_{ik} = 1 & \forall i = 1, 2, ..., N \\ x_{ik} = \{0, 1\} & \forall i = 1, 2, ..., N, \forall k \in \mathcal{T}_i \end{cases}$$

The objective function consists of the cost of routing trees (candidate routes) and the penalty of design rule violations between different nets, which is constrained by the fact that each net can select only one routing tree. c_{it} is the cost of routing tree t of net n_i, which consists of wirelength (in both preferred and non-preferred routing directions), via count, and same-net design rule violation penalty. That is, $c_{it} = (wl_{it} + \alpha \cdot wl'_{it} + \beta \cdot via_{it} + \gamma \cdot drc_{it})$, where wl_{it} is the wirelength in preferred routing direction, wl'_{it} is the wirelength in the non-preferred direction, via_{it} is the via count, drc_{it} is the number of same-net design rule violations. α, β, and γ are weights of different components. $p_{iu,jv}$ is the penalty of different-net design rule violations between routing tree u of net n_i and routing tree v of net n_j. The penalty parameter $p_{iu,jv} = \gamma \cdot drc_{iu,jv}$, where $drc_{iu,jv}$ is the number of design rule violations triggered between the routing trees represented by x_{iu} and x_{jv}.

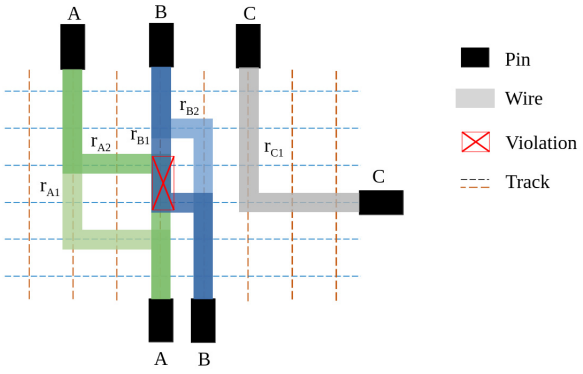


Fig. 1: A local region with three nets for detailed routing. There would be a design rule violation if routes r_{A2} and r_{B1} are selected for nets A and B respectively.

This formulation has some unique properties, compared to the routing-grid-based formulation for design routing of ASIC designs.

- Explicit representation of various design rules in the formulation is not required.
- The size of IP instances depends on the number of nets to be routed and the number of candidate routes.
- The candidate routes are not restricted to be on grid, so it is applicable for both gridded routing and gridless routing.

For the above IP formulation, the number and the quality of candidate routes significantly affect the runtime and solution quality, respectively. There can be a large number of candidate routes for each net. To control the size of IP instances, high quality candidate routes are generated by the rip-up and reroute process. We present the method of generating candidate routes in Section II-C.

To get all penalty parameters $p_{iu,jv}$ related to different-net design rule violations, an intuitive way is to check every combination of different candidate routes of all nets. The number of combinations is in an order of R^N, where N is the number of nets to be routed, and assuming that each net has R candidate routes on average. Considering the computational complexity of design rule checking for each combination of candidate routes into account, this will consume prohibitively long runtime. We show in Section II-D that by designing a design rule checking engine supporting processing nets with

multiple routes, the number of checking can be effectively reduced.

The detailed routing procedure in each local region include two phases, namely IP instance construction and IP solving to select best routes. IP instance construction comprises candidate routes generation, and design rule checking for penalty parameters generation. In candidate route generation, queue-based R&R is used to generate promising routes. Notice that for easy-to-route regions which have no design rule violations after route generation, the IP instances will not be constructed.

C. Candidate Generation with Adaptive Guide Following

Because queue-based R&R approach alleviates the net ordering issue and can produce high quality routes, we leverage fast queue-based R&R for candidate route generation. Routing trees are searched according to the cost of wirelength, via, the design rule violations, and global route guides in the R&R process. The routing trees produced by R&R are treated as candidate routes.

In detailed routing, the routes can be generated with or without the guidance by global routing results. Global routing has a global view of path finding, but lacks of detailed aspects and the results are usually pessimistic, while detailed routing has all the details but lacks of a global view, which does not work well for designs with heavy routing congestion. As a result, we adaptively leverage global routing guides in candidate routes generation. For designs with heavy congestion, the global route guides are used to avoid congested areas in the local detailed routing region in queue-based R&R. Otherwise, the detailed routing paths are searched with more flexibility without guides in the local detailed routing region, for better exploration of detailed routing solution space.

D. Multi-Route Design Rule Checking

Because design rules are typically affected in neighboring regions of a shape, candidate routes far away from each other do not have design rule violations. In addition, by treating each candidate route as a net in design rule checking, multiple candidate routes of a net can be processed at the same time. This transforms checking all combinations of different candidate routes of nets into checking nets with multiple routes once. It dramatically reduce the computational complexity of penalty parameter construction related to design rule violations.

For convenience, we denote the design rule checking which can simultaneously process nets with multiple candidate routes as *multi-route* design rule checking, and the design rule checking which processes nets with single route as *conventional* design rule checking. The difference between conventional and multi-route design rule checking is in handling candidate routes. There is only one route for a net in conventional design rule checking, while multiple routes in multi-route design rule checking.

In multi-route design rule checking, candidate routes of the same net need to be distinguished during checking. We add an auxilury attribute to each geometry (e.g., polygon, rectangle, edge) to distinguish which route it belongs to.

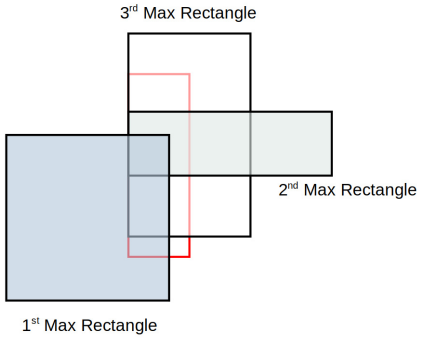


Fig. 2: Checking of non-sufficient metal overlap with multi-route support.

Shapes collected by region query need to be filtered based on the route information. Conditions in design rule checking of different rules are enriched to check route information of objects. When creating DRC violation markers after checking, the route information is added to the markers to record the multi-route checking results.

For checking of same-net rules including minimum width, minimum area, minimum step, non-sufficient overlap, etc, different candidate routes are distinguished during checking. After collecting the shapes of a target net in a region, shapes from a different route are ignored to distinguish multiple routes of a same net. For checking of different-net rules including parallel-run-length (PRL) spacing, end-of-line (EOL) spacing, cut spacing, etc, candidate routes of the same net are also distinguished.

We describe the algorithm of checking non-sufficient metal overlap with multi-route support as an example. The situation of checking is illustrated in Fig. 2. For two maximum rectangles (first and second max rectangles in the figure) extracted from one route of a net, we need to check the size of the intersection between them. There are corner cases about the sizes of the first and second max rectangles, and when there is a third max rectangle of the same route covering the intersection. Algorithm 2 and 3 shows the method of checking non-sufficient metal overlap supporting multiple routes.

In Algorithm 2, line 1 collects all the max rectangles in the neighborhood of given rectangle m within *maxDist* that may cause design rule violations. Line 4 checks if two rectangles n and m belong to the same net. Line 5 checks if n and m belong to the same route. Compared to conventional design rule checking, this line is added to support multi-route checking. Line 6 checks if n and m has non-sufficient metal overlap, which is outlined in Algorithm 3.

In Algorithm 3, for two rectangles m and n of the same net and the same route, line 1 gets the intersection rectangle. Lines 2–3 check if the diagnal length of the intersection exceeds the minimum width. If so, no violation occurs. Lines 4–5 handle a corner case related to minimum width rule, which can be ignored in checking of non-sufficient metal overlap. Lines 6–7 checks if there is a third max rectangle with valid width

Algorithm 2: Filter Shapes to Check NSMetOverlap

```
Input : Max rectangle m
N ← queryMaxRectangles(m, maxDist);
for all n ≠ m in N do
    if isOverlap(n, m) then
        if getNet(n) = getNet(m) then
            if getRoute(n) = getRoute(m) then
                checkNonSufficientMetalOverlap(m, n);
```

Algorithm 3: Check Non-Sufficient Metal Overlap

```
Input : Max rectangle m, n
1 nsRect ← getIntersection(m, n);
2 if diagLen(nsRect ≥ minWidth) then
3     return;
4 if width(m) < minWidth or width(n) < minWidth then
5     return;
6 if hasValid3rdObjWithRoute(nsRect) then
7     return;
8 addMarkerWithRouteInfo(NSMetOverlap);
```

covering the intersection of m and n. To support multi-route checking, the third max rectangle needs to belong to the same route as m and n. After filtering out above corner cases, line 8 adds a new marker between m and n, which also records the route information related to the marker. As shown in Fig. 2, if the rectangle with red outline covers the intersection of the first and the second max rectangles, but belongs to a different route, then it needs to be filtered out.

III. Experiments

In this section, we present the experimental setup and results.

A. Setup

We implement the proposed detailed routing algorithm in C++ programming language and use CPLEX [18] as the IP solver. The queue-based rip-up and reroute for candidate routes generation, and the basis of integrated design rule checking are from TritonRoute-WXL [17]. We use ISPD 2018 detailed routing benchmark suite [19] to evaluate the performance of the detailed router. The statistics of benchmark circuits are listed in Table I. The related design rules include minimum width, minimum area, parallel-run-length spacing, end-of-line spacing, cut spacing, and non-sufficient metal overlap. The detailed router runs on a computer with an Intel CPU @ 2.30GHz and 64 GB RAM. The detailed routing results are verified by design rule checking in Cadence Innovus Implementation Systems 15.2 [20].

B. Comparison with SOTA Results

There are several high-quality academic detailed routers proposed in recent years (e.g. [1]–[7]). Among them, the detailed router in TritonRoute-WXL has the best solution quality, which has the least number of design rule violations and high quality wirelength and via count. So we compare the detailed routing results of the proposed algorithm and the detailed router in TritonRoute-WXL [17]. Both detailed routers run on the same computer using one thread. The global route guides are from ISPD 2018 benchmarks.

TABLE I: Information of ISPD-2018 benchmark circuits.

Circuit	#StdCell	#Block	#Net	#Layer	Tech Node
test1	8879	0	3153	9	45nm
test2	35913	0	36834	9	45nm
test3	35973	4	36700	9	45nm
test4	72090	4	72410	9	32nm
test5	71946	8	72394	9	32nm
test6	107919	0	107701	9	32nm
test7	179865	16	179863	9	32nm
test8	191987	16	179863	9	32nm
test9	192911	0	178858	9	32nm
test10	290386	0	182000	9	32nm

Total wirelength, via count, the number of design rule violations (#DRV), and runtime of detailed routing are listed in Table II. In the table, TR represents the TritonRoute-WXL detailed router. From Table II, we can see that the our detailed routing algorithm IPAG effectively reduces the number of design rule violations in test4. The solutions of IPAG achieve an average of 5.1% (up to 10.4%) reduction on via count, and an average of 0.2% (up to 0.7%) reduction on total wirelength. Interestingly, the runtime of IPAG is smaller or comparable to the TritonRoute-WXL detailed router. This is because of effective reduction of design rule violations by using IP and adaptive global route guidance, in the procedure of iterative region-wise detailed routing.

C. Detailed Analysis

To show the effectiveness of reducing design rule violations by our detailed routing algorithm, we list the the number of design rule violations after each iteration of region-wise detailed routing on test7 in Table III. The TritonRoute-WXL detailed router used 18 iterations to clean the design rule violations, while IPAG used 10 iterations. After each iteration listed in Table III, the number of design rule violations by our detailed router is smaller compared to TritonRoute-WXL detailed routing results. In each iteration, local region-wise detailed routing is conducted in regions containing design rule violations. As a result, the effort taken by succeeding iterations is reduced when the number of design rule violations after a iteration is smaller.

We collect the runtime of important parts in our detailed routing algorithm. The runtime breakdown on test9 is shown in Fig. 3. The runtime composition on other designs is similar. We can see that solving IP instances takes only a fraction of runtime, which indicates the IP instances are moderate in size and efficient to be solved. Multi-route design rule checking takes similar runtime as IP solving, which validates

TABLE II: Results comparison on ISPD-2018 benchmark between TritonRoute-WXL detailed router (TR) and IPAG (Ours). #DRV is the number of design rule violations.

Circuit	Wirelength (μm)		Via Count		#DRV		Runtime (sec)	
	TR	Ours	TR	Ours	TR	Ours	TR	Ours
test1	86440	85851	35416	35459	0	0	60	54
test2	1572819	1570490	361079	355329	0	0	722	662
test3	1751892	1750143	360515	356714	0	0	1132	946
test4	2621650	2617382	729868	725305	7	2	2898	2186
test5	2763865	2759353	906166	821316	0	0	1773	1680
test6	3551832	3546875	1369534	1227756	0	0	2847	2534
test7	6475063	6466423	2228509	2017927	0	0	5524	4930
test8	6503730	6494203	2245372	2038621	0	0	4988	5004
test9	5433663	5425144	2238814	2025410	0	0	4254	4288
test10	6759905	6759507	2419202	2418613	0	0	5722	6474
Average	1.000	0.998	1.000	0.949	–	–	1.000	0.928

TABLE III: Comparison of the number of design rule violations after certain number of DR iterations (#Iter) on test7.

#Iter	0	1	2	3	4	5	8	9	16	17
TR	42762	1112	221	10	6	5	2	2	1	**0**
Ours	41675	573	63	7	5	4	1	**0**	n/a	n/a

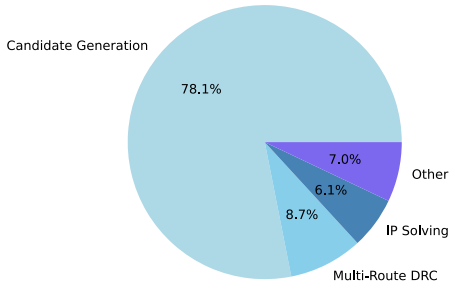
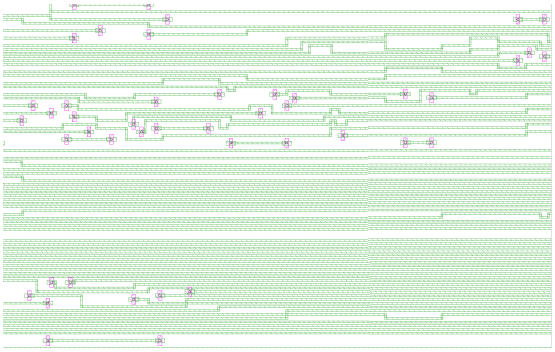
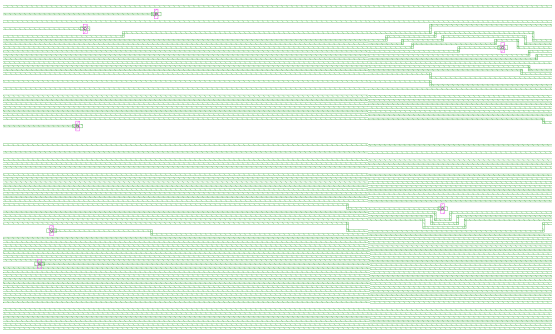


Fig. 3: Runtime breakdown of proposed detailed routing on test9. Candidate generation is conducted in all detailed routing regions using queue-based R&R. Multi-route design rule checking (DRC) and IP solving are carried out if queue-based R&R results have non-zero design rule violations.



(a) TR



(b) Ours

Fig. 4: Layout region of test4 in results of TritonRoute-WXL detailed router (TR) and IPAG (Ours).

the efficiency of simultaneously checking nets with multiple routes.

To visually compare the results, the same layout region in detailed routing solutions by the TritonRoute-WXL detailed router and by IPAG on test4 are captured using Rsyn [21], as shown in Fig. 4. In the figure, the wires are on Metal7, and the via enclosures belong to VIA78 connecting Metal7 and Metal8. By using adaptive global routing guides in candidate routes generation and integer programming in best route selection, IPAG has better solution quality than the TritonRoute-WXL detailed router. There are fewer design rule violations (metal spacing around fat via enclosures or short) and smaller number of vias on high metal layers. This also validates the effectiveness of proposed detailed routing algorithm.

IV. Conclusion

In this work, we propose a high performance detailed routing algorithm named IPAG with a scalable integer programming formulation. High quality candidate routes are generated by queue-based R&R with adaptive global route guidance. Penalty parameters related to design rule violations are efficiently constructed by design rule checking which supports handling nets with multiple routes simultaneously. Validated on ISPD 2018 benchmarks, the proposed detailed

router achieves better solution quality with shorter or comparable runtime as compared to the state-of-the-art results.

REFERENCES

[1] A. B. Kahng, L. Wang, and B. Xu. 2021. "TritonRoute: The Open Source Detailed Router." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40, 3 (2021), 547–559.

[2] G. Chen, C.-W. Pui, H. Li, and E. F. Y. Young. 2020 "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* 39, 9 (2020), 1902–1015.

[3] F.-K. Sun, H. Chen, C.-Y. Chen, C.-H. Hsu, and Y.-W. Chang. 2018. "A multithreaded initial detailed routing algorithm considering global routing guides." In *Proceedings of 2018 IEEE/ACM International Conference on Computer-Aided Design*. 1–7.

[4] S. M. M. Gonçalves, L. S. da Rosa, and F. de S. Marques. 2020. "DRAPS: A design rule aware path search algorithm for detailed routing." *IEEE Transactions on Circuits and Systems—II: Express Briefs"* 67, 7 (2020), 1239–1243.

[5] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Y. Young. 2019. "Dr. Cu 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction." In *Proceedings of 2019 IEEE/ACM International Conference on Computer-Aided Design*, 1–7.

[6] Z. Zhang, G. Liu, T.-Y. Ho, B. Yu, and W. Guo. 2022. "TRADER: A Practical Track-Assignment-Based Detailed Router." In *Proceedings of 2022 Design, Automation and Test in Europe Conference*. 766–771.

[7] A. B. Kahng, L. Wang, and B. Xu. 2022. "TritonRoute-WXL: The Open-Source Router With Integrated DRC Engine." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 4 (2022), 1076–1089.

[8] Y. Lin, T. Qu, Z. Lu, Y. Su, and Y. Wei. 2022. "Asynchronous Reinforcement Learning Framework and Knowledge Transfer for Net-Order Exploration in Detailed Routing." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 9 (2022), 3132–3142.

[9] X. Jia, Y. Cai, Q. Zhou, and B. Yu. 2018. "A Multicommodity Flow-Based Detailed Router With Efficient Acceleration Techniques." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 1 (2018), 217–230.

[10] K. Han, A. B. Kahng, and H. Lee. 2015. "Evaluation of BEOL Design Rule Impacts Using An Optimal ILP-based Detailed Router." In *Proceedings of the 52nd Annual Design Automation Conference*, 68:1–6.

[11] C.-K. Cheng, A. B. Kahng, H. Kim, M. Kim, D. Lee, D. Park, and M. Woo. 2022. "PROBE2.0: A Systematic Framework for Routability Assessment From Technology to Design in Advanced Nodes." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 5 (2022), 1495–1500.

[12] G.-J. Nam, F. Aloul, K. Sakallah, and R. A. Rutenbar. 2001. "A comparative study of two Boolean formulations of FPGA detailed routing constraints." In *Proceedings of 2001 ACM International Symposium on Physical Design.* 222–227.

[13] M. Cho and D. Z. Pan. 2007. "BoxRouter: A New Global Router Based on Box Expansion and Progressive ILP." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 12 (2007), 2130–2143.

[14] M. Cho, K. Lu, K. Yuan, and D. Z. Pan. 2007. "BoxRouter 2.0: Architecture and Implementation of a Hybrid and Robust Global Router." In *Proceedings of 2007 IEEE/ACM International Conference on Computer-Aided Design*, 503–508.

[15] J. Hu, J. A. Roy, and I. L. Markov. 2008. "Sidewinder: A scalable ILP-based router." In *Proceedings of 2008 ACM International Workshop on System Level Interconnect Prediction.* 73–80.

[16] T.-H. Wu, A. Davoodi, and J. T. Linderoth. 2009. "GRIP: Scalable 3D Global Routing Using Integer Programming." In *Proceedings of 2009 ACM/IEEE Design Automation Conference.* 320–325.

[17] TritonRoute-WXL. Accessed June 20, 2022. [Online]. Available: https://github.com/ABKGroup/TritonRoute-WXL

[18] IBM ILOG CPLEX Optimizer. Accessed Aug. 20, 2022. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer

[19] S. Mantik, G. Posser, W.-K. Chow, Y. Ding, and W.-H Liu. 2018. "ISPD 2018 Initial Detailed Routing Contest and Benchmarks." In *Proceedings of 2018 ACM International Symposium on Physical Design.* 140–143.

[20] Cadence Innovus Implementation System 15.2. Accessed Aug. 20, 2022. [Online]. Available: https://www.cadence.com

[21] G. Flach, M. Fogaça, J. Monteiro, M. Johann, and R. Reis. 2017. "Rsyn: An extensible physical synthesis framework." In *Proceedings of 2017 ACM International Symposium on Physical Design*. 33–40.