DREAMPlace 3.0: Multi-Electrostatics Based Robust VLSI Placement with Region Constraints

Jiaqi Gu
UT Austin
jqgu@utexas.edu

Zixuan Jiang
UT Austin
zixuan@utexas.edu

Yibo Lin
Peking University
yibolin@pku.edu.cn

David Z. Pan
UT Austin
dpan@ece.utexas.edu

ABSTRACT

Placement is a critical step for modern very-large-scale integrated (VLSI) design closure. Recently, electrostatics-based analytical placement frameworks (ePlace) demonstrate promising performance in both solution quality and runtime. However, existing ePlace-based placers fail to meet the versatility and robustness requirements on various placement workloads. We propose a versatile and robust placer to solve region-constrained placement problems with better solution quality and faster convergence. We formulate the region-constrained placement problem into a multi-electrostatics system via virtual blockage insertion and field isolation. To achieve robust wirelength minimization with aggressive density constraints, we adopt self-adaptive quadratic density penalty and entropy injection techniques to automatically accelerate and stabilize the nonlinear optimization. Our experiments on ISPD 2015 benchmarks with region constraints demonstrate an average of >13% HPWL improvement and >11% top5 overflow improvement compared with advanced region-aware placers Eh?Placer and NTUplace4dr. Our robustness-boost techniques show an average of ∼1% and ∼10% improvement in HPWL and runtime compared to DREAMPlace on ICCAD 2014 and ISPD 2019 benchmark suites.

1 INTRODUCTION

Placement is a critical step in the VLSI design flow. It determines the physical locations of macros and standard cells in the layout. The placement quality has a significant impact on the following procedures, like clock synthesis and routing. Placement is also important for upstream procedures like physical aware logic synthesis since it provides an accurate estimation of cell congestion and wirelength.

Modern placement needs to handle various constraints to satisfy the design and manufacturing requirements. Fence regions are one kind of such constraints to isolate voltage regions and boost performance [1]. Cells assigned to a certain fence region, consisting of one or multiple rectangular sub-regions, have to be exclusively placed within the region boundary. Global placement not considering region constraints will result in huge quality degradation in legalization and detailed placement. Apart from region constraints, placement also needs to satisfy strict maximum density/utilization constraints for routability, leading to the difficulty in convergence for previous ePlace-based placers. Although we can manually tune parameters for each benchmark, it is impractical and engineering-intensive in front of extensive and diverse placement workloads. Therefore, a versatile and robust placement framework is in high demand that can handle the fence region and restricted density constraints with stable convergence and better solution quality.

Most state-of-the-art placers follow the framework of analytical placement algorithms [2]. An analytical global placer models the problem as a constrained nonlinear optimization which determines the cell locations while trying to minimize the wirelength and cell overlap. Previously, NTUplace4dr [3] was proposed to handle region constraints in placement tasks through a region-aware clustering and a new wirelength model. Eh?Placer [4] and RippleDR [5] were proposed to solve region-constrained placement by using an upper-bound-lower-bound optimization method with look-ahead rough legalization honoring region constraints. There are also many placers that tackle the classic placement problem without considering the region constraints [6–26]. Recently, ePlace-series [22–26] were proposed to model the cells as charges and cast the placement problem as a wirelength minimization task with electric potential energy constraints. However, the current unified electrostatic field is agnostic to region constraints and fails to support many advanced designs [1, 27]. Furthermore, existing ePlace-based placers still encounter slow convergence and inadequate robustness on newly released benchmarks with strict density constraints, which requires cumbersome manual parameter tuning.

In this work, we formulate the region-constrained placement problem into the optimization in a multi-electrostatics system and enable robust wirelength minimization with aggressive density constraints. Our placer improves the stability of the electrostatics-based placement algorithm with faster convergence and better solution quality. The key contributions are highlighted as follows.

- Flexibility: we propose a multi-electrostatics based placement engine that efficiently extend ePlace-series to handle region and restricted density constraints.
- Effectiveness: our quadratic density penalty, individual electric density controlling, and region-aware legalization methods boost the placement solution quality and congestion over 10% compared to a state-of-the-art region-aware placer.
- Robustness: we propose a divergence-aware optimizer with self-adaptive rollback and entropy injection that can improve wirelength and runtime by ∼1% and ∼10% with better convergence stability.

The remainder is organized as follows. Section 2 describes the background for analytical placement and motivation; Section 3 presents details about the proposed method; Section 4 demonstrate experimental results compared with previous placers on newly released benchmarks, followed by the conclusion in Section 5.

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415691

2 BACKGROUND

In this section, we briefly review the placement problem and our motivations.

2.1 Analytical Global Placement

An analytical placer usually splits the placement into three stages: global placement, legalization, and detailed placement. At the first global placement stage, we minimize the wirelength with density constraints, as formulated in Problem (1).

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & \sum_{e\in E} \mathrm{WL}(e;\mathbf{x},\mathbf{y}), \\ \text{s.t.} \quad & \mathcal{D}(\mathbf{x},\mathbf{y}) \le \hat{\mathcal{D}}, \end{aligned} \tag{1}$$

where $\mathrm{WL}(\cdot;\cdot)$ is the wirelength function of a net instance $e \in E$, $\mathcal{D}(\cdot)$ is the density of a certain location in the layout, and $\hat{\mathcal{D}}$ is a target density predefined by users. Then, the legalization procedure will be applied to move all cells to legal locations, during which row alignment, cell overlap, boundary enclosure, fence region constraints will be checked. Finally, detailed placement will continue optimizing the cell location as the last-stage refinement.

Penalty methods are a typical approach to solve the constrained optimization problem in global placement. We replace a constrained optimization problem by a series of unconstrained problems whose solutions ideally converge to the solution of the original constrained problem. Specifically, in global placement, we solve a series of unconstrained problems shown in Eq. (2),

$$\min_{\mathbf{x},\mathbf{y}} \quad \sum_{e\in E} \mathrm{WL}(e;\mathbf{x},\mathbf{y}) + \lambda \mathcal{D}(\mathbf{x},\mathbf{y}) \tag{2}$$

where $D(\cdot)$ is the density penalty to spread cells out in the layout. The density constraints can be satisfied by gradually increasing λ. When λ is small, we focus on minimizing the wirelength. As λ grows, we complete a smooth transition from (1) wirelength centric minimization to (2) wirelength and density co-optimization. The movable cells will spread gradually while keeping the optimal wirelength.

2.2 Electrostatics-based Placement

`ePlace`-series placers [22–25] are among a state-of-the-art family of placement algorithms that model the layout and netlist as a unified electrostatic system. It approximates the non-differentiable half-perimeter wirelength (HPWL) with a weighted-average wirelength (WA) model originally proposed by [28],

$$\mathrm{WA}_e = \frac{\sum_{i\in e} x_i e^{\frac{x_i}{\gamma}}}{\sum_{i\in e} e^{\frac{x_i}{\gamma}}} - \frac{\sum_{i\in e} x_i e^{-\frac{x_i}{\gamma}}}{\sum_{i\in e} e^{-\frac{x_i}{\gamma}}}, \tag{3}$$

where γ is a parameter that controls the smoothness and accuracy of the approximation to HPWL. A smaller γ indicates a more accurate but less smooth HPWL approximation.

Analogous to an electrostatic system, cells are modeled as charges, the cell density is modeled as potential energy, and the gradient corresponds to the electric force that drives the movement of cells. The electrostatic analogy has the following advantages: 1) smooth density penalty function; 2) a global view to the entire placement region even with very fine bin dimensions. The electric potential and field distribution can be computed by solving the Poisson's equation from the charge density distribution [22, 23].

$$\nabla \cdot \nabla \psi(x,y) = -\rho(x,y), \tag{4a}$$

$$\hat{\mathbf{n}} \cdot \nabla \psi(x,y) = \mathbf{0}, \quad (x,y) \in \partial R, \tag{4b}$$

$$\iint_R \rho(x,y) = \iint_R \psi(x,y) = 0, \tag{4c}$$

where R, ∂R, $\hat{\mathbf{n}}$ denote the placement region, the region boundary, and the outer normal vector of the region, respectively. ρ is the charge density, and ψ is the electric potential. The numerical solution of Poisson's equation can be efficiently obtained with spectral methods based on an $M \times M$ grid [22, 23].

A diagonal-Hessian-preconditioned Nesterov's optimizer with Golden-section line-search was proposed in [23] to solve this non-convex optimization problem (2). However, it is not easy to guarantee the convergence to solutions with high quality. Parameter tuning is necessary to search for a good configuration. For example, the placement bin dimensions, the stop criterion, and the density penalty weight often need to be adjusted for specific benchmarks.

2.3 Region and Density Constraints

Various placement constraints are imposed to be compatible with increasingly complex design rules. Region constraints are specified by designers to improve performance, leave space for later optimization, reduce power consumption, etc. [1]. By definition, a fence region is *member-hard* and *non-member-hard*. Specifically, interior cells that are assigned to a fence region must be placed inside the region boundary, and exterior cells that are not assigned to it are not allowed to appear in the region. A fence region consists of one or multiple spatially disjoint rectangular sub-regions, which may have overlaps with physical macros.

Maximum utilization constraints are user-defined local density limits for later-stage optimization and better routability, which are lower-bounded by the native logic utilization. A low target density represents a less dense placement solution, and different regions are likely to have different cell distributions, thus different utilization/density limits. Previous placement algorithms [3–5] adopt clustering, look-ahead rough legalization methods, and routability optimization techniques to address fence region and density constraints. In the current electrostatics-based placement framework [22], electric forces drive all movable cells to spread towards lower potential energy without considering region constraints or region-specific density limits. Based on our empirical experience, similar rough legalization techniques to impose region constraints during optimization does not work well in `ePlace`-based algorithms. Moreover, a relatively low target density is likely to cause convergence issues in `ePlace`-based placers. Therefore, we are motivated to put forward a flexible and robust placer to address region and density constraints with better solution quality and more stable convergence.

3 METHOD

In this section, we present details on our proposed multi-electrostatics based framework to solve VLSI placement tasks with fence region constraints.

3.1 Multi-Electrostatics Based Placement Engine with Region Constraint

Region constraints restrict the legal placement space for movable instances such that they have to be placed within the assigned boundary exclusively, and other exterior cells must be placed outside the boundary. Let v denote the position of all instances, and r represents the valid regions for these instances. Each region can be described as a disjoint set of rectangular subregions. We formulate the placement problem with K fence regions as,

$$\begin{aligned} \min_{v} \quad & \sum_{e \in E} \mathrm{WL}(e; v) + \lambda \mathcal{D}(v), \\ \text{s.t.} \quad & v_k = (\mathbf{x}_k, \mathbf{y}_k) \in r_k, \quad k = 0, \cdots, K, \end{aligned} \tag{5}$$

where the K-th region is for exterior cells that are not assigned to any fence region, i.e., the exterior region r_K is defined as the placeable space excluding physical macros (including placement blockages) and all fence regions. To solve this constrained non-convex optimization problem, we cast it as a multi-electrostatics system by relaxing the constraints as $K+1$ independent density penalty terms,

$$\min_{v} \quad \sum_{e \in E} \mathrm{WL}(e; v) + \langle \lambda, \mathcal{D}(v, r) \rangle, \tag{6}$$

where the density weight $\lambda \in \mathbb{R}^{K+1}$ is a vector $(\lambda_0, \cdots, \lambda_K)$ and $\mathcal{D}(v, r) = (\mathcal{D}(v_0, r_0), \cdots, \mathcal{D}(v_k, r_k))$ is a potential energy vector that considers fence regions into density calculation. We simplify $\mathcal{D}(v_k, r_k)$ to $\mathcal{D}_k$ in later discussion. Now, we discuss how to obtain the multi-electric potential energy and a quadratic penalty method for better convergence. The entire global placement flow of our multi-electrostatics based method is shown in Fig. 1.

3.1.1 Field Isolation and Virtual Blockage Insertion. In order to guide cells to their assigned legal regions simultaneously without interference, we replace the original universal electric field to $K+1$ mutually isolated electrostatic fields with independent fillers, potential maps, target density, electric force, density weight, and optimization stop criterion. This electric field isolation is only applied in electrostatics-related computation, while the wirelength objective remains the same as used in a unified electrostatic system. We denote the regions of total placement site and physical macros (including placement blockages) as A and m, respectively. For the k-th electrostatic field, we denote the assigned instance group and corresponding fence region as v_k and r_k, respectively. Every placeable space outside the assigned region is padded with positively-charged virtual blockages b_k, which is a union of one or multiple rectangles. The virtual rectangular blockages can be obtained by the following geometric operation,

$$b_k = \begin{cases} \texttt{rectangle\_slicing}(A \backslash (r_k \cup m)), & 0 \le k < K \\ \texttt{rectangle\_slicing}(r_k \backslash m), & k = K \end{cases} \tag{7}$$

where $\backslash$ represents geometric difference operation, $\cup$ is geometric union, and $\texttt{rectangle\_slicing}(\cdot)$ slices the polygon into multiple disjoint rectangular boxes. This method guarantees the overlapping between fence region and macros are correctly handled and the virtual blockages have no overlapping with physical macros. We visualize the virtual blockages on ISPD 2015 superblue16_a in Fig. 2. In the electric potential energy computation, the virtual blockages will contribute to the total potential energy such that the spreading of cells v_k is aware of the region constraint r_k. Instead of using a global target density $\hat{\mathcal{D}} \in [0, 1]$ to assign the charge of fixed macros, we observe that an individual target density is important for stable convergence and solution quality. Based on the area utilization statistics for this specific electrostatic field, we calculate the local target density $\hat{\mathcal{D}}_k \in [0, 1]$ as,

$$\hat{\mathcal{D}}_k = \max\left(LocalAreaUtil + \epsilon, \hat{\mathcal{D}}\right) = \max\left(\frac{Area(v_k)}{Area(r_k \backslash m)} + \epsilon, \hat{\mathcal{D}}\right), \tag{8}$$

where ϵ is a density margin that is adopted to avoid divergence issue induced by overly low target density.

Apart from independent virtual blockage insertion, another aspect of field isolation is independent filler insertion. Fillers are necessary virtual instances that can pad whitespaces for compact placement solution. The total area of movable filler cells v_k^f for region r_k is calculated based on the local target density,

$$Area(v_k^f) = \left(A - Area(m) - Area(b_k)\right)\hat{\mathcal{D}}_k - Area(v_k). \tag{9}$$

For better convergence, we set the filler height as the placement row height and width as the average width of movable standard cells $w(v_k^f) = \frac{1}{|v_k|}\sum w(v_k)$, where the smallest 5% and largest 5% outliers are discarded. Fillers will be uniformly randomly placed within its region r_k in the initialization for a smooth initial energy distribution. Figure 3 shows the multi-electrostatics based placement process. The filler initialization and cell spreading under multi-field are demonstrated in Fig 3a,3b,3c. Movable instances and fillers are driven towards their individual fence region by the repulsive force from virtual blockages. Fillers generated from individual target density lead to compact placement in each region.

In terms of algorithmic complexity and runtime, this multi-electric-field method has the same sequential complexity $O(|v| + M^2 \log M)$ as the original unified electric field. Our efficient multi-electrostatics based method has sublinear runtime overhead w.r.t. to number of regions K compared with the original single electrostatic field. The first reason is that the initial density map from virtual blockages only need to be calculated once initially. The second reason is that different regions deal with their own instances independently, thus the complexity for region k is $O(|v_k| + M^2 \log M)$. An efficient parallel density computation for all regions can be expected for further speedup.

3.1.2 Quadratic Density Penalty with Dynamic Density Weight Scheduling. During optimization of Eq. (6), the wirelength dominates in the beginning with a relatively small density weight λ for better solution quality. This mechanism works well when sufficient nets exist among macros, I/O pins, and movable instances, such that the wirelength-induced force can help spread the cells. In other words, the overall wirelength-related gradient has a similar high-dimensional direction to the density-related gradient. However, this statement does not always hold when the connections between movable instances and fixed macros are lacking, such that the wirelength objective tries to collapse all cells as a tiny cluster for hundreds of iterations until density weight gets large enough. To stabilize and accelerate the convergence of our multi-electrostatic system on various benchmarks, we extend the first-order density

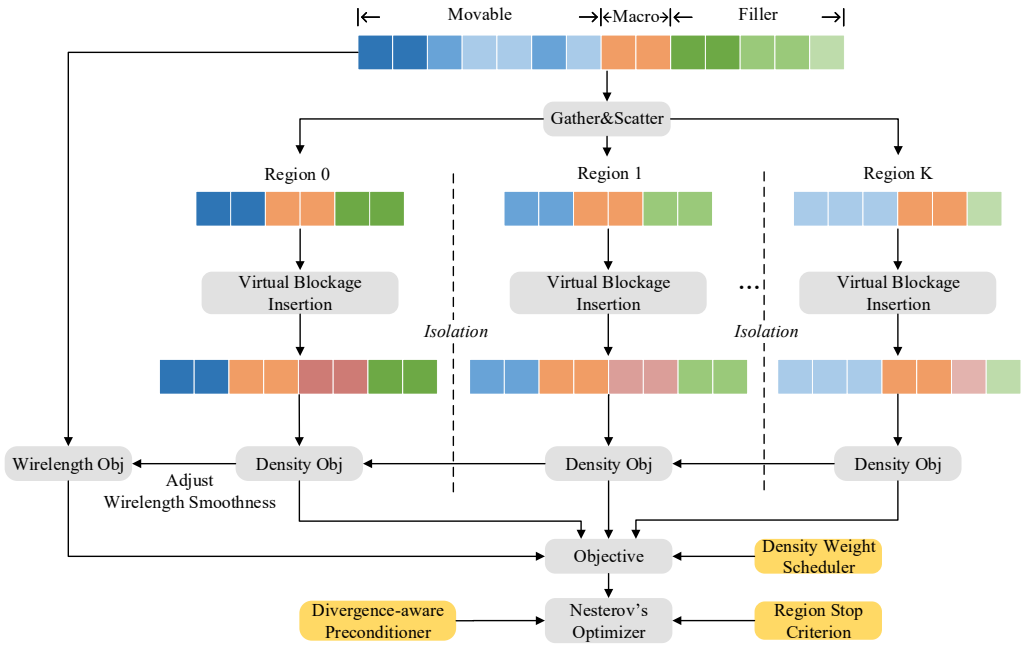

Figure 1: Multi-electrostatics based placement framework with region constraints.

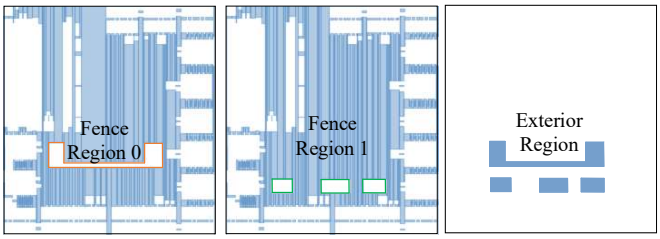

Figure 2: Virtual blockage insertion on ISPD 2015 [1] superblue16_a with 2 fence regions and an exterior region (region 2 in our annotation). Blue shadows are sliced rectangular virtual blockages for regions, where physical macros are excluded based on Eq. (7).

penalty in Eq. (6) using a modified augmented Lagrangian formulation [20, 29],

$$f = \sum_{e \in E} \mathrm{WL}(e; v) + \left\langle \lambda, \mathcal{D}(v, r) + \frac{1}{2}\mu \mathcal{P}_\lambda \odot \mathcal{D}^2(v, r) \right\rangle, \tag{10}$$

where the quadratic penalty is also controlled by the density weight λ to avoid over-spreading of cells. The weighting coefficient μ balances between first-order and second-order density penalty terms, which is empirically set to 1000. $\mathcal{P}_\lambda \in \mathbb{R}^{K+1}$ is a vectorized density weight preconditioner based on the initial density $\mathcal{P}_\lambda = \frac{1}{\mathcal{D}^0}$. The period before the entire instances are completely spread is critical for wirelength optimization. Therefore, we need to carefully adjust the density weight individually for each electrostatic field. We adopt a first-order-gradient-based method [29] to update the density weight λ with a preconditioned subgradient estimator, described in Alg. 1. Lines 1-4 initialize the density weight based on

Algorithm 1 Density weight scheduling for multi-electrostatic system.

Require: electric potential energy function $\mathcal{D}(\cdot)$, wirelength function $\mathrm{WL}(\cdot)$, density weight scaling factor η, maximum density weight λ_{max}, density weight preconditioner $\mathcal{P}_\lambda$, quadratic penalty weighting coefficient μ, density weight step size lower bound α_l and upper bound α_h and maximum iteration T.

1: $c^0 \leftarrow 1 + \mu \mathcal{D}^0 \odot \mathcal{P}_\lambda$
2: $u^0 \leftarrow \mathcal{D}^0 \odot \mathcal{P}_\lambda + \frac{1}{2}\mu (\mathcal{D}^0 \odot \mathcal{P}_\lambda)^2$
3: $u^0 \leftarrow \eta \frac{\|\nabla \sum_e \mathrm{WL}(e, v^0)\|_1}{\sum_{k=0}^{K} u_k^0 c_k^0 \|\nabla \mathcal{D}_k^0\|_1} u^0$
4: $\lambda^0 \leftarrow u^0 \odot c^0$ ▷ Initialize density weight vector
5: $\alpha^0 \leftarrow (\alpha_l - 1)\|u^0\|_2$ ▷ Initialize density weight step size
6: **for** $t \leftarrow 1, \cdots, T$ **do**
7: $\quad \nabla \lambda^t \leftarrow \frac{\mathcal{D}^t \odot \mathcal{P}_\lambda + \frac{\mu}{2}(\mathcal{D}^t \odot \mathcal{P}_\lambda)^2}{\|\mathcal{D}^t \odot \mathcal{P}_\lambda + \frac{\mu}{2}(\mathcal{D}^t \odot \mathcal{P}_\lambda)^2\|_2}$
8: $\quad u^{t+1} \leftarrow u^t + \alpha^t \nabla \lambda^t$
9: $\quad c^{t+1} \leftarrow 1 + \mu \mathcal{D}^t \odot \mathcal{P}_\lambda$
10: $\quad \lambda^{t+1} \leftarrow \min(\lambda_{max}, u^{t+1} \odot c^{t+1})$ ▷ Update density weight vector
11: $\quad \alpha^{t+1} \leftarrow \alpha^t \left(\frac{\left(\ln(\mu \|\mathcal{D}^t \odot \mathcal{P}_\lambda\|_2)\right)_+}{1 + \left(\ln(\mu \|\mathcal{D}^t \odot \mathcal{P}_\lambda\|_2)\right)_+} (\alpha_h - \alpha_l) + \alpha_l \right)$ ▷ Update density weight step size

the ratio of wirelength gradient norm and density gradient norm, where the density weight scaling factor η is set to 8×10^{-5} in our experiments. Line 5 initializes the density weight step size based on the predefined lower bound. After each iteration, we calculate the normalized subgradient of the density weight in line 7 and update the density weight using gradient descent method in lines 8-10. The density weight step size increases exponentially with a density-controlled factor in line 11. We configure the lower and upper bound of density weight step size to 1.03 and 1.04, respectively.

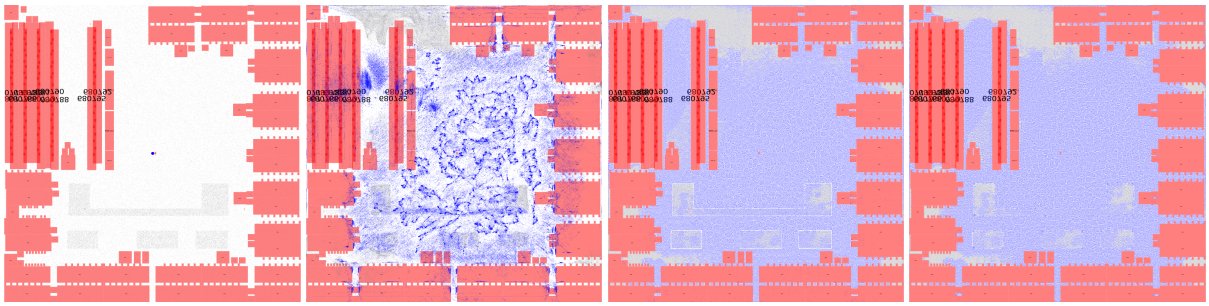

Figure 3: Multi-electrostatics based placement visualization on ISPD 2015 [1] superblue16_a. (a) Initialization with uniformly placed fillers (gray cells) in each region. (b) Region-aware cell spreading based on field isolation and virtual blockage insertion. (c) Global placement result. (d) Legal solution after region-aware legalization.

3.1.3 Multi-field Gradient Propagation with Divergence-aware Preconditioning. To efficiently control the gradients flow and reduce the overhead for multi-electrostatic-field optimization, we leverage the differentiable gather/scatter operation to isolate the gradients from different electric fields, shown in Fig. 1. Moreover, we observe that gradient flows from different electric fields are not balanced such that the optimization for the K-th region r_K may converge slowly or even diverge. We apply divergence-aware gradient preconditioning partially to instances v_K. The second-order derivative of the wirelength objective is estimated with the pin count of the instances [22], while the second-order derivative of the augmented-Lagrangian-based density penalty is approximated by the instance area [22, 25, 29], which is the diagonal Hessian. The divergence-aware preconditioner $\mathcal{P}_K \in \mathbb{R}^{2|v|}$ is given by,

$$\begin{aligned}\mathcal{P}_K &= \min\left(1, \left(\nabla^2_{v_K}\sum_e WL(e,v) + \beta\lambda_K\nabla^2_{v_K}\mathcal{D}(v_K, r_K)\right)^{-1}\right)\\ &= \min\left(1, \left(\#pin(v_K) + \beta\lambda_K w(v_K)\odot h(v_K)\right)^{-1}\right)\end{aligned} \tag{11}$$

where $\#pin(\cdot)$ is the number of pins of an instance, $w(\cdot)$ and $h(\cdot)$ is the cell width and height. For fence regions $0 \le k < K$, we have $\mathcal{P}_k = 1$. We initialize β to 1 and dynamically adjust it when the global density overflow is below a certain threshold, e.g., 0.3. Specifically, β is doubled every 20 optimization iterations to slow down the movement of large-sized cells. The global electric density overflow is given by,

$$OVFL = \frac{\sum_{u_x=0,u_y=0}^{M-1,M-1}\left(\rho(u_x,u_x) - \hat{\mathcal{D}}\right)_+}{Area(v)}, \tag{12}$$

where $\rho(u_x, u_y)$ is the normalized charge density on an $M \times M$ grid. The grid size M is typically configured as 512 or 1024. This global overflow is also used to adjust the wirelength smoothness factor γ. Based on this dynamically-adjusted preconditioner, we give the preconditioned gradient to the Nesterov's optimizer,

$$\hat{\nabla} f = \nabla f \odot \mathcal{P}. \tag{13}$$

The divergence-aware preconditioning is designed to slow down the sensitive movement of large-sized cells when global placement is nearly converged, which can help stabilize the optimization. An upper bound of β, e.g., 2^{10}, is set to avoid numerical instability.

3.1.4 Independent Stop Criterion. Different electrostatic systems converge at different speeds, thus the same optimization iterations can lead to divergence due to density weight explosion given its exponentially increased step size. Therefore, to adapt to different convergence speeds of $K+1$ regions, we first set up an upper bound for the density weight, e.g., 10, and adopt an individual stop criterion for each electric field based on its individual density overflow,

$$OVFL_k = \frac{\sum_{u_x=0,u_y=0}^{M-1,M-1}\left(\rho_k(u_x,u_y) - \hat{\mathcal{D}}_k\right)_+}{Area(v_k)}, \tag{14}$$

where $\rho_k(u_x, u_y)$ is the normalized charge density map for region k. Once the density overflow of the k-th region runs below the global stop overflow, e.g., 0.07, we freeze the movable instances v_k and the corresponding filler cells v_k^f to avoid divergence. The gradient from the wirelength and density objectives will only flow to unfrozen instances.

3.2 Region-Aware Legalization

After all $K+1$ regions finish their global placement flow, we legalize the solution with a region-aware incremental legalizer for our multi-electrostatic system. Similar to the field isolation used in global placement, for region r_k, we transform the fence region constraint into a non-fence-region legalization problem by inserting virtual blockages. We can simultaneously legalize $K+1$ regions without interference. The detailed legalization flow is described in Alg 2. The invalid space constrained by fence region is padded with virtual blockages such that the region-unaware legalizer can generate equivalent solution to a specifically designed region-aware legalizer. Abacus legalizer [30] is used to minimize the total displacement between the final legal solution and the global placement solution. Figure 3d visualize the legal solution after region-aware legalization. The complexity of this region-aware legalization algorithm is $O(|v|)$, and different regions can be potentially legalized in parallel.

Algorithm 2 Fence-region-aware incremental legalization based on virtual blockage insertion.

Require: global placement solution $v^T = (v_0^T, v_1^T, \cdots, v_K^T)$, physical macros and placement blockages m, regions $r = (r_0, r_1, \cdots, r_K)$, virtual blockages for each region $b_0, b_1, \cdots, b_K$, movable macro legalizer $\texttt{ml}(\cdot)$, greedy standard cell legalizer $\texttt{gl}(\cdot)$, Abacus standard cell legalizer $\texttt{al}(\cdot)$, and legality checker with fence region constraints $\texttt{lg\_check}(\cdot, r)$.

Ensure: legalized placement solution with region constraints $\tilde{v} = (\tilde{v_0}, \tilde{v_1}, \cdots, \tilde{v_K})$.

1: **for** $k \leftarrow 0, \cdots, K$ **do**
2: $v_k^m \leftarrow \texttt{ml}(v_k^T, m, b_k)$ ▷ Legalize movable macros
3: $v_k^g \leftarrow \texttt{gl}(v_k^m, m, b_k)$ ▷ Greedily legalize instances in region k
4: **if** $\texttt{lg\_check}(\tilde{v})$ = True **then**
5: **for** $k \leftarrow 0, \cdots, K$ **do**
6: $\tilde{v_k} \leftarrow \texttt{al}(v_k^m, v_k^g, m, b_k)$ ▷ Minimize displacement between v^m and v^g
7: **else**
8: $\tilde{v} \leftarrow (v_0^g, v_1^g, \cdots, v_K^g)$ ▷ Not enough space for a legal placement

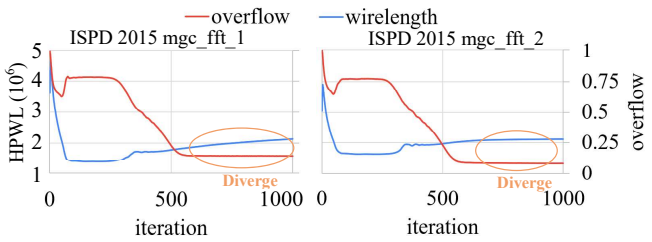

Figure 4: Diverged HPWL and overflow curves using previous method on two ISPD 2015 benchmarks [1]

3.3 Robust Optimization via Self-Adaptive Rollback and Entropy Injection

The above proposed multi-electrostatics based method enables our engine to handle region-constrained placement tasks. However, a robust placement engine is required to achieve stable convergence on various benchmarks, including those without fence regions. Placement problem is a nonconvex optimization problem where poor first-order and second-order stationary points will trap the optimizer, leading to an undesired saddle point in density objective and degraded local minima in wirelength objective [31]. There are some benchmarks where the original Nesterov's optimizer [23, 25] will diverge, leading to longer runtime and unsatisfying solution quality. Even though a preconditioned Nesterov's accelerated gradient descent optimizer with backtracking line search is adopted as our optimizer, in some benchmarks we observe suboptimal placement solutions and slow convergence speed or even divergence, shown in Fig. 4. We propose to a self-adaptive Nesterov's optimizer to achieve faster, more stable convergence with better solution quality. We first address slow convergence issue caused by slow cell spreading in the early stage. A window-based plateau (PLT) detector is adopted to detect slow convergence,

$$\text{PLT} = \begin{cases} \dfrac{\max_L(OVFL) - \min_L(OVFL)}{\text{avg}_L(OVFL)} < \delta_{PLT}, & OVFL > 0.9 \\ \text{False}, & OVFL \leq 0.9, \end{cases} \tag{15}$$

where $\max_L(\cdot)$, $\min_L(\cdot)$, and $\text{avg}_L(\cdot)$ calculate the maximum, minimum, and average value within an L-length window, respectively. δ_{PLT} is the divergence threshold. We empirically set L and δ_{PLT} to 20 iterations and 0.1%. Once PLT is asserted, we perform three steps, 1) turn on quadratic density penalty term if the default engine disables it, 2) increase density weight, e.g., by 2×, and 3) inject perturbation to non-fixed instances with layout shrinking. By default, we enable quadratic penalty and subgradient-based density weight scheduling for region-constrained benchmarks to stabilize multiple electrostatic fields. For benchmarks without fence regions, our engine by default falls back to the original `ePlace` formulation, where the original first-order density penalty and HPWL-based density weight adjustment [22, 25] are used. For brevity, we show the perturbation along the x-axis as,

$$\hat{x} = s\left(x - \frac{\sum_{i \in v} x_i}{|v|}\right) + \frac{\sum_{i \in v} x_i}{|v|} + \Delta x, \tag{16}$$

where Δx is a perturbation vector sampled from a multivariate Gaussian distribution $\Delta x \sim \mathcal{N}(0, \sigma^2)$. The same entropy injection mechanism is also applied to the y-axis. The shrinking factor $s \in (0, 1)$ is designed for backtracking the spreading process and increasing the density overflow, providing opportunities to re-optimize the perturbed wirelength. Inspired by perturbed gradient descent methods [31], this strategy injects necessary entropy into the optimization engine to help escape from saddle points or high potential wells in the solution space [31]. Also, in the stage when overflow is still higher than 0.9, noise injection can be considered as a cell spreading mechanism.

Further, we address the divergence issue of the original Nesterov's optimizer [22]. In benchmarks with restricted density constraints where the target density is very close to the native utilization, the optimizer is likely to diverge after a relatively low density overflow is achieved. Further optimization leads to severe HPWL degradation or even a corrupted placement solution. The primary reason is that the previous Nesterov's optimizer adopts the Golden-section line search with estimated Lipschitz constant to decide the appropriate step size [23], it will regardlessly update the cell location even if the gradient descent direction is hard to find. Given that such a mechanism does not guarantee an objective improvement in each parameter updating step, it fails to tackle the extremely sensitive late-stage in placement while the density overflow fluctuates before reaching stop criterion. We adopt a window-based divergence detector to check whether the overflow fluctuates a lot or wirelength and overflow are worse than the best-recorded checkpoint over a certain threshold, e.g., $\frac{OVFL}{100}$. If divergence is detected in the later optimization stage, e.g., $OVFL < 4\times$StopOVFL, we will immediately stop global placement to avoid prolonged runtime and roll back to the checkpoint with best-recorded overflow as our final global placement solution to prevent quality degradation.

4 EXPERIMENTS

To show the effectiveness of our multi-electrostatics based VLSI placement algorithm, we evaluate the HPWL and congestion metrics on ISPD 2015 contest benchmark suite [1] and compare the results with `Eh?Placer` [4] and `NTUplace4dr` [3] using the same target utilization limit. Since `Eh?Placer` and `NTUplace4dr` binary executable files currently do not support ISPD 2019 [27] or ICCAD

Table 1: Comparison of HPWL, top5 overflow, and runtime with `Eh?Placer` [4] and `NTUplace4dr` [3] on ISPD 2015 benchmark suite with region constraints. OVFL is short for overflow. HPWL and top5 OVFL are normalized to our results. After detailed placement of all placers, we use `NTUplace4dr` and the embedded `NCTUgr` [32] global router to evaluate their dumped final placement solutions and report their HPWL and top5 overflow with no legality violation. *RT* represents runtime in seconds.

Design	#cell	#nets	#region	`Eh?Placer` [4] (8 CPU threads)			`NTUplace4dr` [3] (8 CPU threads)			Ours (GPU)		
				Top5 OVFL	HPWL	RT	Top5 OVFL	HPWL	RT	Top5 OVFL	HPWL	RT
mgc_des_perf_a	108K	115K	4	74.40	2.69E+09	59	73.14	2.44E+09	207	66.17	2.23E+09	16
mgc_des_perf_b	113K	113K	12	62.00	1.77E+09	49	71.71	2.00E+09	212	63.01	1.80E+09	30
mgc_edit_dist_a	127K	134K	1	90.67	5.08E+09	66	92.95	4.91E+09	140	86.68	4.26E+09	12
mgc_matrix_mult_b	146K	152K	3	48.77	3.82E+09	97	49.56	3.64E+09	172	47.87	3.16E+09	17
mgc_matrix_mult_c	146K	152K	3	47.24	3.71E+09	106	48.13	3.48E+09	198	46.57	3.06E+09	16
mgc_pci_bridge32_a	30K	34K	4	39.74	4.10E+08	14	41.23	4.72E+08	54	35.71	3.88E+08	15
mgc_pci_bridge32_b	29K	33K	3	31.94	8.14E+08	15	35.79	6.98E+08	37	31.82	6.36E+08	18
mgc_superblue11_a	926K	936K	4	59.96	3.93E+10	653	58.61	4.00E+10	8684	56.15	3.51E+10	56
mgc_superblue16_a	680K	697K	2	68.81	2.86E+10	359	89.67	2.94E+10	3125	68.95	2.69E+10	32
ratio				1.124	1.206	3.717	1.112	1.133	34.756	1.000	1.000	1.000

2014 [33] benchmark suites due to benchmark format issues, we only evaluate on ISPD 2015 benchmark suite. It is worth noting that in this work, our placement engine is wirelength-driven without explicit routability optimization. Therefore, only comparing the HPWL metric with `Eh?Placer` [4] and `NTUplace4dr` [3], which involve dedicated optimization effects on detailed-routability and design rule violation, etc., is not a very fair evaluation. We choose to show the *top5 overflow*, the average global routing overflow in the top 5% congested routing grids, reported by `NCTUgr` [32] global router as a representative metric for routability evaluation. To clarify, our wirelength-driven region-aware placement engine does not claim better global-routability or detailed-routability than the other two placers as routability is not part of our optimization objective. Hence, a less competitive routed wirelength, routability, and design rule violations can be expected for our method. We also compare with `DREAMPlace` [25] on ISPD 2019 and ICCAD 2014 benchmarks without region constraints to validate our convergence robustness.

4.1 Experimental Settings

We build our multi-electrostatic system on an open-source VLSI placer `DREAMPlace`, including a GPU-based global placer, a CPU-based legalizer, and a GPU-based detailed placer `ABCDPlace` [34]. Our baseline placers are `Eh?Placer` [4], `NTUplace4dr` [3], and `DREAMPlace` [25]. All the experiments run on a Linux server with Intel Core i9-7900X @ 3.30 GHz and one NVIDIA TITAN Xp GPU. We use the datatype of single-precision floating-point for evaluation. `Eh?Placer` and `NTUplace4dr` run on the CPU with 8 threads, while `DREAMPlace` and our framework are executed on the GPU.

4.2 HPWL and Routability Evaluation

On ISPD 2015 benchmarks [1] with region constraints, we compare HPWL, top5 overflow, and runtime with `Eh?Placer` [4] and `NTUplace4dr` [3] in Table 1. `DREAMPlace` [25] is not evaluated on these nine benchmarks since it does not support fence region constraint currently. We tried to modify `DREAMPlace` and perform rough legalization during global placement to force the cell to spread in the assigned region. However, performance and optimization stability are much worse than the other two methods. Thus `DREAMPlace` is not reported in Table 1 for concise demonstration.

On all nine benchmarks with fence region constraints, the proposed placer achieves an average of 20.6% and 13.1% HPWL improvement compared with `Eh?Placer` and `NTUplace4dr`, respectively. Even if we do not perform the routability optimization, our multi-electrostatics based placement engine can achieve an average of 12.4% and 11.2% top5 overflow improvement compared with detailed-routability-driven `Eh?Placer` and `NTUplace4dr`, respectively. However, it is worth noting that this overflow improvement does not necessarily indicate better detailed-routability, as routability-related rules are not considered in our work. A thorough evaluation of detailed-routability and design rule violations can be obtained by the industrial platform provided by the ISPD 2015 contest. But unfortunately, we currently do not have access to the official industrial evaluation platform [1] that has already been unavailable. In terms of runtime, we record the elapsed wall time for the entire placement flow. Our GPU-accelerated multi-electrostatics based placement achieves 3.717× and 34.76× runtime speedup than 8-CPU-threaded `Eh?Placer` and `NTUplace4dr`, respectively.

4.3 Robustness Evaluation

In this section, we evaluate the robustness of our placement engine on newly released benchmarks, including ICCAD 2014 [33], ISPD 2015 [1], and ISPD 2019 [27]. Table 2 shows the comparison results with `Eh?Placer` [4], `NTUplace4dr` [3], and `DREAMPlace` [25] on the rest 11 ISPD 2015 benchmarks without fence regions. We can achieve an average of 3.8% top5 overflow reduction, 17.0% HPWL improvement, and 13.89× speedup compared to `Eh?Placer`. Compared with `NTUplace4dr`, our method achieves an average of 7.2% and 37.8× improvement in HPWL and runtime, respectively. Since our placer is wirelength-driven without routability optimization, we have slightly larger, i.e., 3%, top5 overflow than `NTUplace4dr`. Compared with `DREAMplace` [25], we achieve 1.4% better HPWL, 3.3% lower top5 overflow, and a faster convergence speed. In Table 3, we further evaluate the robustness on ISPD 2019 and ICCAD 2014 benchmark suites. Note that `NTUplace4dr` is not evaluated due to benchmark compatibility issues.

Table 3 compares our method with `DREAMPlace`. On those benchmarks, our method can adaptively detect convergence issues, and

Table 2: Comparison of HPWL, top5 overflow, and runtime with Eh?Placer [4], NTUplace4dr [3], and DREAMPlace [25] on ISPD 2015 benchmark without region constraints. *OVFL* is short for overflow. HPWL and top5 OVFL are normalized to our results. After detailed placement of all placers, we use NTUplace4dr and the embedded NCTUgr [32] global router to evaluate their dumped final placement solutions and report their HPWL and top5 overflow with no legality violation. *RT* represents runtime in seconds. For DREAMPlace, diverged (*div.*) benchmarks are stopped after reaching its maximum iteration. For our method, *r*, *q*, *e* represent rollback, quadratic penalty, or entropy injection is triggered.

Design	#cell	#nets	Eh?Placer [4] (8 CPU threads)			NTUplace4dr [3] (8 CPU threads)			DREAMPlace [25] (GPU)			Ours (GPU)			
			Top5 OVFL	HPWL	RT	Top5 OVFL	HPWL	RT	Top5 OVFL	HPWL	RT	Top5 OVFL	HPWL	RT	
des_perf_1	113K	113K	68.27	1.30E+09	73.7	63.87	1.20E+09	200.9	67.92	1.13E+09	5.4	65.78	1.12E+09	5.3	q&e
fft_1	35K	33K	60.14	4.49E+08	27.9	61.57	4.60E+08	51.9	56.54	4.16E+08	5.3(div.)	56.29	4.12E+08	4.2	r
fft_2	35K	33K	49.17	4.36E+08	16.8	55.10	4.80E+08	69.3	47.91	3.84E+08	5.5(div.)	47.56	3.75E+08	4.3	r
fft_a	34K	32K	37.60	8.42E+08	15.3	36.07	6.39E+08	57.1	35.15	6.29E+08	4.3	34.91	6.27E+08	4.1	
fft_b	34K	32K	58.45	1.02E+09	15.1	53.19	8.48E+08	63.4	52.07	8.47E+08	4.4	52.13	8.48E+08	4.2	
matrix_mult_1	160K	159K	75.09	2.28E+09	70.1	73.43	2.27E+09	188.9	81.84	2.13E+09	5.4	81.63	2.13E+09	5.4	
matrix_mult_2	160K	159K	74.54	2.32E+09	76.9	73.24	2.25E+09	204.0	76.88	2.16E+09	5.6	77.37	2.16E+09	5.5	
matrix_mult_a	154K	154K	50.09	3.68E+09	92.2	48.62	3.33E+09	215.6	48.09	3.03E+09	6.6	48.18	3.03E+09	7.2	
superblue12	1293K	1293K	80.23	3.82E+10	867.4	78.84	3.48E+10	16369	93.17	2.61E+10	35.0	92.90	2.58E+10	34.6	q&e
superblue14	634k	620k	61.55	2.54E+10	497.5	67.76	2.49E+10	7153.9	63.18	2.31E+10	20.1	63.18	2.30E+10	19.3	
superblue19	522K	512K	64.38	1.83E+10	237.5	64.55	1.71E+10	7788.7	61.95	1.57E+10	15.7	61.95	1.57E+10	15.7	
ratio			1.038	1.170	13.887	0.971	1.072	37.838	1.033	1.014	1.019	1.000	1.000	1.000	

Table 3: HPWL and runtime (seconds) comparison with DREAMPlace [25] on ISPD 2019 and ICCAD 2014 benchmark suites. ispd19_test_5 has 3 fence regions which cannot be handled by DREAMPlace [25]. HPWL after detailed placement and runtime are reported by the evaluator in DREAMPlace. Diverged (*div.*) benchmarks are stopped after reaching the maximum iteration of DREAMPlace. *q&e* means quadratic penalty and entropy injection are triggered. Benchmarks that do not trigger our self-adaptive robust optimization mechanism have the same results in two methods.

Design	#cell	#net	DREAMPlace [25]		Ours		
			HPWL	RT	HPWL	RT	
ispd19test1	9K	3K	8.20E+07	5.63(div.)	8.08E+07	4.21	q&e
ispd19test2	73K	72K	3.58E+09	5.77	3.51E+09	5.70	q&e
ispd19test3	8K	9K	1.41E+08	3.77	1.40E+08	4.17	q&e
ispd19test4	151K	146K	3.41E+09	6.42	3.41E+09	6.42	
ispd19test5	29K	29K	-	-	6.30E+08	12.10	
ispd19test6	181K	180K	8.96E+09	7.87	8.94E+09	7.68	q&e
ispd19test7	362K	359K	1.78E+10	11.42	1.76E+10	11.92	q&e
ispd19test8	543K	538K	2.68E+10	15.76	2.67E+10	15.12	q&e
ispd19test9	903K	895K	4.12E+10	32.36	4.13E+10	20.39	q&e
ispd19test10	903K	895K	4.20E+10	27.38	4.19E+10	21.90	q&e
iccad14b19	219K	219K	2.63E+08	7.03	2.61E+08	6.91	q&e
iccad14leon2	795K	795K	2.32E+09	34.8	2.32E+09	30.02	q&e
iccad14leon3mp	649K	649K	1.07E+09	25.65	1.07E+09	21.77	q&e
iccad14edit_dist	133K	133K	3.98E+08	6.18	3.98E+08	6.18	
iccad14matrix_mult	155K	159K	2.38E+08	5.64	2.38E+08	5.64	
iccad14netcard	959K	961K	2.80E+09	35.04(div.)	2.69E+09	29.25	q&e
iccad14vga_lcd	165K	165K	2.70E+08	7.57	2.68E+08	7.20	q&e
ratio			1.007	1.108	1.000	1.000	

turn on corresponding techniques to help accelerate and stabilize optimization. For example, on mgc_fft_1 benchmark, the target density limit is equal to the native logic utilization 83.5%. Such a highly restricted density constraint leads to divergence in DREAMPlace. Our window-based divergence detector detects that the normalized overflow increases too much compared to the best-recorded overflow in the 619-th iteration while the degrading trend of HPWL also exceeds the predefined threshold, i.e., $\frac{OVFL}{100}$. Thus it immediately rolls back to the 613-th iteration with the lowest overflow, which prevents further quality degradation or prolonged runtime.

On ISPD 2019 and ICCAD 2014 benchmarks, our method boosts the runtime by 10.8% compared to DREAMPlace [25], while improving the detailed placement HPWL by an average of 0.7%. 13 benchmarks marked as *q&e* in Table 2 indicate that our self-adaptive quadratic penalty and entropy injection mechanisms are triggered to accelerate the cell spreading in the early optimization stage.

5 CONCLUSIONS

In this work, we propose a flexible and robust VLSI placer that can handle fence region constraints with better solution quality and more stable convergence. We propose a multi-electrostatics based method to honor fence region constraints in the electrostatic-based placement algorithm. Our field isolation and virtual blockage insertion techniques enable efficient region-aware placement with a global view that leads to better solution quality than previous methods. We adopt a quadratic density penalty and individual electric density controlling methods to accelerate and stabilize cell spreading for each region. A region-aware legalizer is proposed to efficiently legalize the solution under region constraints. Besides, a self-adaptive rollback and entropy injection mechanism is proposed to boost the robustness and solution quality of the nonconvex optimization. Experimental results demonstrate that on ISPD 2015 benchmarks we can achieve >13% and >11% HPWL improvement and top5 overflow reduction compared to previous region-aware placers Eh?Placer [4] and NTUplace4dr [3]. On ISPD 2019 and ICCAD 2014, we can achieve ~1% and ~3% improvement in HPWL and overflow over DREAMPlace [25] with more stable convergence.

ACKNOWLEDGMENTS

The authors would like to thank Prof. Yao-Wen Chang at National Taiwan University for preparing the binary of NTUplace4dr [3] and Mr. Nima Karimpour Darav from Efinix Inc. for preparing the Eh?Placer binary [4]. This project is supported in part by the National Key Research and Development Program of China (No. 2019YFB2205000).

REFERENCES

[1] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "ISPD 2015 Benchmarks with Fence Regions and Routing Blockages for Detailed-Routing-Driven Placement," in *Proc. ISPD*, 2015, p. 157–164.

[2] I. L. Markov, J. Hu, and M. Kim, "Progress and Challenges in VLSI Placement Research," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 1985–2003, 2015.

[3] C. Huang, H. Lee, B. Lin, S. Yang, C. Chang, S. Chen, Y. Chang, T. Chen, and I. Bustany, "NTUplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE TCAD*, vol. 37, no. 3, pp. 669–681, 2018.

[4] N. K. Darav, A. Kennings, A. F. Tabrizi, D. Westwick, and L. Behjat, "Eh?Placer: A high-performance modern technology-driven placer," *ACM TODAES*, vol. 21, no. 3, Apr. 2016.

[5] W. Chow, J. Kuang, P. Tu, and E. F. Y. Young, "Fence-aware detailed-routability driven placement," in *Proc. SLIP*, 2017, pp. 1–7.

[6] A. B. Kahng, S. Reda, and Q. Wang, "Architecture and details of a high quality, large-scale analytical placer," in *Proc. ICCAD*, 2005, pp. 891–898.

[7] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *Proc. ISPD*, 2005, pp. 185–192.

[8] A. B. Kahng and Q. Wang, "A faster implementation of APlace," in *Proc. ISPD*, 2006, pp. 218–220.

[9] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE TCAD*, vol. 27, no. 7, pp. 1228–1240, 2008.

[10] M.-K. Hsu, Y.-F. Chen, C.-C. Huang, S. Chou, T.-H. Lin, T.-C. Chen, and Y.-W. Chang, "NTUplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE TCAD*, vol. 33, no. 12, pp. 1914–1927, 2014.

[11] C. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, vol. 38, no. 9, pp. 1717–1730, 2019.

[12] N. Viswanathan, M. Pan, and C. Chu, "FastPlace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *Proc. ASPDAC*, 2007, pp. 135–140.

[13] X. He, T. Huang, L. Xiao, H. Tian, and E. F. Y. Young, "Ripple: A robust and effective routability-driven placer," *IEEE TCAD*, vol. 32, no. 10, pp. 1546–1556, 2013.

[14] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "POLAR: placement based on novel rough legalization and refinement," in *Proc. ICCAD*, 2013, pp. 357–362.

[15] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "POLAR: A high performance mixed-size wirelengh-driven placer with density constraints," *IEEE TCAD*, vol. 34, no. 3, pp. 447–459, 2015.

[16] M.-C. Kim, D.-J. Lee, and I. L. Markov, "SimPL: An effective placement algorithm," *IEEE TCAD*, vol. 31, no. 1, pp. 50–60, 2012.

[17] M.-C. Kim, N. Viswanathan, C. J. Alpert, I. L. Markov, and S. Ramji, "MAPLE: multilevel adaptive placement for mixed-size designs," in *Proc. ISPD*, 2012, pp. 193–200.

[18] T. Chen, Z. Jiang, T. Hsu, H. Chen, and Y. Chang, "NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints," *IEEE TCAD*, vol. 27, no. 7, pp. 1228–1240, 2008.

[19] J. Chen, L. Yang, Z. Peng, W. Zhu, and Y. Chang, "Novel proximal group ADMM for placement considering fogging and proximity effects," in *Proc. ICCAD*, 2018, p. 3.

[20] Z. Zhu, J. Chen, Z. Peng, W. Zhu, and Y. Chang, "Generalized augmented lagrangian and its applications to VLSI global placement," in *Proc. DAC*, 2018, pp. 149:1–149:6.

[21] W. Zhu, Z. Huang, J. Chen, and Y.-W. Chang, "Analytical solution of poisson's equation and its application to vlsi global placement," in *Proc. ICCAD*, 2018.

[22] J. Lu, P. Chen, C. Chang, L. Sha, D. Huang, C. Teng, and C. Cheng, "ePlace: Electrostatics-Based Placement using Fast Fourier Transform and Nesterov's Method," *ACM TODAES*, vol. 20, no. 2, p. 17, 2015.

[23] J. Lu, H. Zhuang, P. Chen, H. Chang, C. Chang, Y. Wong, L. Sha, D. Huang, Y. Luo, C. Teng, and C. Cheng, "eplace-ms: Electrostatics-based placement for mixed-size circuits," *IEEE TCAD*, vol. 34, no. 5, pp. 685–698, 2015.

[24] C. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, vol. 38, no. 9, pp. 1717–1730, 2019.

[25] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," in *Proc. DAC*, 2019.

[26] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," *IEEE TCAD*, pp. 1–1, 2020.

[27] S. Dolgov, A. Volkov, L. Wang, and B. Xu, "2019 CAD Contest: LEF/DEF Based Global Routing," in *Proc. ICCAD*, 2019, pp. 1–4.

[28] M. Hsu, Y. Chang, and V. Balabanov, "TSV-aware analytical placement for 3D IC designs," in *Proc. DAC*, 2011, pp. 664–669.

[29] W. Li, Y. Lin, and D. Z. Pan, "elfPlace: Electrostatics-based Placement for Large-Scale Heterogeneous FPGAs," in *Proc. ICCAD*, 2019, pp. 1–8.

[30] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Abacus: Fast legalization of standard cell circuits with minimal movement," in *Proc. ISPD*, 2008, p. 47–53.

[31] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, "How to escape saddle points efficiently," in *Proc. ICML*, 2017, p. 1724–1732.

[32] K. Dai, W. Liu, and Y. Li, "NCTU-GR: Efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-d global routing," *IEEE TVLSI*, vol. 20, no. 3, pp. 459–472, 2012.

[33] M. Kim, J. Huj, and N. Viswanathan, "ICCAD-2014 CAD contest in incremental timing-driven placement and benchmark suite: Special session paper: CAD contest," in *Proc. ICCAD*, 2014, pp. 361–366.

[34] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan, "ABCDPlace: Accelerated Batch-based Concurrent Detailed Placement on Multi-threaded CPUs and GPUs," *IEEE TCAD*, pp. 1–1, 2020.