TOFU: A Two-Step Floorplan Refinement Framework for Whitespace Reduction

Shixiong Kai[1*] , Chak-Wa Pui[1] , Fangzhou Wang[2] , Shougao Jiang[3] , Bin Wang[1] , Yu Huang[3] , and Jianye Hao[1,4]
[1]Huawai Noah's Ark Lab, [2]The Chinese University of Hong Kong, [3]Hisilicon, [4]Tianjin University
[*]kaishixiong@huawei.com

***Abstract*—Floorplanning, as an early step in physical design, will greatly affect the PPA of the later stages. To achieve better performance while maintaining relatively the same chip size, the utilization of the generated floorplan needs to be high and constraints related to design rules, routability, power should be honored. In this paper, we propose a two-step framework, called TOFU, for floorplan whitespace reduction with fixed-outline and soft/pre-placed/hard modules modeled. Whitespace is first reduced by iteratively refining the locations of modules. Then the modules near whitespace will be changed into rectilinear shapes to further improve the utilization. To ensure the legality and quality of the intermediate floorplan during the refinement process, a constraint graph-based legalizer with a novel constraint graph construction method is proposed. Experimental results show that the whitespace of the initial floorplans generated by Corblivar [1] can be reduced by about 70% on average and up to 90% in several cases. Moreover, the resulting wirelength is also 3% shorter due to a higher utilization.**

I. Introduction

Floorplanning has been one of the most crucial stages in modern physical design flow. Having the same chip size, a decent floorplan with higher utilization and shorter wirelength will normally lead to a more compact design with better power, performance, and area (PPA). In alignment with the emerging real-world physical implementation requirements, more constraints have been added to the classical floorplanning problem. As mentioned in [2], floorplanning with a predetermined outline is more practical as it enables hierarchical design for the increasingly complex circuits. Besides, a modern floorplanning algorithm should consider pre-placed modules (PPM).

It has been empirically shown that the fixed-outline and pre-placed-module constraints can make it significantly harder for an algorithm to produce legal floorplans with high utilization as well as short interconnections [3]. Therefore, in this work, we will mainly discuss the realization of a high-utilization legal floorplan with a fixed outline and PPMs.

In recent years, analytical approaches like [4], [5] have shown a good potential of handling the fixed-outline floorplanning problem with PPMs. A two-stage flow is frequently used, which consists of a global distribution and a legalization step. In the global distribution stage, modules will be spread in the fixed outline while minimizing the total wirelength. As a result, module overlaps and out-of-boundary modules usually exist, which are handled in the legalization step. Zhan et al. [4] solve an unconstrained optimization problem to remove the existing module overlaps and penalize those violating the fixed-outline constraint. However, with the appearance of large hard modules, their approach may fail to obtain a legal floorplan. In [5], a pull-push model is introduced for global distribution, whose result is later legalized by a constraint graph-based method. They also explicitly address the existence of PPMs and utilize rectilinear shapes to fix related overlaps. However, in their proposed legalization step, the approach for constraint graph extraction is not robust enough, which may (i) misinterpret the relative positions of two modules or (ii) have unconstrained node pairs. Those factors can lead to possible overlaps between modules and longer wirelength. The misinterpretation of relative position of modules will be discussed specifically in Section III-B. Most importantly, none of the aforementioned works explicitly treat whitespace as an optimization goal.

To further improve the utilization of a preliminary floorplan, we present TOFU, an efficient floorplan refinement framework that is aware of the fixed-outline and pre-placed-module constraints. Our contributions can be summarized as follows.

- We propose a two-step framework for floorplan whitespace removal with the awareness of fixed-outline and pre-placed-module constraints. It can be used as a postprocessing step for any floorplanning methods to further improve the utilization of their solutions.
- A robust and effective constraint graph extraction methodology is introduced to improve the one mentioned in [5]. Compared to the original method, ours can better model the relative positions of two modules under the presence of PPMs, hard modules, and extra-large modules.
- Two methods that target different scenarios are proposed to reduce the whitespace between modules. A module relocation-based method is first used to address the whitespace induced by poor initial solutions and misinterpretation of relative positions when building the constraint graphs. Then, an area reallocation-based method is used to handle the whitespace that cannot be removed by rectangular shapes.

The rest of this paper is organized as follows. Section II introduces some floorplanning-related methodologies and defines the floorplan refinement problem. Section III describes our proposed frameworks for floorplan refinement. Section IV presents our experimental settings and results. Section V delivers a conclusion of our work.

II. Preliminaries

A. Floorplanning

Floorplanning, as the "bridge" between logic synthesis and physical design, is a well-studied topic in academics. Given the netlist and constraints, the floorplanning problem is used to determine the shapes and locations of the modules such that whitespace and wirelength are minimized. Note that, the routability can be achieved through feedthrough [6] with near zero whitespace floorplan solution. In practice, engineers also use channel routing [7] by digging channels between modules within the allowable reduction of module area. The constraints in floorplanning can be categorized into the following types.

- Module-property constraints: There are three types of modules: (1) soft modules, (2) hard modules, and (3) pre-placed modules (PPM). Both the shapes and locations can be changed for soft modules while only locations can be changed for hard modules. As for PPMs, their shapes and locations should remain untouched as given.
- Fixed-outline constraint: Fixed outline is a rectilinear region where the modules can only be placed inside. This constraint is usually used in hierarchical designs where modules should stay inside the boundary of the modules in their parent hierarchy .

Floorplan refinement, similar to detailed placement in the placement problem, is essential to the modern floorplanning flow. Different floorplan refinement methods can also be easily used as a plug-in for any floorplanning method to further improve the quality or remove the constraint violations.

B. Constraint Graph-based Legalization

Given an initial placement of rectangular shapes (modules, cells, devices, etc), the legalization problem is to determine the height, width, and location of each rectangle such that there is no constraint violation and the target metrics are optimized. The given input of legalization is

978-3-9819263-7-8/DATE23/© 2023 EDAA

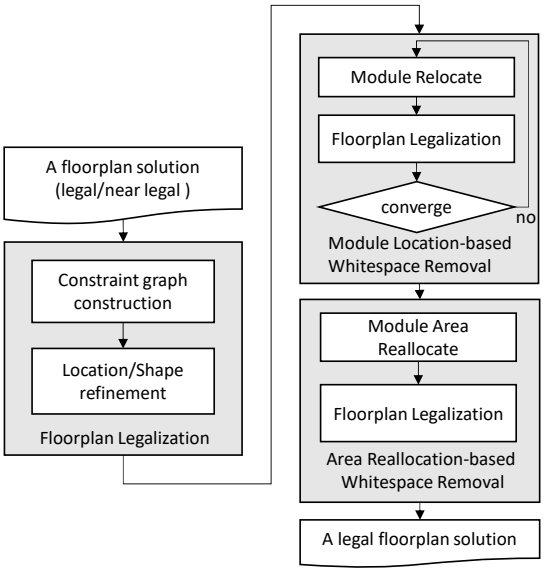


Fig. 1: Overall Flow of the Proposed Algorithm.

usually considered as a solution that is nearly legal. Hence, the relative position between the rectangles should be honored and the constraint graph-based legalization is commonly used to achieve this. A constraint graph including two directed acyclic graphs denoted as G_v and G_h. Each node n_i in the graph represents the ith rectangle m_i and an edge $e_{ij} \in G_h$ ($e_{ij} \in G_v$) from n_i to n_j suggests that m_j should be on the right (bottom) of m_i. The legalization can be formulated as Equation (1).

$$\begin{aligned} \min \quad & \text{target metrics} && \text{(1a)} \\ s.t. \quad & w_i \cdot h_i = a_i && \text{(1b)} \\ & x_i + w_i \le x_j \quad \forall e_{ij} \in G_h && \text{(1c)} \\ & y_i + h_i \le y_j \quad \forall e_{ij} \in G_v && \text{(1d)} \\ & \text{other application specific constraints.} && \text{(1e)} \end{aligned}$$

Let x_i, y_i, w_i, h_i and a_i be the bottom-left coordinate, width, height and area of the ith rectangle.

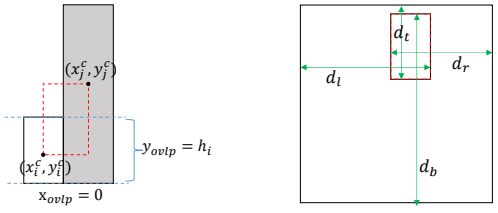
C. Problem Definition

Given a netlist of modules and their initial positions and shapes, the floorplan refinement problem is to reduce the whitespace such that there is no overlap between modules and all constraints (i.e. module-property and fixed-outline) are satisfied. Note that, here we assume the input solutions should be either legal or near legal (modules can be legalized with a small displacement) and we should honor the initial positions of modules for better wirelength.

III. Proposed Algorithms

A. Overview

As shown in Figure 1, our algorithm consists of three parts: (1) floorplan legalization, (2) module location-based whitespace removal, and (3) area reallocation-based whitespace removal. In the first step, a constraint graph-based floorplan legalization method is proposed to ensure all constraints are satisfied and improve the utilization. And it will be called multiple times during the whole algorithm process. In the second step, the locations of modules near whitespace are adjusted to handle the whitespace induced by the modules' initial positions. In the third step, starting from the bottom region, the modules will be changed from rectangle to rectilinear shape to fill the whitespace. Details of our legalization method and whitespace removal framework will be discussed in Sections III-B and III-C respectively.



(a) **(b)**

Fig. 2: Misinterpretations of relative position in [5].

B. Floorplan Legalization

During floorplan legalization, we need to determine the shape and location of each module such that all the constraints are satisfied. The constraint graph-based method mentioned in Section II-B is used and the formulation is as below.

$$\begin{aligned} \min \quad & |W - \overline{W}| + |H - \overline{H}| && \text{(2a)} \\ s.t. \quad & W = \max_{\forall m_i \in M} (x_i + w_i) && \text{(2b)} \\ & H = \max_{\forall m_i \in M} (y_i + h_i) && \text{(2c)} \\ & w_i \cdot h_i = a_i && \text{(2d)} \\ & \frac{1}{ar_i} \le \frac{w_i}{h_i} \le ar_i && \text{(2e)} \\ & x_i + w_i \le x_j \quad \forall e_{ij} \in G_h && \text{(2f)} \\ & y_i + h_i \le y_j \quad \forall e_{ij} \in G_v && \text{(2g)} \end{aligned}$$

Let x_i, y_i, w_i, h_i, a_i and ar_i be the bottom-left coordinate, width, height, area and maximum aspect ratio of the ith module. The width and height of an outline is denoted by (W, H). The objective is to minimize the difference between the actual outline (W, H) and fixed outline $(\overline{W}, \overline{H})$. The actual outline is determined by the top right corner of each module as shown in Equations (2b) and (2c). The constraints of module area and aspect ratio are modeled in Equations (2d) and (2e). Finally, Equations (2f) and (2g) ensure that there is no module overlap as long as the constraint graph is built appropriately. Details of our constraint graph construction method will be introduced in the following section. Note that, unlike [5], we fix the locations/shapes of PPMs and the shapes of hard modules when solving Equation (2). Equation (2) can be solved by second order cone programming (SOCP) [5] or Lagrangian relaxation [8]. In our implementation, we use Gurobi [9] to solve the equations as an SOCP problem for the purpose of easy integration.

1) Constraint Graph Construction: Our constraint graph (CG) construction method is based on the one mentioned in [5]. First, the Delaunay triangulation (DT) method is used to find the neighboring information among the modules. Any two nodes connected in the DT graph represent two modules that are adjacent to each other. An edge will be constructed in either G_v or G_h to represent their relative position. However, the approach to construct CG from DT graph in [5] has two major drawbacks as discussed in Section I.

Figure 2 shows two examples of how the relative position of two modules can be misinterpreted. In Figure 2a where m_i and m_j are separated, an edge is added to G_v since $(x_j^c - x_i^c)$ is smaller than $(y_j^c - y_i^c)$. Figure 2b shows an example for two overlapped modules m_i and m_j, an edge is added to G_v since height is larger than width for the overlapped area. To overcome these shortcomings, our method determines the geometric relation of two modules m_i and m_j as follows,

- One completely covered by the other: Without loss of generality, we assume m_i is completely covered by m_j. We choose a direction that we can move m_i out of m_j such that the distance is minimized. The relative position of the two after overlap removal determines how they are connected in the CG.
- Otherwise: Let p_i^x, p_i^y (p_j^x, p_j^y) be the projection of m_i (m_j) on X-axis (Y-axis). If their projection has no overlap on both axes, we will

Algorithm 1 Module relocation-based whitespace removal.

Require: A legal floorplan.
Ensure: A legal floorplan with less whitespace.
1: **for each** empty grid $g_{ij} \in G$ **do**
2: let w_{ij} be the largest whitespace of g_{ij}
3: let M_{ij} be the adjacent modules set of w_{ij} sorted by relative position to w_{ij} (top ¿ right ¿ left)
4: **for each** module $m_k \in M_{ij}$ **do**
5: remove m_k from layout
6: recalculate the largest whitespace w'_{ij} of g_{ij}
7: **if** m_i can be successfully placed in w'_{ij} with appropriate aspect ratio and overlap less than 10% of its area **then**
8: commit the result
9: **break**
10: **end if**
11: **end for**
12: **end for**
13: legalize the floorplan
14: go to line 1 if whitespace or wirelength is improved

follow the rules in [5]. Otherwise, an edge is added to G_h (G_v) if the overlapped length on Y-axis (X-axis) is longer.

After building the CG from DT graph using our method, the constraint graph may not be sufficient to remove all overlaps. First, two matrices representing the relative position of all module pairs in the CG are built using the method in [10]. Then, for any two modules that have no relation in the CG, we will use the method mentioned previously to add an edge to the corresponding CG.

C. Two-Step Whitespace Removal Framework

Given a legal floorplan, our two-step whitespace removal framework will try to improve the utilization by relocating and reshaping the modules near whitespace. In this section, we will introduce how the target whitespace is identified in our method and how they can be reduced by our methods.

To identify the whitespace in a given floorplan, the layout is first decomposed into a 2D grid G whose width and height are m and n respectively. Then for any grid cell $g_{ij} \in G$, we will mark the modules that have overlaps with it by traversing each module and the grid cells it covers. After this preprocessing, we can easily find the rectangular whitespace by scanning the grid from the lower-left corner row by row. To be specific, for an empty grid cell g_{ij}, we will find the largest rectangular whitespace w_k whose lower left corner is g_{ij} such that all $g_{pq} \in w_k$ are empty. Note that, since long and narrow channels will usually induce routability issues, we will skip the whitespace that has a big aspect ratio (9 in our implementation).

The general idea of our whitespace removal framework is to squeeze the whole floorplan towards the bottom left corner. The first step of our framework is to remove the whitespace induced by poor initial positions while the second step targets those that cannot be resolved by rectangular modules. Details of these two steps will be discussed in the next sections. Note that, in the examples of this section, PPMs are denoted as f_i and marked in grey while the others are assumed to be soft modules.

1) Module Location-based: As shown in Figure 1, a constraint graph-based legalization will be used to determine the shape and location of each module during the whole whitespace refinement process. The legalization results are mainly determined by the module initial locations, which is represented as a constraint graph in legalization. Some large whitespace may be induced by either constraint graph misinterpretations or poor initial locations, which are hard to observe until the actual floorplan is generated. For example, as shown in Figure 3a, the horizontal constraints among f_1, m_1, m_2 together with the vertical constraints between m_2, f_3 will induce large whitespace. This kind of whitespace can be easily

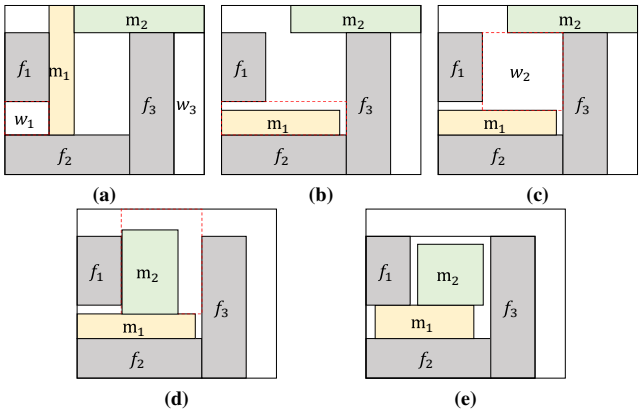


Fig. 3: Relocate modules adjacent to whitespace.

Algorithm 2 Area reallocation-based whitespace removal.

Require: A legal floorplan.
Ensure: A legal floorplan with less whitespace.
1: **while** utilization can be improved **do**
2: let M^r be the set of modules that have been reshape
3: scan for a candidate whitespace w_i from lower left corner
4: m_i=`Reshape`(w_i) and add m_i to M^r
5: legalize the floorplan with all modules lower than m_i fixed
6: set the modules in M^r to be fixed in later iterations
7: **end while**

8: **function** `Reshape`(w_i)
9: let W_f be the set of whitespace to be filled
10: add w_i to W_f
11: find the best module m_i to fill w_i
12: **for each** adjacent whitespace w_j of m_i **do**
13: add w_j to W_f if m_i is the best module to fill w_j
14: **end for**
15: reallocate the region of m_i to W_f and m_i
16: **return** m_i
17: **end function**

resolved by relocating the nearby modules. Hence, in the first step of our whitespace removal framework, we will improve the utilization by relocating the modules near whitespace.

Our module relocation-based method is an iterative process as shown in Algorithm 1. As we are trying to push the modules towards the bottom left corner of the outline, modules below w_{ij} will be skipped and the selected modules M_{ij} are processed in an order of their relative positions to w_{ij}, that is, we will first consider modules that are on top of w_{ij}, then right, and finally left. Note that, in line 7, when we try to place a module m_{ij} in w'_{ij}, we will reshape m_{ij} to the same aspect ratio as w'_{ij} and align their lower left corners. If the overlaps between m_{ij} and the others are less than 10% of its area, the relocation is considered to be successful. Figure 3 shows an example of Algorithm 1. Whitespace w_3 is first identified but is skipped since its aspect ratio is too large. Then w_1 is processed and module m_1 is selected as the module to relocate. After removing m_1 from the layout, w_1 will be expanded to a bigger rectangle as shown in Figure 3b, where m_1 can be successfully placed. As shown in Figures 3c and 3d, m_2 will be relocated to w_2 similarly. Finally, these modules will be refined by the legalization method mentioned in Section III-B.

2) Area Reallocation-based: After relocating the modules around whitespace, there are still some whitespace that cannot be removed due to the limitation of rectangular shapes. Hence, in this stage, some rectangular modules will be transformed into rectilinear shapes to further reduce the whitespace.

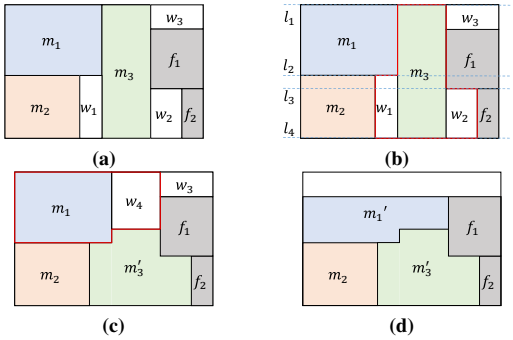


Fig. 4: Area reallocation for a module adjacent to whitespace.

Details of our area reallocation-based method are shown in Algorithm 2. Similar to the module relocation-based method, reshaping and floorplan legalization are called iteratively to improve the utilization. When selecting the best module to fill a whitespace w_i, we will consider its adjacent modules in the same way as the one mentioned in the module relocation-based method. For any candidate module m_i of whitespace w_i, the longer their common edge is, the more m_i is preferred. To avoid producing "glitches" on the boundary of a module, if the length of the common edge between m_i and w_i is less than 20% of the touching edge of m_i, m_i will not be considered as a candidate for filling w_i. For example, in Figure 4a, when processing w_1, m_1 is skipped due to its short common edge with w_1. When using m_i to fill the region of W_f and m_i, the rectilinear region is first cut into rectangles horizontally as shown in Figure 4b. Then these rectangles are filled by the area of m_i from the bottom as shown in Figure 4c.

Since we restrict the shape of non-rectangle modules with many rules, which make the module a relatively regular shape, and feed-through [6], pin assignment and channel routing [7] follow floorplan to achieve routability, non-rectangle modules have little impact on global routing and detail routing.

IV. Experimental Results

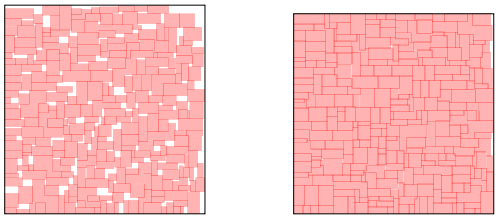
Our floorplan refinement framework is implemented in C++ and all the experiments are performed on a Linux server with 3.00 GHz Intel Xeon CPUs (sixty threads available). To show the effectiveness and the scalablity of the proposed framework, we conduct a series of experiments with GSRC [11], MCNC [12], and IBM-HB+ [13] benchmark suites. Corblivar [1], [14], a state-of-the-art SA-based floorplanner we can get is used to generate a set of legal solutions[1], which will be referred to as the base floorplans in the rest of the paper. Note that, the proposed framework does not assume a specific floorplan solution as input, allowing it to be adopted as a postprocessing step by a wide range of floorplanning methods.

According to the usual assumptions of academia [3], [5], we assume that all pins are located at the center of each module's bounding box, IO pads' coordinates will shrink with the actual outline, and set the maximum aspect ratio to be 3 for all soft modules. HPWL is computed based on the center of each module.

A. Effectiveness for Whitespace Removal

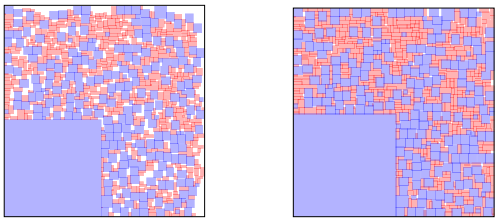
We first use our proposed framework on the base floorplans generated by Corblivar. Quantitative results are shown in Table II. We can see that

[1]We reference configurations in the configs/2dies/regular folder from the repository. For IBM-HB+ cases, configurations are adapted from ibm01.conf and ibm01_tech.conf; for GSRC and MCNC benchmarks, configurations are adapted from Corblivar.conf and Technology.conf. Specifically, in technology files, we set "Layers for 3D IC" and "Scaling factor for block dimensions" to 1. The width and height of fixed outline are both set to $\sqrt{1.3 \times \text{totalModuleArea}}$. Since thermal effect is not considered in this work, we use power files all filled with 1s. For each case, we run Corblivar for 25 times and pick the legal floorplan with the shortest wirelength.



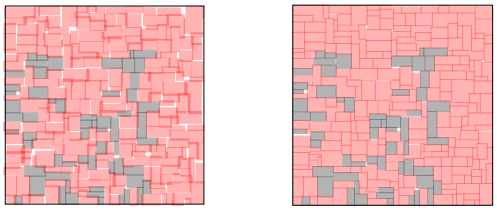
(a) Generated by Corblivar. **(b)** After processed by TOFU.

Fig. 5: Floorplans of n300.



(a) Generated by Corblivar. **(b)** After processed by TOFU.

Fig. 6: Floorplans of ibm01.



(a) After random perturbation. **(b)** After processed by TOFU.

Fig. 7: Floorplans of n300 where PPMs are marked in grey.

TABLE I: Experiments on Illegal Floorplans with PPMs

Benchmarks		Before Perturbation		Refined by TOFU	
Design	#(PPM)	HPWL	WS (%)	HPWL	WS (%)
n10	2	36,059	1.76	35,186	0.70
n30	6	109,222	0.62	109,075	0.73
n50	10	148,419	0.76	148,009	0.86
n100	20	252,250	1.30	252,100	1.40
n200	40	485,313	1.63	484,949	1.58
n300	60	713,212	1.56	715,668	2.18
ami33	6	71,039	0.63	71,071	0.59
ami49	9	1,144,420	0.56	1,141,580	0.69

our framework produces floorplans with much smaller whitespace while being able to reduce the total wirelength, especially for cases with only soft modules. For large cases like ibm01, ibm03, and ibm07, it is the first time that such low levels of whitespace are achieved. To the best of our knowledge, the lowest reported whitespace ratios are 13%, 14%, and 12% respectively from [15]. Figure 5 (Figure 6) show the floorplans before and after refinement for case n300 (ibm01), where red and blue colors are used to mark soft and hard modules respectively. It can be observed that the resulting floorplans are more compact, leading to the aforementioned wirelength reduction. Besides, our proposed framework finishes the most time consuming case (ibm04) in 175 seconds, which is only 15% of the runtime of Corblivar. The scalability of our proposed framework is therefore demonstrated.

B. Refinement with Pre-Placed Modules

We then demonstrate TOFU's ability of handling pre-placed modules (PPM) in illegal floorplans. For this experiment, we use GSRC and MCNC benchmark. As the original benchmark does not include any PPM, we need to generate them ourselves, which will be discussed as below.

TABLE II: Experimental Results Based on Corblivar

Benchmarks				Corblivar [1], [14]			Refined by TOFU				
Design	#(Soft)	#(Hard)	#(Nets)	HPWL	WS (%)	RT(s)	HPWL	Δ(HPWL)	WS (%)	Δ(WS)	RT(s)
n10	10	0	118	37,986	10.17	0.51	35,213	7.30%	1.52	85.06%	1.35
n30	30	0	349	112,947	9.01	2.57	109,189	3.33%	0.61	93.18%	0.97
n50	50	0	485	155,775	11.14	5.88	147,650	5.22%	0.56	94.94%	1.55
n100	100	0	885	266,771	11.55	21.74	250,942	5.93%	1.22	89.39%	2.58
n200	200	0	1585	508,774	11.67	83.26	484,507	4.77%	1.57	86.52%	3.81
n300	300	0	1893	750,367	12.88	189.04	712,547	5.04%	1.54	88.05%	8.21
ami33	33	0	123	72,784	11.45	1.72	71,069	2.36%	0.63	94.48%	1.15
ami49	49	0	408	1,159,160	17.16	4.41	1,144,620	1.25%	0.56	96.75%	1.58
ibm01	665	246	5829	10,188,200	15.59	325.22	9,831,540	3.50%	3.64	76.68%	64.67
ibm02	1200	271	8508	22,190,500	16.42	1046.14	21,433,600	3.41%	5.57	66.11%	125.66
ibm03	999	290	10279	26,788,000	15.58	743.35	26,704,700	0.31%	6.48	58.45%	169.35
ibm04	1289	295	12456	30,896,400	13.36	1214.47	29,999,500	2.90%	3.18	76.23%	174.93
ibm06	571	178	9963	25,643,300	13.17	337.95	24,975,600	2.60%	3.72	71.78%	40.58
ibm07	829	291	15047	47,771,700	17.30	610.58	46,887,900	1.85%	7.03	59.38%	86.01
ibm08	968	301	16075	53,491,000	17.35	917.82	52,933,700	1.04%	4.64	73.27%	119.87
ibm09	860	253	18913	64,156,000	17.14	759.60	62,747,600	2.20%	3.82	77.73%	99.52
ibm10	809	786	27508	136,315,000	18.54	1915.04	133,782,000	1.86%	16.16	12.85%	53.95
ibm11	1124	373	27477	98,677,800	17.78	1501.49	94,878,900	3.85%	8.95	49.66%	57.6
ibm12	582	651	26320	120,516,000	18.80	1100.95	119,978,000	0.45%	15.98	14.99%	32.10
ibm13	530	424	27011	114,411,000	18.29	638.21	110,913,000	3.06%	9.69	47.02%	13.67
ibm14	1021	614	43062	240,371,000	22.65	1266.23	219,871,000	8.53%	8.71	61.56%	57.42
ibm15	1019	393	52779	278,768,000	13.98	2172.13	272,761,000	2.15%	8.13	41.81%	32.97
ibm16	633	458	47821	327,233,000	19.56	997.38	313,025,000	4.34%	11.52	41.13%	19.02
ibm17	682	760	56517	391,821,000	21.91	1618.57	368,221,000	6.02%	9.78	55.35%	43.36
ibm18	658	285	42200	263,100,000	18.45	1012.85	246,894,000	6.16%	3.45	81.32%	23.66
Avg.	/	/	/	/	/	/	/	3.58%	/	67.75%	/

* WS=whitespace; RT=runtime; Δ(HPWL) and Δ(WS) represent the reduction ratios of the corresponding metrics.

To show the robustness of our method, we randomly select some modules such that the pre-placed ones make up 20% of all modules. One assumption about PPMs is that they are usually placed in good locations. Otherwise, the floorplan utilization can never be improved since the outline is determined by those poorly placed modules. In our experiment, we notice that the results after legalization are good enough for setting the locations of PPMs. Hence, this part of experiment is conducted as follows.

1) Legalize the results from Corblivar.
2) Select the modules to be fixed.
3) Fix the locations of PPMs according to the legalization result.
4) Perturb the locations of movable modules.
5) Run our method to verify its robustness and effectiveness.

Experimental results are shown in Table I, which demonstrate that we can retrieve a legal floorplan even with PPMs. Meanwhile, a compact floorplan can still be generated, as shown in Figure 7b.

V. Conclusion

In this paper, we propose a two-step whitespace removal framework, which can be used as a postprocessing step for any floorplanning tool. Several methods have been proposed to reduce whitespace under the fixed-outline and pre-placement-module constraints. Experimental results show that our methods are practical and scalable. Future works may include modeling more PPA-related constraints, such as routing congestion, timing, etc. Also, an algorithm to produce a near legal solution with rectilinear shape modules modeled will help further improve the performance of the current floorplanning toolchain. Last but not the least, analytical methods still need investigation to better model floorplanning specific characteristics, such as rectilinear shape, changeable rectangular shape, gaps to the final floorplans, etc.

References

[1] J. Knechtel, E. F. Young, and J. Lienig, "Planning massive interconnects in 3-d chips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1808–1821, 2015.

[2] A. B. Kahng, "Classical floorplanning harmful?," in *Proceedings of the 2000 international symposium on Physical design*, pp. 207–213, 2000.

[3] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE TVLSI*, vol. 11, no. 6, pp. 1120–1135, 2003.

[4] Y. Zhan, Y. Feng, and S. S. Sapatnekar, "A fixed-die floorplanning algorithm using an analytical approach," in *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pp. 771–776, 2006.

[5] J.-M. Lin and Z.-X. Hung, "Ufo: Unified convex optimization algorithms for fixed-outline floorplanning considering pre-placed modules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 7, pp. 1034–1044, 2011.

[6] T. Koide, S. Wakabayashi, and N. Yoshida, "Pin assignment with global routing for vlsi building block layout," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1575–1583, 1996.

[7] J. Lienig and K. Thulasiraman, "A genetic algorithm for channel routing in vlsi circuits," *Evolutionary Computation*, vol. 1, no. 4, pp. 293–311, 1993.

[8] C. C. Chu and E. F. Young, "Nonrectangular shaping and sizing of soft modules for floorplan-design improvement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 1, pp. 71–79, 2004.

[9] Gurobi Optimization Inc., "Gurobi optimizer reference manual." http://www.gurobi.com, 2016.

[10] B. Xu, S. Li, C.-W. Pui, D. Liu, L. Shen, Y. Lin, N. Sun, and D. Z. Pan, "Device layer-aware analytical placement for analog circuits," in *Proceedings of the 2019 International Symposium on Physical Design*, pp. 19–26, 2019.

[11] W. Dai, L. Wu, and S. Zhang, "GSRC Benchmarks." http://vlsicad.eecs.umich.edu/BK/GSRCbench/, 2000.

[12] Microelectronics Center of North Carolina (MCNC), "MCNC Benchmarks." http://vlsicad.eecs.umich.edu/BK/MCNCbench/.

[13] A. N. Ng, R. Aggarwal, V. Ramachandran, and I. Markov, "IBM-HB+ benchmarks." http://vlsicad.eecs.umich.edu/BK/ISPD06bench/, 2006.

[14] J. Knechtel, "Corblivar." https://github.com/IFTE-EDA/Corblivar.

[15] J.-M. Lin, T.-T. Chen, Y.-F. Chang, W.-Y. Chang, Y.-T. Shyu, Y.-J. Chang, and J.-M. Lu, "A fast thermal-aware fixed-outline floorplanning methodology based on analytical models," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.