MORPH: More Robust ASIC Placement for Hybrid Region Constraint Management

Jing Mai[1,2,3], Zuodong Zhang[2,3], Yibo Lin[2,3,4*], Runsheng Wang[2,3,4], Ru Huang[2,3,4]
{[1]School of Computer Science, [2]School of Integrated Circuits, [3]Institute of EDA}, Peking University
[4]Beijing Advanced Innovation Center for Integrated Circuits
{jingmai, zuodongzhang, yibolin, r.wang, ruhuang}@pku.edu.cn

ABSTRACT

Modern ASIC placement tools encompass three categories of region constraints: default regions, fence regions, and guide regions. Region constraints pose significant challenges to existing placement algorithms, compromising the versatility and robustness required for diverse placement workloads. In this work, we propose **MORPH**, a more robust ASIC placer designed for hybrid region constraints. We integrate hybrid region constraints into a unified multi-electrostatic formulation that features a shared electrostatics model and a binary-lifting-based region pruning algorithm. We develop a more robust nonlinear placement framework that includes second-order information and a hybrid-region-aware legalization algorithm to address convergence issues. Experiments on the ISPD 2015 benchmark suite demonstrate 5.6-14.3% HPWL improvement and 10-24% overflow reduction compared to state-of-the-art region-aware placers. Further experiments on the ISPD 2015 benchmark suite and its variants show that the proposed techniques can achieve over 30% HPWL improvement and up to a twofold reduction in overflow with more stable convergence.

1 INTRODUCTION

Region constraints are an essential feature in contemporary ASIC CAD tools [1], allowing chip designers to allocate macros or standard cells to specified areas on the chip layout based on their domain knowledge. This leads to different instances having different placeable areas, making the region-aware placement problem extremely challenging. Moreover, contemporary ASIC CAD tools are also required to manage hybrid types of region constraints simultaneously, such as fence regions, default regions, and guide regions, further increasing the difficulty of the region-aware placement problem. Therefore, a placement framework that can manage hybrid region constraints to achieve high-quality solutions is urgently needed.

The region-aware placement algorithms proposed in recent years are predominantly based on analytical approaches [13, 16, 19, 21]. Analytical placers use mathematical analysis and optimization techniques to efficiently and effectively achieve a placement solution. `NTUPlace4dr` [21] proposes handling fence region constraints through fence region-aware clustering, along with fence-region-aware density and wirelength models. `Eh?Placer` [16] and `RippleDR` [13] suggest using an upper-bound-lower-bound optimization method in conjunction with look-ahead region-aware rough legalization to manage fence regions. `DREAMPlace 3.0` [19] introduces a multi-electrostatics-based placement model for fence region constraints, achieving superior placement results in wirelength optimization. This technique can also be adapted to FPGA placement with similar region constraints [4, 35, 36, 39, 47, 48]. However, existing analytical placers can only manage fence regions and fail to support complex designs with hybrid region constraints. Furthermore, existing analytical placers still need to resort to randomization to address robustness issues, which can reduce solution quality and requires parameter tuning when facing newly released benchmarks with more complex region constraints and stricter density requirements.

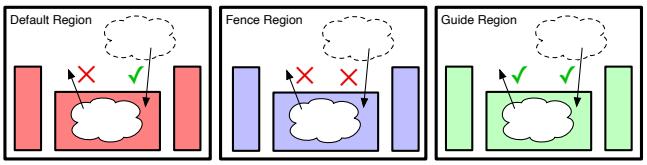

Figure 1: The three sub-figures respectively display the instance layout constraints under the default region (red), fence region (purple), and guide region (green). Each region consists of several rectangular sub-boxes. Solid cloud shapes represent instances subject to the region constraint; dashed cloud shapes represent other instances not subject to the region constraint.

In this work, we propose `MORPH`, a more robust ASIC placer for managing hybrid region constraints. We address region-aware placement by considering hybrid region constraints through an innovative shared electrostatics formulation. Our placement methodology enhances the solution quality of multi-electrostatic-based algorithms, ensuring robust convergence and superior stability. The key contributions of our work are as follows:

- We propose a shared electrostatics model coupled with a binary-lifting-based region pruning algorithm that effectively integrates hybrid region constraints into a unified electrostatics formulation.
- We propose a wirelength-prioritized penalty method for guide regions to ensure balanced optimization without compromising the critical aspect of wirelength minimization.
- We propose a modified Nesterov's accelerated LBFGS algorithm with second-order information that significantly improves the solution quality and the stability of the placement process with minor runtime overhead.

Experimental results on the ISPD 2015 benchmark suite [5] demonstrate that our proposed algorithm adeptly addresses hybrid region constraints with high quality and stability. We achieve a 5.6-14.3% improvement in Half-Perimeter Wirelength (HPWL) and a 10-24% reduction in overflow when compared to the state-of-the-art fence region-aware placers. Moreover, further experiments indicate that

*Corresponding author

ICCAD '24, October 27–31, 2024, New York, NY, USA

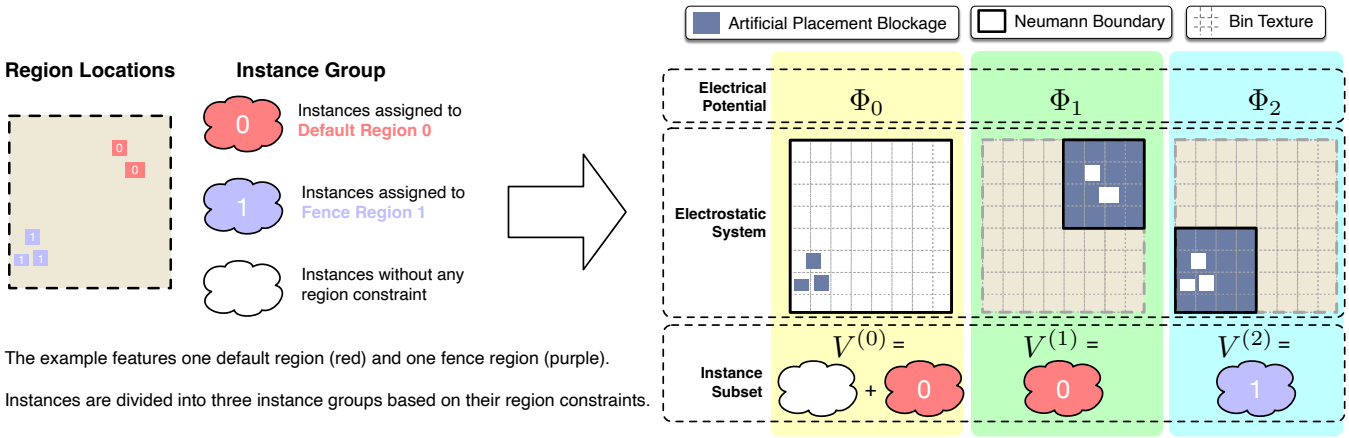

Figure 2: A schematic diagram showing how to construct the shared electrostatics model (Section 3.1.3) along with the shrinked Neumann boundary to minimize memory usage (Section 3.1.4). Each Electrostatics system is constructed solely within the Neumann boundary. All electrostatic field systems share the same bin sizes and have aligned bin textures (dashed line) to ensure consistency in the calculation of contributions to density and overflow when the same instance is in different electrostatic systems.

our proposed optimization techniques outperform those from previous works, achieving over a 30% improvement in HPWL and up to a 2-fold reduction in overflow with more stable convergence.

2 PRELIMINARIES

In this section, we introduce the specification of region constraints, the outline of the multi-electrostatics-based placement algorithm for fence regions, and the background of unconstrained optimization algorithms.

2.1 Region Constraints

ASIC CAD tools provide *region constraints* to assist designers in restricting the placement locations of certain instances [2]. A region constraint consists of 1) a region composed of several rectangular sub-box areas on the chip layout (these rectangular sub-box areas can be non-adjacent), and 2) an instance group subject to this constraint. There are three types of region constraints, namely default regions, fence regions, and guide regions. Specifically, the differences among the three are as follows:

- **Default Region** (*hard constraint*): All instances subject to this constraint must be placed inside the region boundaries, and other instances can also be placed inside the region.
- **Fence Region** (*hard constraint*): All instances subject to this constraint must be exclusively placed inside the region boundaries. No other instances are permitted inside the region.
- **Guide Region** (*soft constraint*): All instances subject to this constraint should be placed inside the region boundaries. However, it is a preference, not a hard constraint. Other constraints, such as wire length and timing, can override this preference.

Figure 1 illustrates the differences between these three types of region constraints. Generally, an instance is subject to at most one region constraint. Fence regions must not overlap[1], while default and guide regions can overlap.

[1]Otherwise, by definition, no instance can be placed at the overlapping area.

2.2 Multi-Electrostatics-based Placement Algorithm for Fence Region Constraint

The electrostatic-based placement algorithm is one of the best-performing placement algorithms in recent years, achieving state-of-the-art placement effects on many workloads [12, 19, 30, 32–34]. This algorithm has subsequently been extended to a multi-electrostatics-based placement algorithm [19, 35–37, 39, 47, 48].

The multi-electrostatics-based placement algorithm aims to minimize the wirelength cost while adhering to multiple density constraints in an optimization problem framework. A density constraint is relaxed to a density penalty term, analogous to the potential energy of an electrostatic system, where instances are modeled as electric particles, as in `DREAMPlace 3.0` [19]. The relaxed optimization formulation is as follows:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \widetilde{W}(\boldsymbol{x},\boldsymbol{y}) + \sum_{s\in S} \lambda_s \Phi_s(\boldsymbol{x}^{(s)},\boldsymbol{y}^{(s)}), \tag{1}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ denote the locations of instances, $\widetilde{W}(\cdot)$ denotes the wirelength objective, S is the set of electrostatic systems, $\boldsymbol{x}^{(s)}$ and $\boldsymbol{y}^{(s)}$ denote the locations of instances assigned to electrostatic system $s \in S$, and $\Phi_s(\cdot)$ and λ_s denote the electrical potential energy of electrostatic system $s \in S$ and the corresponding density multiplier, respectively. The density constraints can be satisfied by gradually increasing the density weights during the optimization process.

The multi-electrostatics-based placement algorithm is suitable for scenarios with various placeable area requirements for different instances. The fundamental idea of this algorithm is that instances with the same placeable area requirement are assigned to the same electrostatic system; in each electrostatic system, illegal placement areas are blocked by *artificial placement blockages*, ensuring that instances assigned to this electrostatic system can only be placed within legal placement areas. The artificial placement blockages for different electrostatic systems may vary, allowing instances assigned to different electrostatic systems to have distinct legal placement areas.

Such scenarios commonly occur in FPGA placement problems [3, 6, 7, 9, 11, 17, 18, 23–29, 35–39, 44–48, 51, 52, 55] (where instances

		Instance Subset $V^{(k)}$	Artificial Placement Blockages
$\sum_{k=0}^{K_1-1} \lambda_k \mathcal{D}_k$	$k = 0$	Instances not subject to any region constraint + Instances assigned to any default region or guide region.	Areas within any fence region.
	$k \in \{1, 2, \ldots, \lvert R^D \rvert\}$	Instances assigned to default region $R_i^D (i = k-1)$	Areas other than the default region R_i^D.
	$k \in \{\lvert R^D \rvert + 1, \ldots, \lvert R^D \rvert + \lvert R^F \rvert\}$	Instances as signed to fence region $R_i^F (i = k-1-\lvert R^D \rvert)$	Areas other than the fence region R_i^F.
$\sum_{k=K_1}^{K_1+K_2-1} \eta_k \Gamma_k(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})$	$k \in \{\lvert R^D \rvert + \lvert R^F \rvert + 1, \ldots, \lvert R^D \rvert + \lvert R^F \rvert + \lvert R^G \rvert\}$	Instances assigned to guide region $R_i^G (i = k-1-\lvert R^D \rvert - \lvert R^F \rvert)$	-

Figure 3: The instance subsets and artificial placement blockages for different objective terms in Eq. (3).

are categorized into types such as LUT, FF, DSP, and BRAM, with each type of instance having different placeable areas) as well as in fence region-aware placement problems [19]. It should be noted that the formulation in [19] assumes that instances assigned to different electrostatic systems have orthogonal placeable areas, which cannot handle the default regions and guide regions described in Section 2.1.

2.3 Unconstrained Optimization Algorithms

In global placement problems, we typically transform constrained optimization problems into unconstrained ones and solve them through multiple iterations [12, 13, 30, 32, 53] as follows:

$$\min_x f(x). \tag{2}$$

Let $g^{(k)} = \nabla f^{(k)}(x)$ and $H^{(k)} = \nabla^2 f^{(k)}(x)$ denote the gradient and Hessian matrix at the k-th iteration, respectively. Common algorithms for solving unconstrained optimization problems include the gradient method, Newton's method, and the Quasi-Newton method [41, 42, 50].

Gradient Method: $x^{(k+1)} = x^{(k)} - \alpha^{(k)} g^{(k)}$, where $\alpha^{(k)}$ is the step size. Common gradient methods include the steepest descent, the momentum method, and Nesterov's accelerated gradient [40], among others. Most previous works [12, 19–22, 30, 32] are based on gradient methods that fundamentally rely solely on the first-order derivative information of the function to select the descent direction.

Gradient methods are easy to implement, have low computational complexity, and minimal memory requirements. However, they can be sensitive to the choice of the initial solution and learning rate, potentially getting stuck in local minima or saddle points in non-convex optimization landscapes, such as those found in electrostatics-based placement algorithms. Especially in hybrid region placement problems, the non-convexity of the optimization landscape is more pronounced, making it more challenging for gradient methods to converge (as demonstrated by the experiments in Section 4.2).

Newton's Method: $x^{(k+1)} = x^{(k)} - [H^{(k)}]^{-1} g^{(k)}$. Newton's method constructs an iterative framework using second-order gradient information. The derivation of Newton's method involves neglecting the higher-order terms of the second-order Taylor expansion and setting the right side of the expansion as a function concerning the descent direction $d^{(k)}$, thereby obtaining $d^{(k)} = -[H^{(k)}]^{-1} g^{(k)}$ when $H^{(k)}$ is non-singular[2]. Note that the step size is always 1 here, meaning that the selection of the step size does not need to be considered additionally. Because it utilizes more information, Newton's method can outperform gradient methods in actual performance. However, it also imposes higher demands on the function $f(x)$.

[2]We omit the derivation details here.

Quasi-Newton Method: $x^{(k+1)} = x^{(k)} - \alpha^{(k)} d^{(k)}$. Newton's method has achieved good results both theoretically and in practice. However, for large-scale problems, calculating the Hessian matrix is either particularly expensive or difficult to obtain, and even if obtained, solving a large-scale system of linear equations is necessary.

To address the drawbacks of Newton's method, quasi-Newton methods construct approximations to the Hessian matrix (denoted as $B^{(k)}$; $d^{(k)} = [B^{(k)}]^{-1} g^{(k)}$) or approximations to its inverse (denoted as $T^{(k)}$; $d^{(k)} = T^{(k)} g^{(k)}$) to perform Newton-like iterations. They can generate an approximate matrix at each step at a lower computational cost, and the iterative sequence produced by using the approximate matrix instead of the Hessian matrix still maintains the property of superlinear convergence. Quasi-Newton methods generally require a line search to determine an appropriate step size $\alpha^{(k)}$.

3 ALGORITHMS

In this section, we further detail our proposed algorithm.

3.1 Unified Multi-Electrostatics Formulation for Hybrid Region Constraints

3.1.1 Overview. Let the default regions, fence regions, and guide regions be represented by the sets R^D, R^F, and R^G, respectively. We construct $K = 1 + |R^D| + |R^F| + |R^G|$ instance subsets (denoted as $V^{(0)}, V^{(1)}, \ldots, V^{(K-1)}$), and denote the locations of the instances belonging to the k-th instance subset as $(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})$. The construction method for the instance subsets will be provided in subsequent sections. We construct $K_1 = 1 + |R^D| + |R^F|$ electrostatic systems and $K_2 = |R^G|$ guide region penalty terms to model the hybrid region constraints. We extend the first-order density penalty term in Eq. (1) by leveraging a modified Lagrangian formulation [53],

$$\min_{\boldsymbol{x},\boldsymbol{y}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = \widetilde{W}(\boldsymbol{x}, \boldsymbol{y}) + \sum_{k=0}^{K_1-1} \lambda_k \mathcal{D}_k + \sum_{k=K_1}^{K_1+K_2-1} \eta_k \Gamma_k(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}), \tag{3}$$

$$\mathcal{D}_k = \Phi_k(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}) + \frac{1}{2} \mathcal{C}_k \Phi_k^2(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}), \tag{4}$$

where Γ_k and η_k are the guide region penalty term and the corresponding guide multiplier, respectively. We adopt a normalized subgradient-based method [19] to initialize and update the density multiplier λ_k and the density multiplier preconditioner $\mathcal{C}_k$. The initialization and updating of η_k and Γ_k are detailed in Section 3.1.5.

3.1.2 Initial Solution. The initial solution for the optimization problem is as follows:

- For instances not subject to any region constraints, their positions are normally distributed with a mean at the center

of the chip layout and standard deviations of 2.5% of the instance's own width and height in the horizontal and vertical directions, respectively.

- For instances subject to a constraint by a certain region, their positions are normally distributed with a mean at the center of the minimum bounding box of the rectangular sub-boxes within that region, and standard deviations of 2.5% of the instance's own width and height in the horizontal and vertical directions, respectively.

3.1.3 Shared Electrostatics Model. As mentioned in Section 3.1.1, we construct $K_1 = 1 + |R^D| + |R^F|$ electrostatic systems. The corresponding instance subsets for these K_1 electrostatic systems, i.e., $V^{(0)}, V^{(1)}, \ldots, V^{(K_1-1)}$, and the artificial placement blockages are depicted in the first three content rows of Figure 3. Note that these K_1 instance subsets are not mutually exclusive; an instance can belong to multiple instance subsets, and different instance subsets can share the same instance.

Figure 2 illustrates a schematic diagram of three electrostatic systems with one default region and one fence region. When the optimization algorithm reduces the potential energy of each electrostatic system to a sufficiently low level, we achieve the following placement effects:

- For instances not constrained by any fence region or default region, they are ultimately placed outside the fence region areas. Note that they can be placed within the areas where the default regions are located.
- For instances constrained by the fence region, they are ultimately placed inside the fence region.
- For instances constrained by the default region, they are ultimately placed inside the default region.
- In the area of the default region, the overlap between instances constrained by this default region and those not constrained by this default region can be reduced.

The first three correspond to the effects of artificial placement blockages in each electrostatic system. The last item is because instances assigned to any default region are also necessarily within the instance subset $V^{(0)}$. This indicates that our proposed algorithm can simultaneously optimize the degree of overlap between them in the 0-th electrostatic system, thereby allowing instances not constrained by any default region to be placed within the areas where the default regions are situated without overlapping with instances assigned to any default region.

3.1.4 Binary-Lifting-based Region Pruning Algorithm. The area within the Neumann boundary of an electrostatic system is divided into a two-dimensional bin grid. The electric field on each bin is calculated in backward propagation via the spectral method, for which the grid size is generally a power of two [32].

The boundary is intuitively located at the boundary of the chip layout [19]. However, for the electrostatic systems related to regions, i.e., the 1-st to $(K_1 - 1)$-th electrostatic systems in Eq. (3), there are two drawbacks to this method. 1) Firstly, there is a waste of footprint. Let the area of the chip layout be A. The space complexity of this method is $O(A \times (|R^D| + |R^F|))$, which can easily lead to memory overflow when there are many regions. 2) Secondly, there is a waste of runtime. Affected by artificial placement blockages, the instance usually does not deviate too far from the region, hence the electric field at locations that are relatively distant from the region is essentially not utilized during the backpropagation process.

Algorithm 1 Binary-Lifting-based Region Pruning Algorithm for Φ_0

1: **Input:** Layout boundary (xl, yl, xh, yh), target density ρ, number of movable instance areas N_m, and total movable instance area A_m.
2: **Output:** $(xl^{(0)}, yl^{(0)}, xh^{(0)}, yh^{(0)})$, $(n_x^{(0)}, n_y^{(0)})$
3: $xl^{(0)}, yl^{(0)}, xh^{(0)}, yh^{(0)} \leftarrow xl, yl, xh, yh$
4: $w, h \leftarrow xh - xl, yh - yl$
5: $\hat{a}_b \leftarrow \frac{A_m}{N_m \cdot \rho}$ ▷ $\hat{a}_b$ *denotes ideal bin area*
6: $\hat{N}_b \leftarrow \max(128, \lfloor \frac{w \times h}{\hat{a}_b} \rfloor)$ ▷ $\hat{N}_b$ *denotes ideal bin number*
7: $r \leftarrow \frac{h}{w}$ **if** $h > w$ **else** $\frac{w}{h}$ ▷ r *denotes the aspect ratio*
8: $r \leftarrow 2^{\lfloor \log_2 r \rceil}$
9: $n \leftarrow 2$
10: **while** $n \leq 4096$ **do**
11: $n' \leftarrow rn^2$
12: **if** $n' > \frac{\hat{N}_b}{2}$ **and** $n' \leq 2\hat{N}_b$ **then**
13: **break**
14: **end if**
15: $n \leftarrow n \times 2$
16: **end while**
17: **if** $h > w$ **then**
18: $n_x^{(0)}, n_y^{(0)} \leftarrow n, rn$
19: **else**
20: $n_x^{(0)}, n_y^{(0)} \leftarrow rn, n$
21: **end if**

We propose an innovative binary-lifting-based region pruning algorithm to address these issues, as illustrated in Algorithm 1 and Algorithm 2[3]. Denote $(xl^{(k)}, yl^{(k)}, xh^{(k)}, yh^{(k)})$ and $(n_x^{(k)}, n_y^{(k)})$ as the boundary and the grid size of the electrostatic system associated with Φ_k $(k = 0, 1, \ldots, K_1 - 1)$ [4], respectively. Algorithm 1 solves for the boundary and grid size of Φ_0. Since instances in Φ_0 that are not subject to any region constraints can be placed anywhere on the layout, the boundary of Φ_0 should be the chip layout boundary (line 3). The bin area should neither be too large nor too small. Ideally, we expect that on average, one movable instance is contained within a bin given the target density ρ, which implies $\hat{N}_b \cdot \rho = \frac{A_m}{N_m}$, leading to line 5. From this we can further determine the ideal number of bins (line 6). And finally we approximate the grid size on the horizontal and vertical coordinates based on the aspect ratio of the boundary, aiming to make the bin number as close as possible to the ideal bin number (lines 7-21).

Algorithm 2 solves for the boundary and grid size of Φ_k $(k = 1, 2, \ldots, K_1 - 1)$. The bin size and bin texture of Φ_k are consistent with Φ_0 (line 3) (see Figure 2). This is to ensure consistency in the calculation of contributions to density and overflow when the same instance is in different electrostatic systems. The boundary width $w_x^{(k)}$ of Φ_k is at least twice the size of the smallest bounding box of the rectangles (lines 4-10), representing a trade-off between the range of instance movement and spatial complexity. Finally, the

[3]We only discuss Algorithm 2 on the x direction, and that on the y direction is analogous.
[4]For brevity, we abbreviate the electrostatic system associated with Φ_k as Φ_k in the following context.

Algorithm 2 Binary-Lifting-based Region Pruning Algorithm for Φ_k $(k = 1, \ldots, K_1 - 1)$ (on x direction)

1: **Input:** Layout boundary (xl, yl, xh, yh), $(n_x^{(0)}, n_y^{(0)})$, and the rectangular sub-boxes coordinates of the corresponding region $\{(xl_i^{(k)}, yl_i^{(k)}, xh_i^{(k)}, yh_i^{(k)}) | 0 \le i < N^{(k)}\}$
2: **Output:** $(xl^{(k)}, yl^{(k)}, xh^{(k)}, yh^{(k)})$, $(n_x^{(k)}, n_y^{(k)})$
3: $s_x \leftarrow \frac{xh-xl}{n_x^{(0)}}$ ▷ Φ_k *has the same bin size* s_x *as* Φ_0.
4: $xl_{mbb}, xh_{mbb} \leftarrow \min_{0 \le i < N^{(k)}} xl_i^{(k)}, \max_{0 \le i < N^{(k)}} xh_i^{(k)}$
5: $n \leftarrow 128$
6: **while** $n \cdot s_x \le xh_{mbb} - xl_{mbb}$ **do**
7: $\quad n \leftarrow n \times 2$
8: **end while**
9: $n \leftarrow n \times 2$
10: $w_x^{(k)} \leftarrow \min(xh - xl, n \cdot s_x)$ ▷ $w_x^{(k)}$ *is the boundary width of* Φ_k.
11: $n_x^{(k)} \leftarrow \frac{w_x^{(k)}}{s_x}$ ▷ *grid size of* Φ_k.
12: $ix_c \leftarrow \lfloor \frac{(xh_{mbb}+xl_{mbb})/2-xl}{s_x} \rfloor$
13: $ix_b \leftarrow \max(0, ix_c - n_x^{(k)}/2)$
14: $xl^{(k)}, xh^{(k)} \leftarrow xl + ix_b \cdot s_x, xl + (ix_b + n_x^{(k)}) \cdot s_x$
15: **if** $xh^{(k)} > xh$ **then**
16: $\quad xl^{(k)}, xh^{(k)} \leftarrow xl^{(k)} - (xh^{(k)} - xh), xh$
17: **end if**

range of the boundary is determined based on the boundary width and length of Φ_k (lines 11-17).

3.1.5 Wirelength-Prioritized Penalty Method for Guide Regions. As stated in Section 2.1, wirelength constraints take precedence over guide region constraints. To prevent wirelength from becoming excessively high due to overly restrictive guide regions, we propose a wirelength-prioritized penalty method for guide regions. Each instance i that is subject to a guide region constraint has a *target rectangular region* B_i, and the guide region penalty is calculated based on B_i. B_i is initialized to the minimum bounding box of the corresponding guide rectangular sub-boxes. Let N_{max} be the maximum number of rectangular sub-boxes in all guide regions, and let the stop overflow for Φ_0 be o_{stop} (e.g., 0.1). We set N_{max} thresholds $\delta_i = 1.0 - (i+1) \times \frac{1.0-o_{stop}}{N_{max}+1}$, for $i = 1, \ldots, N_{max}$. When the overflow of Φ_0 reaches a threshold, we re-evaluate the trade-off between the wirelength objective and the guide region objective and update B_i. We leverage the median region algorithm [43] to find an optimal region ξ_i for instance i, and consider the following cases:

(1) If there is more than one rectangular sub-box within the current B_i, we discard the one that is farthest from ξ_i in Manhattan distance, and use the minimum bounding box of the remaining rectangular sub-boxes as the new B_i.
(2) If there is exactly one rectangular sub-box within the current B_i, and B_i intersects with ξ_i, then we keep B_i unchanged.
(3) Otherwise, we cancel B_i, and instance i will not be penalized for the guide region in the subsequent iterations.

Given B_i, we define a barrier function $\Gamma_i(\cdot)$ for instance i as follows:

$$\Gamma_i(x_i, y_i) = g(x_i, B_i^{xl}, B_i^{xh}, xl, xh) + g(y_i, B_i^{yl}, B_i^{yh}, yl, yh), \quad (5)$$

where $g(x, B^l, B^r, l, r)$ is defined as

$$g(x, B^l, B^r, l, r) = \begin{cases} 0 & \text{if } B^l \le x \le B^r, \\ \zeta\left(\frac{x-l}{B^l-l}; \frac{1}{2}\right), & \text{if } x < B^l \\ \zeta\left(\frac{r-x}{r-B^r}; \frac{1}{2}\right), & \text{if } B^r < x \end{cases} \quad (6)$$

and $\zeta(t; s)$ is a second-order differentiable function that is monotonically decreasing on $[0, 1]$ with the steepest slope at $t = s$.

$$\zeta(t; s) = \begin{cases} 1 - \frac{1}{s}t^2, & \text{if } 0 \le t \le s \\ \frac{1}{1-s}(t-1)^2, & \text{if } s < t \le 1 \end{cases} \quad (7)$$

The Γ_k in Eq. (3) is the sum of all instances' $\Gamma_i(\cdot)$ within $V^{(k)}$. We use the following method to initialize η_k:

$$\eta_k = \frac{\|\nabla \widetilde{W}\|_1}{\|\nabla \Gamma_k'\|_1} \frac{\alpha}{\mu^\beta}, \quad (8)$$

where $\alpha = 10^{-4}$, $\mu = 2$, and $\beta = 10$. Every 100 iterations, η_k is multiplied by μ. β is an estimate of the number of times η_k is updated, and α controls the ratio of the wirelength gradient to the guide region penalty gradient upon convergence. $\Gamma_k' = \sum_{i \in V^{(k)}} \Gamma_i'(x_i, y_i)$ is used to estimate the gradient of the guide region penalty during the iteration process as follows:

$$\Gamma_i'(x_i, y_i) = g'(x_i, B_i^{xl}, B_i^{xh}, xl, xh) + g'(y_i, B_i^{yl}, B_i^{yh}, yl, yh) \quad (9)$$

where $g'(x, B^l, B^r, l, r)$ is defined as

$$g'(x, B^l, B^r, l, r) = \begin{cases} \zeta\left(\frac{x-l}{m-l}; \frac{B^l-l}{m-l}\right) & \text{if } x \le m = \frac{B^l+B^r}{2} \\ \zeta\left(\frac{h-x}{h-m}; \frac{h-B^h}{h-m}\right) & \text{if } m < x \end{cases} \quad (10)$$

3.2 Preconditioning for Hybrid Region Constraints

Inspired by the Jacobi preconditioner method [32], we propose a hybrid region-aware precondition technique to address the gradient deviation issue. We have found that gradient deviation can occur in any electrostatic system, thus we apply divergence-aware gradient preconditioning to all instances. The second-order derivative of the wirelength objective $\widetilde{W}$ is approximated by the number of pins of the instance [32]; the second-order derivative of the augmented-Lagrangian-based density penalty $\mathcal{D}_k$ can be approximated using the instance area [39]; and the second-order derivative of the guide region penalty Γ_k can be directly computed. Therefore, the preconditioner $\mathcal{P} \in \mathbb{R}^{2 \times |v|}$ is given as follows [5],

$$\mathcal{P}_{0,i} = \max\left\{1, [\frac{\partial^2 \mathcal{L}}{\partial x_i^2}]^{-1}\right\} \quad (11)$$

$$= \max\left\{1, [\frac{\partial^2 \widetilde{W}}{\partial x_i^2} + \sum_{k=0}^{K_1-1} \lambda_k \frac{\partial^2 \mathcal{D}_k}{\partial x_i^2} + \sum_{k=K_1}^{K_1+K_2-1} \eta_k \frac{\partial^2 \Gamma_k}{\partial x_i^2}]^{-1}\right\} \quad (12)$$

$$= \max\left\{1, [\#\text{pins}(v_i) + \tau \sum_{k=0}^{K_1-1} \lambda_k \mathbb{I}_k(v_i) \text{area}(v_i) + \sum_{k=K_1}^{K_1+K_2-1} \eta_k \frac{\partial^2 \Gamma_k}{\partial x_i^2}]^{-1}\right\} \quad (13)$$

$\mathbb{I}_k(v_i)$ is an indicator function that equals 1 if and only if $v_i \in V^{(k)}$. Otherwise, it equals 0. The setting of τ is the same as in [19], and it is intended to slow down the movement of large-sized instances. Then we give the preconditioned gradient $\nabla \hat{\mathcal{L}} = \nabla \mathcal{L} \odot \mathcal{P}$ to the optimizer.

[5] We only discuss the gradient on x direction and that on y direction is the same.

Algorithm 3 A Modified Nesterov's Accelerated LBFGS Algorithm

1: **Input:** major solution $u^{(k)}$, reference solution $v^{(k)}$, optimization parameter $a^{(k)}$, LBFGS memory length m.
2: **Output:** $u^{(k+1)}$, $v^{(k+1)}$, $a^{(k+1)}$.
3: $g^{(k)}, g^{(k-1)} \leftarrow \nabla f(v^{(k)}), \nabla f(v^{(k-1)})$
4: $s_{k-1} \leftarrow v^{(k)} - v^{(k-1)}$ ▷ solution difference
5: $y_{k-1} \leftarrow g^{(k)} - g^{(k-1)}$ ▷ gradient difference
6: $\rho_{k-1} \leftarrow \frac{1}{y_{k-1}^T s_{k-1}}$
7: store $s_{k-1}, y_{k-1}, \rho_{k-1}$
8: **if** $k > m$ **then**
9: remove $s_{k-m-1}, y_{k-m-1}, \rho_{k-m-1}$ from memory
10: **end if**
11: $\hat{T}_{k-m} \leftarrow \frac{y_{k-1}^T s_{k-1}}{y_{k-1}^T y_{k-1}} I$ ▷ approximation of $[H^{(k-m)}]^{-1}$
12: $d^{(k)} \leftarrow \texttt{LBFGS}(g^{(k)}, \hat{T}_{k-m})$ ▷ descent direction
13: $\alpha_0 \leftarrow 1$ ▷ initial step size
14: $\alpha^{(k+1)} \leftarrow \textsc{LineSearch}(v^{(k)}, d^{(k)}$, starts from $\alpha_0)$
15: $u^{(k+1)} \leftarrow v^{(k)} - a^{(k)} d^{(k)}$
16: $a^{(k+1)} \leftarrow \left(1 + \sqrt{4a^{(k)} + 1}\right)/2$
17: $v^{(k+1)} \leftarrow u^{(k+1)} + \frac{a^{(k+1)}-1}{a^{(k+1)}}(u^{(k+1)} - u^{(k)})$
18: **return** $u^{(k+1)}$, $v^{(k+1)}$, $a^{(k+1)}$
19: /* LBFGS Dual Loop Recursive Algorithm */
20: **function** LBFGS($g^{(k)}, \hat{T}_{k-m}$)
21: initialize $q \leftarrow g^{(k)}$
22: **for** $i = k-1, k-2, \ldots, k-m$ **do**
23: $\alpha_i \leftarrow \rho_i s_i^T q$
24: $q \leftarrow q - \alpha_i y_i$
25: **end for**
26: $r \leftarrow \hat{T}_{k-m} q$
27: **for** $i = k-m, k-m+1, \ldots, k-1$ **do**
28: $\beta_i \leftarrow \rho_i y_i^T r$
29: $r \leftarrow r + (\alpha_i - \beta_i) s_i$
30: **end for**
31: **return** r ▷ approximation of $[H^{(k)}]^{-1} g^{(k)}$
32: **end function**

3.3 Robust Optimization via a Modified Nesterov's Accelerated LBFGS Algorithm

Multi-electrostatics-based placement with hybrid region constraints needs to handle multiple optimization objectives simultaneously, which can lead to local optimal solutions. First-order optimization methods often require randomization techniques to escape local optima or saddle points to achieve robustness [19], but this can somehow degrade the quality of the algorithm's outcomes.

In our framework, we propose a modified Nesterov's accelerated Limited Memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm along with Wolfe line search [15], as depicted in Algorithm 3. LBFGS is a type of quasi-Newton method, which approximates the Hessian matrix without computing it fully. Unlike the full-memory BFGS method, LBFGS does not store the entire history of the Hessian matrix approximations. Instead, it maintains only a limited number (m in Algorithm 3) of the most recent m vectors and scalars (solution difference $s_{(\cdot)}$, gradient difference $y_{(\cdot)}$, and $\rho_{(\cdot)}$ in Algorithm 3) that represent the approximation (lines 4-10). This makes LBFGS more memory-efficient.

Table 1: Statistics of `ISPD2015-FR` benchmarks [5] and its variant `ISPD2015-HR`.

Design	#Cells	#Nets	ISPD2015-FR	ISPD2015-HR		
			#Fence	#Fence	#Default	#Guide
mgc_des_perf_a	108K	115K	4	1	2	1
mgc_des_perf_b	113K	113K	12	4	4	4
mgc_edit_dist_a	127K	134K	1	0	0	1
mgc_matrix_mult_b	146K	152K	3	1	1	1
mgc_matrix_mult_c	146K	152K	3	1	1	1
mgc_pci_bridge32_a	30K	34K	4	1	2	1
mgc_pci_bridge32_b	29K	33K	3	1	1	1
mgc_superblue11_a	926K	936K	4	0	4	0
mgc_superblue16_a	680K	697K	2	0	1	1

LBFGS uses a two-loop recursion to update the inverse Hessian approximation (line 20). We use the diagonal matrix $\gamma_k I$ to approximate the inverse Hessian matrix $[H^{(k-m)}]^{-1}$ at the $(k-m)$-th iteration [31] (line 11)[6], where

$$\gamma_k = \frac{y_{k-1}^T s_{k-1}}{y_{k-1}^T y_{k-1}} \tag{14}$$

The two loops generate an approximation to the descent direction, i.e., $[H^{(k)}]^{-1} g^{(k)}$, at the k-th iteration (lines 21-30). For the sake of brevity, we omit the derivation details here.

The time-space complexity of the LBFGS algorithm is $O(mn)$, mainly determined by the memory length m and the dimensionality of the problem n. Since m is usually much less than n [7], the LBFGS algorithm is very efficient for large-scale optimization problems.

Applying Nesterov's acceleration technique to the LBFGS algorithm (lines 15-17) results in the modified Nesterov's accelerated LBFGS algorithm. This algorithm not only leverages the memory efficiency of the LBFGS algorithm in dealing with large-scale problems but also enhances the convergence speed through Nesterov acceleration. It can be proven that our proposed algorithm has a quadratic convergence rate and can derive robustness through second-order information.

3.4 Hybrid-Region-aware Legalization

Hybrid region constraints present a complex challenge in the legalization process. For fence region constraints, the placeable area and the instance subset within the fence region are orthogonal. This means that several independent legalization sub-problems can be constructed each consisting of a placeable area, artificial placement blockages, and the corresponding instance subset for each fence region constraint. These can then be converted into standard legalization sub-problems [8, 19, 21, 54]. However, for default regions, the placeable areas of instances not subject to any region constraint and instances assigned to any default region overlap and influence each other. This makes it impossible to divide them into independent legalization sub-problems as with fence region constraints. The fact that default regions can also overlap adds further complexity to the legalization process.

To address this issue, we propose a two-stage hybrid region-aware legalization algorithm. In the first stage, we relax the default region constraint, treating instances constrained by the default region constraint as if they had no region constraints. In this scenario, only the fence region constraint remains a hard constraint.

[6] Although the multipliers in Eq. (3) will change during the iterative process, the approximation remains valid due to the small magnitude of changes in the most recent iterations.
[7] We empirically set m to be 3 in our placer.

Table 2: HPWL ($\times 10^5$), overflow ($\times 10^3$), and runtime (seconds) comparison on ISPD2015-FR and ISPD2015-HR.

Design	NTUplace4dr (8 threads) (on ISPD2015-FR)			DREAMPlace 3.0 (GPU) (on ISPD2015-FR)			MORPH (GPU) (on ISPD2015-FR)			MORPH (GPU) (on ISPD2015-HR)		
	HPWL	Overflow	RT	HPWL	Overflow	RT	HPWL	Overflow	RT	HPWL	Overflow	RT
mgc_des_perf_a	27.018	14.51	288.47	25.564	17.36	16.73	25.309	17.67	24.01	24.387	11.32	20.92
mgc_des_perf_b	22.237	3.47	309.12	20.821	1.11	39.83	19.231	0.91	41.06	18.677	0.52	31.24
mgc_edit_dist_a	53.754	157.73	265.80	47.114	108.94	12.92	46.875	109.16	15.90	47.428	88.85	4.36
mgc_matrix_mult_b	41.342	18.83	244.71	38.024	26.77	17.15	36.606	24.31	22.42	34.009	17.76	19.99
mgc_matrix_mult_c	39.128	17.54	302.19	36.789	22.72	16.82	35.027	19.35	23.13	33.939	16.88	19.86
mgc_pci_bridge32_a	6.942	5.05	60.22	5.937	3.48	18.04	5.381	2.92	20.68	5.182	2.60	16.96
mgc_pci_bridge32_b	8.444	1.20	44.72	8.765	3.82	14.24	7.756	4.21	16.14	7.007	2.15	12.43
mgc_superblue11_a	432.685	216.37	11462.82	402.197	223.14	39.23	390.249	178.28	60.68	395.421	182.44	98.08
mgc_superblue16_a	341.991	1027.67	3815.06	300.326	309.16	39.89	284.606	274.92	28.73	280.084	266.85	31.22
Ratio	1.143	1.24	15.21	1.056	1.10	0.84	1.000	1.00	1.00	0.967	0.76	0.81

We decompose the legalization problem into $1 + |R^F|$ independent legalization sub-problems for solving, following the suggestion of [19].

In the second stage, we extract the currently illegal instances. These instances must have violated the default region constraints because we relaxed the default region constraints in the first stage. We treat the instances that are currently legally placed as placement blockages, the instances violating the default region constraints as legalization subjects, and the default region as a fence region. Similarly to the first stage, we construct $|R^D|$ legalization sub-problems and solve them using the greedy legalizer [10] and the Abacus legalizer [49].

4 EXPERIMENTAL RESULTS

We implement our proposed algorithm in C++ and achieve GPU acceleration. We conduct experiments on a Linux system equipped with two Intel(R) Xeon(R) Platinum 8358 CPUs (2.60GHz, 32 cores), 1024GB of RAM, and one NVIDIA A800 GPU. All experimental data are run on our server. We use Cadence Innovus 2022 [1] to measure the HPWL [8] and the global routing overflow of the placement results. All the ratios in the experimental tables are geometric means.

We conduct experiments on the ISPD 2015 benchmark suite [5] (denoted as ISPD2015-FR) to evaluate our placer's performance under fence region constraints only. To test the performance of our proposed algorithm under hybrid region constraints, we modify the ISPD 2015 benchmarks by converting some of the fence regions into default and guide regions, denoted as ISPD2015-HR. The statistics for the two benchmark suites are shown in Table 1. We compare our results with those of state-of-the-art region-aware placers, NTUplace4dr [21] and DREAMPlace 3.0 [19], using the same target utilization limit. It should be noted that both DREAMPlace 3.0 and our placer focus on wirelength optimization without explicitly enhancing routability, which justifies a comparison of HPWL between the two. However, NTUplace4dr considers detailed routability and design rule checks, which may make a direct HPWL comparison less informative. Therefore, we also present the global routing overflow reported by Innovus as an indicator of routability. As our placer does not explicitly optimize routability, we mainly compare the wirelength metric.

[8]The HPWL values reported in Innovus differ from those in [19] reported by NCTUgr [14], but they still reflect the relative magnitude of HPWL.

4.1 HPWL and Overflow Evaluation

The second to fourth columns of Table 2 show the comparison results with NTUplace4dr [21] and DREAMPlace 3.0 [19] on ISPD2015-FR with fence region constraints only. We can observe that our placer consistently achieves better HPWL than NTUplace4dr and DREAMPlace 3.0 across all designs. Compared to NTUplace4dr, our placer has improved HPWL by 14.3% and overflow by 24% with 15.21× speedup. Our placer can achieve an average improvement of 5.6% HPWL and a 10% reduction in overflow compared to DREAMPlace 3.0. Additionally, our algorithm is slightly 16% slower than DREAMPlace 3.0, due to the larger computational workloads required by the quasi-Newton's method. This indicates that our proposed algorithm can significantly enhance solution quality without a substantial time overhead.

The last column of Table 2 shows the HPWL, overflow, and runtime of our placer on ISPD2015-HR with hybrid region constraints. It should be noted that there is no existing work that considers both default regions and guide regions; NTUplace4dr and DREAMPlace 3.0 treat all regions as fence regions, and the comparison results here serve as a reference. Experimental results show that our proposed algorithm can stably converge to better results under hybrid region constraints compared to NTUplace4dr and DREAMPlace 3.0. Compared to NTUplace4dr, our placer achieves an 18.2% reduction in HPWL and a 64% decrease in overflow with 18.7× speedup. Our placer also delivers a 9.2% enhancement in HPWL and a 45% reduction in overflow, with a runtime similar to that of DREAMPlace 3.0.

4.2 Robustness Evaluation

We evaluate the HPWL, overflow, number of iterations, and runtime on the ISPD2015-HR and ISPD2015-FR benchmarks. We validate the effectiveness of our proposed techniques by replacing specific optimization techniques as follows:

- The preconditioning algorithm described in Section 3.2 is replaced with the one from [19], denoted as MORPH-precond;
- The optimization algorithm in Section 3.3 is replaced with the first-order gradient method used in [30], denoted as MORPH-GD;
- These two techniques are combined to form a baseline, denoted as MORPH-precond-GD.

Table 3 and Table 4 respectively present comparative experimental results on ISPD2015-HR and ISPD2015-FR. Our proposed Nesterov's accelerated LBFGS optimization algorithm is found to

Table 3: HPWL ($\times 10^5$), overflow ($\times 10^3$), number of iterations, and runtime (seconds) comparison on ISPD2015-HR.

Design	MORPH-precond-GD (GPU)				MORPH-precond (GPU)				MORPH-GD (GPU)				MORPH (GPU)			
	HPWL	Overflow	#Iters	RT	HPWL	Overflow	#Iters	RT	HPWL	Overflow	#Iters	RT	HPWL	Overflow	#Iters	RT
mgc_des_perf_a-HR	24.681	12.18	1500†	18.13	24.097	11.26	1500†	33.25	*	*	1500†	17.71	24.387	11.32	1009	20.92
mgc_des_perf_b-HR	*	*	1500†	26.19	18.393	0.47	940	34.00	18.137	0.44	941	17.22	18.677	0.52	944	31.24
mgc_edit_dist_a-HR	47.336	87.25	602	2.44	47.428	88.85	596	4.36	47.336	87.25	602	2.41	47.428	88.85	596	4.36
mgc_matrix_mult_b-HR	35.872	24.97	1500†	15.79	34.289	18.66	1088	20.23	41.278	45.02	1500†	16.07	34.009	17.76	1082	19.99
mgc_matrix_mult_c-HR	*	*	1500†	16.72	33.967	16.01	1103	22.80	34.131	17.26	1500†	15.44	33.939	16.88	1093	19.86
mgc_pci_bridge32_a-HR	5.501	2.64	1500†	16.53	*	*	1500†	35.63	5.087	2.28	1074	11.75	5.182	2.60	965	16.96
mgc_pci_bridge32_b-HR	7.105	2.22	1500†	12.51	7.032	2.31	1500†	22.58	7.041	2.24	848	8.61	7.007	2.15	847	12.43
mgc_superblue11_a-HR	482.200	1074.49	1500†	82.18	393.247	185.48	1140	97.58	*	*	1500†	78.57	395.421	182.44	1136	98.08
mgc_superblue16_a-HR	*	*	1500†	23.93	280.165	269.02	993	27.56	*	*	1500†	25.66	280.084	266.85	1018	31.22
Ratio	1.058	1.45	1.43	0.82	0.998	1.00	1.17	1.24	1.026	1.12	1.22	0.72	1.000	1.00	1.00	1.00

† Stopped after reaching their maximum iteration. The same for Table 4.
∗ Diverged. The same for Table 4.

Table 4: HPWL ($\times 10^5$), overflow ($\times 10^3$), number of iterations, and runtime (seconds) comparison on ISPD2015-FR.

Deisgn	MORPH-precond-GD (GPU)				MORPH-precond (GPU)				MORPH-GD (GPU)				MORPH (GPU)			
	HPWL	Overflow	#Iters	RT	HPWL	Overflow	#Iters	RT	HPWL	Overflow	#Iters	RT	HPWL	Overflow	#Iters	RT
mgc_des_perf_a	28.199	31.36	1500†	28.91	25.353	17.15	969	22.13	40.913	73.83	1500†	28.91	25.309	17.67	988	24.01
mgc_des_perf_b	*	*	1500†	38.01	19.190	1.12	990	38.51	*	*	1500†	38.01	19.231	0.91	1000	41.06
mgc_edit_dist_a	48.673	110.29	1084	9.87	46.774	107.33	1500†	20.33	46.884	111.14	1084	9.87	46.875	109.16	1034	15.90
mgc_matrix_mult_b	54.654	113.24	1500†	36.06	37.421	29.14	1500†	38.05	66.285	218.31	1500†	36.06	36.606	24.31	1067	22.42
mgc_matrix_mult_c	*	*	1500†	34.17	40.691	39.34	1500†	39.71	43.854	52.16	1500†	34.17	35.027	19.35	1086	23.13
mgc_pci_bridge32_a	10.532	9.57	1500†	18.94	5.565	3.48	1500†	35.40	5.568	3.10	1500†	18.94	5.381	2.92	1006	20.68
mgc_pci_bridge32_b	9.087	4.15	860	14.87	8.323	4.39	824	13.49	7.683	3.84	860	14.87	7.756	4.21	830	16.14
mgc_superblue11_a	*	*	1500†	45.64	394.611	190.97	1132	59.71	528.591	1493.16	1500†	45.64	390.249	178.28	1140	60.68
mgc_superblue16_a	392.796	1738.36	1500†	22.50	284.325	281.72	995	27.49	*	*	1500†	22.50	284.606	274.92	993	28.73
Ratio	1.327	2.35	1.34	0.98	1.032	1.16	1.17	1.18	1.262	2.61	1.34	0.98	1.000	1.00	1.00	1.00

bring the greatest improvement to wirelength when comparing our placer with MORPH-GD, and MORPH-precond with MORPH-precond-GD on the two tables. MORPH-precond-GD and MORPH-GD, which employ the first-order gradient method, diverge on three designs on the ISPD2015-HR benchmark. Compared to MORPH-precond-GD, the number of diverged designs for MORPH-precond is reduced by two, and our placer experiences no divergence. Similar observations are made on ISPD2015-FR. This suggests that the gradient method does not effectively handle complex hybrid region constraints. Instead, the LBFGS algorithm's limited-memory approximation to the Hessian matrix makes it an efficient and robust choice for solving large-scale optimization problems, balancing the quality of solutions with the computational resources required.

Comparative experiments also demonstrate the effectiveness of our proposed preconditioning technique. On the ISPD2015-HR benchmark, MORPH-precond reaches the maximum iteration limit on three designs. Our proposed preconditioning technique enables convergence for one design that MORPH-precond diverges on. Moreover, on designs where MORPH-precond does not diverge, our placer achieves similar results in terms of HPWL and overflow, while also reducing the number of iterations by 17% and the runtime by 24%. On ISPD2015-FR, our proposed preconditioning technique optimizes wirelength, overflow, number of iterations, and runtime by 3.2%, 16%, 17%, and 18%, respectively, compared to MORPH-precond. This indicates that our proposed preconditioner stabilizes global placement iterations and enables better solution quality upon convergence.

5 CONCLUSION

In this work, we propose MORPH, an innovative ASIC placer specifically designed to manage hybrid region constraints with enhanced robustness and efficiency. We propose a shared electrostatics model and a binary-lifting-based region pruning algorithm that integrate these constraints into a unified multi-electrostatic formulation. Additionally, the application of a wirelength-prioritized penalty method for guide regions ensures that wirelength constraints are not unduly compromised by restrictive guide regions. Our Nesterov's accelerated LBFGS algorithm provides a robust optimization strategy, effectively navigating the complex landscape of multiple optimization objectives. This optimization strategy, when combined with our preconditioning techniques, not only improves solution quality but also enhances the stability of the placement process. The incorporation of second-order information into our nonlinear placement backbone, along with a hybrid region-aware legalization algorithm, allows for better management of convergence issues. Experimental results demonstrate that on the ISPD 2015 benchmarks, we achieve a 5.6-14.3% HPWL improvement and a 10-24% overflow reduction compared to previous state-of-the-art region-aware placers. Further ablation experiments indicate that our proposed optimization techniques can achieve over 30% improvement in HPWL and more than 2× reduction in overflow compared to the optimization techniques in previous works with more stable convergence.

ACKNOWLEDGMENTS

This work was supported in part by the Natural Science Foundation of Beijing, China (Grant No. Z230002), and National Science Foundation of China (Grant No. T2293700, T2293701), and the 111 Project (B18001).

REFERENCES

[1] [n. d.]. Cadence Innovus Implementation System. http://www.cadence.com.
[2] [n. d.]. LEF/DEF Language Reference. http://www.ispd.cc/contests/14/web/doc/lefdefref.pdf.
[3] Ziad Abuowaimer, Dani Maarouf, Timothy Martin, Jeremy Foxcroft, Gary Grewal, Shawki Areibi, and Anthony Vannelli. 2018. GPlace3. 0: Routability-driven analytic placer for UltraScale FPGA architectures. *ACM TODAES* 23, 5 (2018), 1–33.

[4] Ismail Bustany, Grigor Gasparyan, Amit Gupta, Andrew B Kahng, Meghraj Kalase, Wuxi Li, and Bodhisatta Pramanik. 2023. The 2023 MLCAD FPGA Macro Placement Benchmark Design Suite and Contest Results. In *Proc. MLCAD*.

[5] Ismail S. Bustany, David Chinnery, Joseph R. Shinnerl, and Vladimir Yutsis. 2015. ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement. In *Proc. ISPD*. 157–164.

[6] Gengjie Chen, Chak-Wa Pui, Wing-Kai Chow, Ka-Chun Lam, Jian Kuang, Evangeline FY Young, and Bei Yu. 2018. RippleFPGA: Routability-driven simultaneous packing and placement for modern FPGAs. *IEEE TCAD* 37, 10 (2018), 2022–2035.

[7] Jianli Chen, Zhifeng Lin, Yun-Chih Kuo, Chau-Chin Huang, Yao-Wen Chang, Shih-Chun Chen, Chun-Han Chiang, and Sy-Yen Kuo. 2020. Clock-aware placement for large-scale heterogeneous FPGAs. *IEEE TCAD* 39, 12 (2020), 5042–5055.

[8] Jianli Chen, Ziran Zhu, Longkun Guo, Yu-Wei Tseng, and Yao-Wen Chang. 2021. Mixed-cell-height placement with drain-to-drain abutment and region constraints. *IEEE TCAD* 41, 4 (2021), 1103–1115.

[9] Sheng-Yen Chen and Yao-Wen Chang. 2015. Routing-architecture-aware analytical placement for heterogeneous FPGAs. In *Proc. DAC*. 27:1–27:6.

[10] Tung-Chieh Chen, Tien-Chang Hsu, Zhe-Wei Jiang, and Yao-Wen Chang. 2005. NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs. In *Proc. ISPD*. 236–238.

[11] Yu-Chen Chen, Sheng-Yen Chen, and Yao-Wen Chang. 2014. Efficient and effective packing and analytical placement for large-scale heterogeneous FPGAs. In *Proc. ICCAD*. 647–654.

[12] C. Cheng, A. B. Kahng, I. Kang, and L. Wang. 2019. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE TCAD* 38, 9 (2019), 1717–1730.

[13] Wing-Kai Chow, Jian Kuang, Peishan Tu, and Evangeline FY Young. 2017. Fence-aware detailed-routability driven placement. In *Proc. SLIP*. 1–7.

[14] Ke-Ren Dai, Wen-Hao Liu, and Yih-Lang Li. 2012. NCTU-GR: efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing. *IEEE TVLSI* 20, 3 (2012), 459–472.

[15] Yu-Hong Dai. 2002. Convergence properties of the BFGS algoritm. *SIAM Journal on Optimization* 13, 3 (2002), 693–701.

[16] Nima Karimpour Darav, Andrew Kennings, Aysa Fakheri Tabrizi, David Westwick, and Laleh Behjat. 2016. Eh? Placer: A high-performance modern technology-driven placer. *ACM TODAES* 21, 3 (2016), 1–27.

[17] Wenyi Feng, Jonathan Greene, Kristofer Vorwerk, Val Pevzner, and Arun Kundu. 2014. Rent's rule based FPGA packing for routability optimization. In *Proc. FPGA*. 31–34.

[18] Marcel Gort and Jason H. Anderson. 2012. Analytical placement for heterogeneous FPGAs. In *Proc. FPL*. 143–150.

[19] Jiaqi Gu, Zixuan Jiang, Yibo Lin, and David Z. Pan. 2020. DREAMPlace 3.0: Multi-Electrostatics Based Robust VLSI Placement with Region Constraints. In *Proc. ICCAD*.

[20] Meng-Kai Hsu, Yi-Fang Chen, Chau-Chin Huang, Sheng Chou, Tzu-Hen Lin, Tung-Chieh Chen, and Yao-Wen Chang. 2014. NTUplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs. *IEEE TCAD* 33, 12 (2014), 1914–1927.

[21] Chau-Chin Huang, Hsin-Ying Lee, Bo-Qiao Lin, Sheng-Wei Yang, Chin-Hao Chang, Szu-To Chen, Yao-Wen Chang, Tung-Chieh Chen, and Ismail Bustany. 2017. NTUplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints. *IEEE TCAD* 37, 3 (2017), 669–681.

[22] Andrew B Kahng and Qinke Wang. 2005. Implementation and extensibility of an analytic placer. *IEEE TCAD* 24, 5 (2005), 734–747.

[23] Wuxi Li, Mehrdad E Dehkordi, Stephen Yang, and David Z Pan. 2019. Simultaneous placement and clock tree construction for modern fpgas. In *Proc. FPGA*. 132–141.

[24] Wuxi Li, Shounak Dhar, and David Z. Pan. 2016. UTPlaceF: A Routability-driven FPGA Placer with Physical and Congestion aware Packing. In *Proc. ICCAD*. 66:1–66:7.

[25] Wuxi Li, Meng Li, Jiajun Wang, and David Z Pan. 2017. UTPlaceF 3.0: A parallelization framework for modern FPGA global placement. In *Proc. ICCAD*. IEEE, 922–928.

[26] Wuxi Li, Yibo Lin, Meng Li, Shounak Dhar, and David Z Pan. 2018. UTPlaceF 2.0: A high-performance clock-aware FPGA placement engine. *ACM TODAES* 23, 4 (2018), 1–23.

[27] Wuxi Li and David Z Pan. 2018. A new paradigm for FPGA placement without explicit packing. *IEEE TCAD* 38, 11 (2018), 2113–2126.

[28] Tingyuan Liang, Gengjie Chen, Jieru Zhao, Sharad Sinha, and Wei Zhang. 2021. Amf-placer: High-performance analytical mixed-size placer for fpga. In *Proc. ICCAD*. IEEE, 1–9.

[29] Tzu-Hen Lin, Pritha Banerjee, and Yao-Wen Chang. 2013. An efficient and effective analytical placer for FPGAs. In *Proc. DAC*. 10:1–10:6.

[30] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. 2019. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In *Proc. DAC*.

[31] Dong C Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical programming* 45, 1 (1989), 503–528.

[32] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2015. ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method. *ACM TODAES* 20, 2 (2015), 1–34.

[33] Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, et al. 2015. ePlace-MS: Electrostatics-Based Placement for Mixed-Size Circuits. *IEEE TCAD* 34, 5 (2015), 685–698.

[34] Jingwei Lu, Hao Zhuang, Ilgweon Kang, Pengwen Chen, and Chung-Kuan Cheng. 2016. ePlace-3D: electrostatics based placement for 3D-ICs. In *Proc. ISPD*. 11–18.

[35] Jing Mai, Yibai Meng, Zhixiong Di, and Yibo Lin. 2022. Multi-electrostatic FPGA placement considering SLICEL-SLICEM heterogeneity and clock feasibility. In *Proc. DAC*. 649–654.

[36] Jing Mai, Jiarui Wang, Zhixiong Di, and Yibo Lin. 2023. Multi-Electrostatic FPGA Placement Considering SLICEL-SLICEM Heterogeneity, Clock Feasibility, and Timing Optimization. *IEEE TCAD* (2023).

[37] Jing Mai, Jiarui Wang, Zhixiong Di, Guojie Luo, Yun Liang, and Yibo Lin. 2023. OpenPARF: an open-source placement and routing framework for large-scale heterogeneous FPGAs with deep learning toolkit. In *Proc. ASICON*. 1–4.

[38] Timothy Martin, Dani Maarouf, Ziad Abuowaimer, Abeer Alhyari, Gary Grewal, and Shawki Areibi. 2019. A flat timing-driven placement flow for modern FPGAs. In *Proc. DAC*. 1–6.

[39] Yibai Meng, Wuxi Li, Yibo Lin, and David Z Pan. 2021. elfPlace: Electrostatics-based placement for large-scale heterogeneous fpgas. *IEEE TCAD* 41, 1 (2021), 155–168.

[40] Yurii Nesterov. 1983. A method of solving a convex programming problem with convergence rate O (1/k** 2). *Doklady Akademii Nauk SSSR* 269, 3 (1983), 543.

[41] Yurii Nesterov. 2013. *Introductory lectures on convex optimization: A basic course*. Vol. 87. Springer Science & Business Media.

[42] Jorge Nocedal and Stephen J Wright. 1999. *Numerical optimization*. Springer.

[43] Min Pan, Natarajan Viswanathan, and Chris Chu. 2005. An efficient and effective detailed placement algorithm. In *Proc. ICCAD*. 48–55.

[44] Ryan Pattison, Ziad Abuowaimer, Shawki Areibi, Gary Grewal, and Anthony Vannelli. 2016. GPlace: A congestion-aware placement tool for UltraScale FPGAs. In *Proc. ICCAD*. 68:1–68:7.

[45] Chak-Wa Pui, Gengjie Chen, Wing-Kai Chow, Ka-Chun Lam, Jian Kuang, Peishan Tu, Hang Zhang, Evangeline F. Y. Young, and Bei Yu. 2016. RippleFPGA: A Routability-driven Placement for Large-scale Heterogeneous FPGAs. In *Proc. ICCAD*. 67:1–67:8.

[46] Chak-Wa Pui, Gengjie Chen, Yuzhe Ma, Evangeline FY Young, and Bei Yu. 2017. Clock-aware ultrascale FPGA placement with machine learning routability prediction. In *Proc. ICCAD*. 929–936.

[47] Rachel Selina Rajarathnam, Mohamed Baker Alawieh, Zixuan Jiang, Mahesh Iyer, and David Z Pan. 2022. DREAMPlaceFPGA: An open-source analytical placer for large scale heterogeneous FPGAs using deep-learning toolkit. In *Proc. ASPDAC*. IEEE, 300–306.

[48] Rachel Selina Rajarathnam, Zixuan Jiang, Mahesh A Iyer, and David Z Pan. 2023. DREAMPlaceFPGA-PL: An Open-Source GPU-Accelerated Packer-Legalizer for Heterogeneous FPGAs. In *Proc. ISPD*. 175–184.

[49] Peter Spindler, Ulf Schlichtmann, and Frank M. Johannes. 2008. Abacus: fast legalization of standard cell circuits with minimal movement. In *Proc. ISPD*. 47–53.

[50] Wenyu Sun and Ya-Xiang Yuan. 2006. *Optimization theory and methods: nonlinear programming*. Vol. 1. Springer Science & Business Media.

[51] Stephen Yang, Aman Gayasen, Chandra Mulpuri, Sainath Reddy, and Rajat Aggarwal. 2016. Routability-Driven FPGA Placement Contest. In *Proc. ISPD*. 139–143.

[52] Stephen Yang, Chandra Mulpuri, Sainath Reddy, Meghraj Kalase, Srinivasan Dasasathyan, Mehrdad E Dehkordi, Marvin Tom, and Rajat Aggarwal. 2017. Clock-Aware FPGA Placement Contest. In *Proc. ISPD*. 159–164.

[53] Ziran Zhu, Jianli Chen, Zheng Peng, Wenxing Zhu, and Yao-Wen Chang. 2018. Generalized augmented lagrangian and its applications to VLSI global placement. In *Proc. DAC*. 1–6.

[54] Ziran Zhu, Jianli Chen, Wenxing Zhu, and Yao-Wen Chang. 2020. Mixed-cell-height legalization considering technology and region constraints. *IEEE TCAD* 39, 12 (2020), 5128–5141.

[55] Ziran Zhu, Yangjie Mei, Zijun Li, Jingwen Lin, Jianli Chen, Jun Yang, and Yao-Wen Chang. 2022. High-performance placement for large-scale heterogeneous FPGAs with clock constraints. In *Proc. DAC*. 643–648.