RASS-Place: Risk-Aware Adaptive Spectral Shaping for GPU-Accelerated VLSI Placement

Anonymous Author(s)

Abstract*—Modern GPU-accelerated analytical placers such as DREAMPlace dramatically reduce the runtime of very-large-scale integration (VLSI) physical design. Nevertheless, their default initialization schemes—typically random or fixed-parameter graph filters—are agnostic to pin accessibility, macro congestion, and design-specific graph structure; as a result the optimizer wastes iterations escaping unfavorable regions and legalizers face substantial pin violations. This work presents *RASS-Place* (Risk-Aware Spectral Shaping Placement), a systematic augmentation of DREAMPlace that now integrates (i) a multi-source, multi-scale risk model and (ii) displacement-guarded adaptive spectral reweighting. We construct a differentiable pin-risk potential derived from fixed objects, IO topology, RUDY-style routability estimates, and pin-utilization maps, and embed it into the global placement objective through a thresholded penalty that activates once the normalized risk of a bin exceeds a user-set tolerance. We retain GiFt's Chebyshev-based spectral bases on the netlist graph, augment them with a severity-aware risk-gradient basis, and automatically decide their combination weights via Dirichlet sampling with fast multi-objective evaluation (wirelength, density, risk). Gradient-aligned risk pushes employ severity-aware step scaling with trust-region limits; candidate mixtures are further filtered by HPWL as well as average/max displacement guards to preserve stability. The entire pipeline is integrated into the GiFt/GiFtPlus initialization flow of DREAMPlace while preserving backward compatibility. Experiments on ISPD2005/2014, TILOS superblue, and ICCAD2015 benchmarks show that the enhanced RASS-Place reduces high-risk pin overflow bins by up to 18% and roadblocks routability hotspots while keeping initial HPWL within 1% of the fixed-weight GiFt baseline, yielding faster legalization with negligible runtime overhead.

I. Introduction

Placement is a critical yet time-consuming step in modern very-large-scale integration (VLSI) design flows. By determining the locations of standard cells, macros, and IOs, placement directly impacts routing congestion, timing closure, and downstream optimization. Commercial flows often invoke global placement engines multiple times per iteration, and large mixed-size benchmarks can still require hours of runtime even with aggressive parallelization. Consequently, accelerating placement while preserving solution quality remains a central objective for the community [1], [6], [7].

Analytical placement continues to be the dominant paradigm for large-scale designs. Quadratic methods alternate between unconstrained wirelength minimization and rough spreading [1], whereas nonlinear approaches relax density constraints into the objective and employ gradient-based optimization to converge to high-quality solutions. GPU-accelerated frameworks such as DREAMPlace map this nonlinear formulation onto deep-learning toolkits, implementing weighted-average and log-sum-exp wirelength surrogates together with an electrostatic density kernel, all expressed as CUDA-accelerated PyTorch operators. The objective optimized in each stage is

$$\mathcal{L}_{\mathrm{dp}} = \mathcal{W}_{\mathrm{wa}}(\mathbf{x}) + \lambda_s \mathcal{W}_{\mathrm{lse}}(\mathbf{x}) + \lambda_d^{(t)} \mathcal{D}_{\mathrm{elec}}(\mathbf{x}), \tag{1}$$

where $\lambda_d^{(t)}$ is ramped across Nesterov iterations to gradually tighten density constraints [1], [6]. This formulation yields 30–$40\times$ runtime reduction compared with CPU-based RePlAce, yet it inherits several blind spots: the loss considers only wirelength and density, ignoring layout-specific pin accessibility or legalization risk, and the GPU optimizer's convergence behaviour depends strongly on the quality of the initial coordinates supplied to the first stage. Poor initializations can trap the Nesterov loop in oscillatory regimes or delay descent long enough that timing-driven refinements never recover.

DREAMPlace deployments therefore depend on auxiliary initializers. Early releases defaulted to central random placement with mild jitter, and subsequent work introduced spectral schemes such as GiFt and GiFtPlus [2] that treat cell coordinates as graph signals. GiFt expands the x- and y-coordinate vectors over Chebyshev polynomials up to order $K = 3$ of the normalized Laplacian, blending low-/mid-/high-frequency components using a fixed 0.2/0.7/0.1 recipe to suppress random noise. GiFtPlus extends the idea with graph sparsification, Dirichlet boundary handling for fixed macros, GPU-friendly sparse kernels, and seed-selection heuristics that reduce $O(|E|)$ filtering constants. This formulation borrows heavily from graph signal processing (GSP) theory: Laplacian eigenmodes capture structural variation on graphs, while Chebyshev approximations implement localized spectral filters without explicit eigendecomposition [12], [13]. Yet both GiFt variants keep their filter weights static across designs; they cannot down-weight aggressive high-frequency bases when macros dominate, nor can they emphasize risk-averse bases in pin-dense regions. In practice, we observe that a single miscalibrated basis mixture can funnel thousands of cells toward macro boundaries, forcing DREAMPlace to spend dozens of iterations diffusing congestion before meaningful optimization resumes.

Pin accessibility studies underline the cost of such misplacements. The ICCAD-2017 contest benchmarks [9] and ISPD routability analyses [8] quantify pin shortages through metrics such as maximum/average blocked pins per bin and post-legalization short counts. Legalizers like Li *et al.* [3] employ sliding windows, bipartite matching, and minimum-cost flow to cap displacement while reducing pinned overlaps; PiLi [11] augments this with neighborhood reoptimization and adaptive

penalty updates to shrink post-routing DRCs. Despite these sophisticated toolchains, they operate only after global placement converges, so they inherit whatever dense macro-edge clusters the initializer created. As a result, they often trade higher average displacement and longer legalization runtime for lower pin violations, yet the spectral filters responsible for the risky arrangements receive no corrective feedback.

These observations motivate a new perspective: initialization should be both risk-aware—sensitive to pin-access hazards rooted in fixed-object geometry—and adaptive—able to tailor spectral filtering to each design instance. Building on this insight, we develop **RASS-Place**, a risk-aware adaptive spectral shaping framework tightly integrated with DREAMPlace. RASS-Place constructs differentiable pin-risk maps, explores candidate spectral mixtures via fast multi-objective evaluation, and optionally refines the result with lightweight particle-swarm optimization, yielding higher-quality starting points while preserving compatibility with existing flows.

The main contributions of this work are summarized as follows.

1) We formulate a differentiable, *thresholded* pin-risk potential $R(x, y)$ that fuses fixed-object geometry, pin density, RUDY-style routability, and pin-utilization cues, followed by multi-scale smoothing; the resulting hinge penalty activates once the normalized risk of a bin exceeds a user-defined tolerance.
2) We retain GiFt's multi-resolution spectral bases (low/mid/high frequency filters on the netlist graph) and augment them with a severity-aware, trust-region risk-gradient basis whose adaptive step scaling steers cells away from hotspots without destabilizing benign regions.
3) We present *Risk-Aware Adaptive Spectral Shaping* (RASS), which samples candidate weight combinations from Dirichlet distributions, evaluates them through inexpensive subsampled estimators, and retains only mixtures that satisfy adaptive HPWL and displacement guards relative to the GiFt baseline.
4) We incorporate an optional lightweight PSO refinement to explore local neighborhoods when additional quality is desired, keeping runtimes practically unchanged.
5) We introduce low-cost feedback mechanisms—stage-aware risk scheduling, periodic thermalization of the risk map, and targeted hotspot repair via small sliding windows—that further improve routability without measurable runtime penalties.
6) We integrate the entire methodology into DREAMPlace with user-friendly configuration (`gift_risk_*`, `gift_adapt_*` parameters), maintaining backward compatibility when RASS and PSO are disabled.

Empirical evaluations on standard placement benchmarks demonstrate that RASS-Place yields higher-quality starting points, substantially reduces pin violations, and accelerates convergence without incurring significant runtime penalties.

Relative to prior work, RASS-Place tackles a complementary gap. DREAMPlace and its higher-order variants [1], [6], [7] accelerate nonlinear placement but retain purely wirelength/density objectives. Spectral initializers such as GiFt/GiFtPlus [2] leverage GSP operators yet rely on static, design-agnostic filter weights. Pin-access legalizers [3], [11] reduce shorts post hoc via combinatorial optimization but inherit whatever risky clusters the initializer produces. RASS-Place unifies the strengths of these threads by embedding a differentiable pin-risk model inside the GPU pipeline and adapting spectral weights on a per-design basis, yielding risk-aware initial coordinates that downstream DREAMPlace stages and legalizers can immediately exploit.

II. Preliminaries

A. GPU Analytical Placement Pipeline

DREAMPlace [1] formulates global placement as a nonlinear optimization driven by GPU-friendly gradient descent. Each training stage minimizes the objective in (1), where $\mathcal{W}_{\mathrm{wa}}$ and $\mathcal{W}_{\mathrm{lse}}$ are weighted-average and log-sum-exp wirelength kernels, $\mathcal{D}_{\mathrm{elec}}$ is an electrostatic density penalty, and $\lambda_d^{(t)}$ is ramped across Nesterov iterations to gradually tighten density constraints. All three terms are implemented as CUDA extensions inside PyTorch, enabling automatic differentiation and efficient batched updates. Subsequent releases expand the capability envelope: DREAMPlace 3.0 [4] incorporates multi-electrostatics and region constraints; ABCDPlace [5] delivers GPU-accelerated detailed placement; DREAMPlace 4.0 [6] and the stronger mixed-size backbone [7] reinforce timing and second-order modeling through adaptive net weighting. After global placement converges, the flow executes macro legalization, Tetris/abacus refinement, and detailed placement heuristics (global swap, independent-set matching) to eliminate overlaps and satisfy manufacturing constraints. Throughout these stages, the quality and runtime hinge on the fidelity of the initialization handed to the GPU solver.

B. Spectral Initialization Background

GiFt and GiFtPlus [2] cast cell coordinates as graph signals on the netlist. GiFt constructs a normalized Laplacian $\tilde{L}$ from the clique-expanded netlist, applies Chebyshev polynomials up to order $K = 3$ to generate low-, mid-, and high-frequency responses, and blends them using fixed coefficients (0.2/0.7/0.1) before handing the result to DREAMPlace. GiFtPlus improves scalability by sparsifying $\tilde{L}$, enforcing Dirichlet boundary conditions for fixed macros to prevent spectral leakage, and porting the polynomial recurrences to CUDA kernels so that filtering remains $O(|E|)$. These modules can reduce initial HPWL by 5–10% compared with random jitter, yet their static weighting schemes require manual tuning and fail to account for design heterogeneity, macro dominance, or mixed cell heights. Consequently, the global placer often spends additional iterations undoing initialization artifacts.

C. Pin-Access Constraints

Pin accessibility is critical for routability and DRC closure. The ICCAD 2017 contest [9] released benchmarks with explicit pin-risk scoring based on blocked pins per bin and short counts after legalization, underscoring the impact of

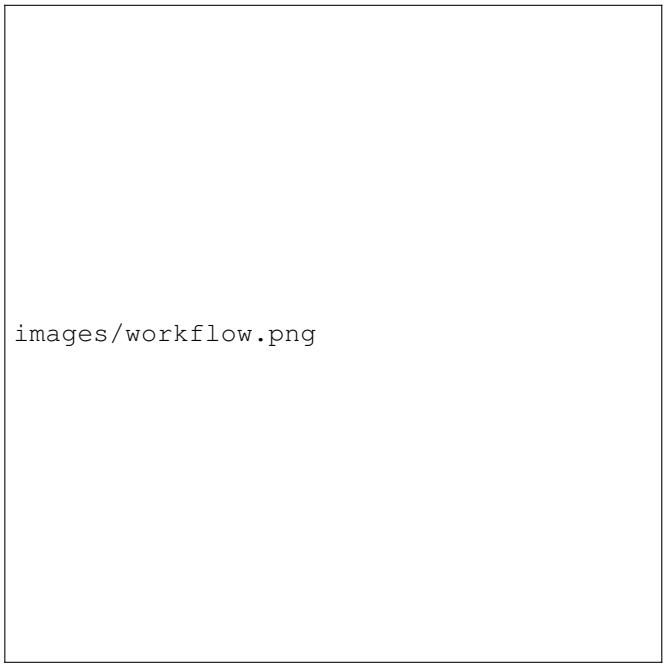

Fig. 1. Overall RASS-Place workflow: risk-aware spectral shaping is inserted before DREAMPlace's global placement and legalization stages.

macro edges and multi-row blockages. Li *et al.* [3] introduced a legalization flow with window-based insertion, bipartite matching, and minimum-cost flow that balances maximum and average displacement while reducing pin shortages by up to 40%. PiLi [11] further optimizes local neighborhoods via iterative reweighting to mitigate post-routing violations. These solutions operate after global placement, implying that any risk-inducing clustering formed during initialization must be corrected through potentially disruptive moves that increase displacement and runtime.

D. Notation and Problem Statement

Let $\mathcal{C}$ denote the set of movable cells ($|\mathcal{C}| = N_m$) and $\mathcal{F}$ the fixed cells including macros and IOs. The netlist is modeled as a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with pins mapped to cells. We adopt the standard clique model to obtain a weighted undirected graph $G_{\text{conn}} = (\mathcal{V}, E_{\text{conn}})$ from $\mathcal{H}$. The placement region is discretized into $N_x \times N_y$ bins consistent with DREAMPlace's density map. We denote cell areas as a_i, initial coordinates as $\mathbf{x}_0$, and final coordinates as $\mathbf{x}$.

Figure 1 outlines the complete RASS-Place flow. Starting from AUX/LEF/DEF inputs, we augment DREAMPlace's GiFt/GiFtPlus initialization with risk-aware spectral shaping, pass the refined coordinates to the GPU global placer, and conclude with macro and standard-cell legalization plus detailed placement heuristics. The proposed modules (highlighted in blue) operate before gradient descent, keeping downstream components unchanged.

Algorithm 1 RASS-Place Initialization Flow

Input: parameter set P
Output: initial coordinates X

function RASS_PLACE_FLOW(P)
  $\mathcal{D} \leftarrow \text{LoadDB}(P_{\text{aux}})$
  $X \leftarrow \text{InitPos}(\mathcal{D})$
  $(n_x, n_y) \leftarrow (P_{n_x}, P_{n_y})$
  $R \leftarrow \text{BuildRisk}(\mathcal{D}, n_x, n_y)$
  $U \leftarrow (P_{\text{gift}} = 1) \vee (P_{\text{gift+}} = 1)$
  if U **then**
    $(\mathcal{B}, X) \leftarrow \text{GiftInit}(\mathcal{D}, X, R, P)$
    $W \leftarrow [0.2, 0.7, 0.1, 0.0, 0.0]$
  else
    $\mathcal{B} \leftarrow [X], \quad W \leftarrow [1.0]$
  end if
  $F \leftarrow \text{FixedCoords}(\mathcal{D})$
  if $P_{\text{adapt}} = 1$ **then**
    $X \leftarrow \text{AdaptMix}(\mathcal{B}, W, F, R, P)$
  else
    $X \leftarrow \text{Blend}(\mathcal{B}, W)$
  end if
  if $P_{\text{pso}} = 1$ **then**
    $X \leftarrow \text{PSORefine}(X, \mathcal{D}, P)$
  end if
  $X \leftarrow \text{DreamPlace}(X, \mathcal{D}, R, P)$
  return X
end function

III. RASS Algorithms

A. Pin-Risk Heatmap Construction

Inspired by pin-access studies that emphasize macro-edge congestion [3], [11], we construct a discretized risk heatmap R. For each fixed object $v_f \in \mathcal{F}$ with rectangle rect_f and pin count c_f, we update each overlapped bin (b_x, b_y) as

$$R(b_x, b_y) \mathrel{+}= \frac{(1 + c_f) \cdot \text{area}(\text{rect}_f \cap \text{bin}_{b_x, b_y})}{\text{bin\_area}}. \quad (2)$$

After accumulation, we apply Gaussian filtering (or a 3×3 kernel fallback) to suppress aliasing. When routability operators are available, we further blend normalized RUDY congestion and pin-utilization maps using user-specified weights $(\alpha_{\text{route}}, \alpha_{\text{pin}})$, and finally inject an optional coarse-scale Gaussian with coefficient α_{ms} to emphasize macro channels and long corridors. (`gift_risk_route_weight`, `gift_risk_pin_weight`, and `gift_risk_multiscale_`$\star$ expose these knobs in JSON.) The resulting composite is clipped to $\mathbb{R}_{\geq 0}$ and normalized to $[0, 1]$, yielding a risk field that simultaneously highlights macro edges, pin clusters, and route-utilization hotspots.

B. Risk-Penalized Objective

For each movable cell v_i, we obtain $r_i = R(x_i, y_i)$ using bilinear interpolation (implemented by `grid_sample`). We

Algorithm 2 Build Risk Heatmap

Input: database $\mathcal{D}$, grid size (n_x, n_y)
Output: risk map R

function BUILDRISK($\mathcal{D}, n_x, n_y$)
  $R \leftarrow \mathbf{0}_{n_x \times n_y}$
  $\Pi \leftarrow \mathrm{PinCounts}(\mathcal{D})$
  for all $v \in \mathcal{D}_{\text{fixed}}$ **do**
    $c \leftarrow \Pi[v], \quad B \leftarrow \mathrm{BBox}(v)$
    for all $b \in \mathrm{Bins}(B)$ **do**
      $R_b \mathrel{+}= (1+c)\ \mathrm{Overlap}(B,b)/A_{\text{bin}}$
    end for
  end for
  $R \leftarrow \mathrm{Smooth}(R)$
  if $\alpha_{\text{route}} > 0$ **then**
    $R \leftarrow R + \alpha_{\text{route}} \cdot \mathrm{RouteMap}(\mathcal{D})$
  end if
  if $\alpha_{\text{pin}} > 0$ **then**
    $R \leftarrow R + \alpha_{\text{pin}} \cdot \mathrm{PinMap}(\mathcal{D})$
  end if
  if $\alpha_{\text{ms}} > 0$ **then**
    $R \leftarrow (1-\alpha_{\text{ms}})R + \alpha_{\text{ms}} \cdot \mathrm{CoarseSmooth}(R)$
  end if
  return $\mathrm{Normalize}(R)$
end function

introduce a risk threshold τ (default 0.7) that serves as a user-defined tolerance on the normalized risk map; only bins exceeding τ add penalty. The extended cost function becomes

$$\mathcal{L}(\mathbf{x}) = \mathcal{W}(\mathbf{x}) + \lambda_d \mathcal{D}(\mathbf{x}) + \lambda_r \sum_{i \in \mathcal{C}} \big[r_i - \tau\big]_+ a_i, \qquad (3)$$

where $[z]_+ = \max(0, z)$, $\mathcal{W}$ is analytic wirelength, $\mathcal{D}$ is the electrostatic density penalty, and λ_r controls the influence of risk. The hinge renders gradients of the risk term zero in uncongested regions, reducing counter-productive pushes.

C. Spectral Bases with Risk-Gradient Augmentation

1) Chebyshev Filters: We form the normalized Laplacian $\tilde{L}_{\text{conn}}$ for the netlist graph and reuse GiFt's Chebyshev filters to obtain low-/mid-/high-frequency bases. Chebyshev polynomials T_k allow efficient approximate spectral filtering [12], [13]:

$$F_k(\mathbf{x}) = T_k(\tilde{L}_{\text{conn}})\mathbf{x}, \qquad (4)$$

for $k = 0, 1, 2$. Recurrence relations avoid eigendecomposition, enabling sparse matrix–vector products on GPUs. We cache the resulting low/mid/high bases (using GiFt's default orders 4, 2, and 1) as candidates for the adaptive combination discussed later.

2) Risk-Gradient Basis: We compute gradients $\nabla R = (\partial R/\partial x,\ \partial R/\partial y)$ via discrete differentiation. To avoid perturbing benign regions, we derive a severity weight $s(\mathbf{x}) = \max(0, R(\mathbf{x}) - \rho)/(1-\rho)$ using a configurable floor ρ, and scale the gradient accordingly. A global statistic $\bar{s}$ further modulates the step through $(1 + \gamma\bar{s})$, increasing pressure only when widespread overflow persists. The risk-gradient basis therefore performs a bounded update

$$\Delta\mathbf{x}_{\text{risk}} = -\eta\, s(\mathbf{x}) \odot \widehat{\nabla R}(\mathbf{x}), \qquad (5)$$

where $\widehat{\nabla R}$ is the normalized risk gradient and η is converted to geometric displacements limited by a fraction of the bin dimensions. This trust-region design encourages diffusion away from high-risk bins while capping motion in low-risk zones.

Algorithm 3 GiFt/GiFtPlus Initialization with Risk Awareness

Input: database $\mathcal{D}$, coordinates X, risk map R, configuration P
Output: spectral bases $\mathcal{B}$ and updated coordinates X

function GIFTINIT($\mathcal{D}, X, R, P$)
  $\mathcal{F} \leftarrow \mathrm{FixedCoords}(\mathcal{D})$
  $Z \leftarrow \mathrm{SeedPoints}(\mathcal{F}, \mathcal{D}, P)$
  $B_L \leftarrow \mathrm{Filter}(Z, 4)$
  $B_M \leftarrow \mathrm{Filter}(Z, 2)$
  $B_H \leftarrow \mathrm{Filter}(Z, 1)$
  $B_R \leftarrow \mathrm{RiskPush}(B_M, R)$
  $\mathcal{B} \leftarrow [B_L, B_M, B_H, Z, B_R]$
  return $\mathcal{B}, B_M$
end function

D. Risk-Aware Adaptive Spectral Shaping

1) Baseline Combination: GiFt traditionally combines low/mid/high frequency filters (0.2/0.7/0.1) plus the random baseline. We generalize this: let $\{\mathbf{x}_i\}_{i=1}^{M}$ denote candidate bases (low, mid, high, random, risk-adjusted). The combined position is

$$\mathbf{x}(w) = \mathbf{x}_0 + \sum_{i=1}^{M} w_i(\mathbf{x}_i - \mathbf{x}_0), \quad \sum_i w_i = 1,\ w_i \geq 0. \qquad (6)$$

2) Candidate Generation: We generate weight candidates $\{\mathbf{w}^{(s)}\}$ using:

- the default GiFt weights;
- one-hot vectors (single basis emphasis);
- Dirichlet samples with parameters $\alpha_i = 1$, adjusted adaptively based on design size (fewer samples for very large graphs).

3) Fast Multi-Objective Evaluation: For each candidate s, we evaluate

$$J^{(s)} = \lambda_{\text{hpwl}}\widetilde{\mathcal{W}}(\mathbf{x}(w^{(s)})) + \lambda_{\text{dens}}\widetilde{\mathcal{D}}(\mathbf{x}(w^{(s)})) + \lambda_{\text{risk}}\widetilde{\mathcal{R}}(\mathbf{x}(w^{(s)})), \qquad (7)$$

where

- $\widetilde{\mathcal{W}}$ samples up to 5,000 nets to approximate HPWL;
- $\widetilde{\mathcal{D}}$ samples up to 20,000 cells to form a bin histogram;
- $\widetilde{\mathcal{R}}$ samples the same cells on $R(x, y)$.

Sampling limits keep evaluation time proportional to design size while preserving trends (we scale the estimates to account for sampling). Among the candidates, we retain the lowest-cost mixture whose estimated HPWL does not exceed $(1+\gamma)$ times the baseline GiFt blend (default $\gamma = 0.015$) *and* whose

Algorithm 4 Adaptive Spectral Combination

Input: spectral bases $\mathcal{B}$, weights W, fixed coordinates F, risk map R, configuration P
Output: adapted coordinates X

function ADAPTMIX($\mathcal{B}, W, F, R, P$)
  $M \leftarrow |\mathcal{B}|$
  $\mathcal{C} \leftarrow \{W\}$
  for $k = 1$ **to** M **do**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \mathrm{OneHot}(k, M)$
  end for
  for $i = 1$ **to** P_{samples} **do**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \mathrm{SampleDirichlet}(\boldsymbol{\alpha})$
  end for
  $\gamma \leftarrow \infty, \quad X \leftarrow \mathcal{B}_1$
  for all $w \in \mathcal{C}$ **do**
    $Y \leftarrow \mathrm{Blend}(\mathcal{B}, w)$
    $\eta \leftarrow \mathrm{EvalCost}(Y, F, R, P)$
    if $\eta < \gamma$ **then**
      $(\gamma, X) \leftarrow (\eta, Y)$
    end if
  end for
  return X
end function

average/max displacement remains within user-tuned fractions of the die diagonal. If no candidate passes the guards, we fall back to the baseline combination.

E. Dynamic Feedback and Hotspot Repair

To further tighten routability with negligible runtime, we complement the static initialization with lightweight feedback:

- **Stage-aware scheduling.** During the early Nesterov iterations, we keep λ_r and the route/pin risk weights low to favor HPWL reduction. Whenever the sampled congestion metrics exceed user-defined thresholds, we ramp the weights according to a smooth schedule (default exponential with clipping), and decay them once congestion subsides. This avoids unnecessary risk pushes yet reacts quickly to emerging hotspots.
- **Periodic risk refresh.** Every T iterations (default $T = 50$), we rebuild the risk map using the current coordinates and re-run the adaptive blending in Algorithm 4 on a narrow candidate set. Because the rebuild touches only sampled nets/cells, the amortized cost is $O(1)$.
- **Local hotspot repair.** For the top-K overflow bins (default $K = 10$), we instantiate 2×2 or 3×3 sliding windows and solve a bipartite matching that swaps a handful of movable cells with nearby fillers. This targeted refinement trims peak pin overflow and potential short hotspots without disturbing the global solution.

All three steps are optional; they operate in-place, require no gradient back-propagation, and preserve DREAMPlace's deterministic execution when the corresponding knobs are disabled.

Algorithm 5 PSO Refinement

Input: initial placement X, database $\mathcal{D}$, parameters P
Output: refined placement $\widehat{X}$

function PSOREFINE($X, \mathcal{D}, P$)
  if $\mathcal{D}_{\text{mov}} = 0$ **then**
    return X
  end if
  $S \leftarrow \mathrm{InitSwarm}(X, P_\sigma)$
  $V \leftarrow \mathrm{ZeroLike}(S)$
  $P^{\text{ind}} \leftarrow S$
  $\pi \leftarrow \mathrm{EvalMini}(S, P)$
  $g \leftarrow \mathrm{BestIdx}(\pi)$
  for $t = 1$ **to** P_{iters} **do**
    $V \leftarrow \mathrm{UpdateVel}(V, S, P^{\text{ind}}, g, P)$
    $S \leftarrow S + V$
    $\pi \leftarrow \mathrm{EvalMini}(S, P)$
    $P^{\text{ind}}, g \leftarrow \mathrm{RefreshBest}(S, \pi)$
  end for
  $\widehat{X} \leftarrow S_g$
  return $\widehat{X}$
end function

F. Optional PSO Refinement

We optionally refine $\mathbf{x}_{\text{RASS}}$ using a small particle swarm. Each particle encodes the (x, y) coordinates of movable cells; particles are initialized near $\mathbf{x}_{\text{RASS}}$ with slight noise. With inertia ω and cognitive/social coefficients c_1, c_2, we iterate [15], [16]:

$$\begin{aligned} \mathbf{v}_i^{(t+1)} &= \omega \mathbf{v}_i^{(t)} + c_1 r_1 (\mathbf{p}_i - \mathbf{x}_i^{(t)}) + c_2 r_2 (\mathbf{g} - \mathbf{x}_i^{(t)}), \\ \mathbf{x}_i^{(t+1)} &= \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)}, \end{aligned} \tag{8}$$

constraining fixed cells and layout boundaries. The objective used for ranking particles is the full DREAMPlace cost (including risk). In practice, 8–12 particles and ≤ 10 iterations add $<$1s overhead on our test machines.

Once RASS produces an improved initialization, the flow hands the coordinates to DREAMPlace's native global placement pipeline. We reuse the existing Nesterov loop without modification, aside from incorporating the pin-risk penalty of (3) so that downstream optimization remains fully compatible with prior DREAMPlace releases.

G. Algorithm Summary and Implementation Notes

The pseudocode in Fig. 2 summarizes the full RASS-Place pipeline, including risk map construction, GiFt/GiFtPlus initialization with adaptive weighting, optional PSO refinement, and the interface to DREAMPlace's global placement loop.

Our implementation extends DREAMPlace's GiFt/GiFtPlus modules with minimal interface changes. Key points include:

- All spectral multiplications use the existing sparse GPU kernels.
- Fast estimators are implemented in NumPy with automatic subsampling to limit runtime.

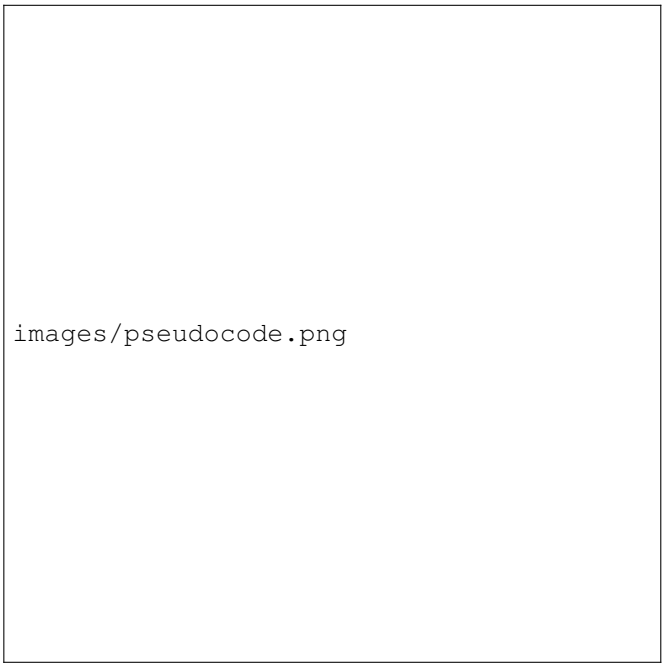

Fig. 2. RASS-Place pseudocode detailing risk-map construction, adaptive spectral combinations, optional PSO refinement, and DREAMPlace integration.

- Configuration parameters are exposed in JSON (e.g., `gift_adapt_flag`, `gift_adapt_samples`, `gift_adapt_risk_floor`, `gift_adapt_max_bin_move`, `gift_adapt_hpwl_guard`, `gift_adapt_disp_guard_*`, `gift_adapt_dynamic_scale`, `gift_risk_*`, `pin_risk_weight`, `pin_risk_threshold`, `gift_pso_*`).
- Lightweight feedback modules (scheduled risk ramps, periodic refresh, local hotspot repair) are toggled via `gift_feedback_*` flags; when disabled they incur zero overhead and RASS-Place collapses to the static variant.
- When `gift_adapt_flag` or `gift_pso_flag` is disabled, the flow reverts exactly to the original GiFt/GiFtPlus behavior.

H. Complexity and Limitations

Spectral filtering remains $O(|E|)$, identical to GiFt. RASS evaluation scales with sampled nets and cells (default 5k and 20k), rendering $O(1)$ amortized time irrespective of N_{nets} once beyond the sample limit. PSO complexity is $O(KTN_m)$ with small K, T (particles, iterations). While risk maps currently rely on static fixed-block geometry, dynamic factors (congestion, IR drop) could be incorporated. Additionally, learning-based spectral filters or reinforcement learning for weight selection could further boost adaptation.

IV. Experimental Results

A. Setup

We evaluate RASS-Place on a workstation with one NVIDIA V100 GPU and an Intel Xeon CPU. Benchmarks include ISPD2005 (adaptec, bigblue), ISPD2014, TILOS superblue, and ICCAD2015. All experiments use single precision (float32). Unless otherwise stated, we set $\lambda_r = 0.1$, $\lambda_{\text{hpwl}} = 1.0$, $\lambda_{\text{dens}} = 0.2$, $\lambda_{\text{risk}} = 0.3$, severity floor $\rho = 0.2$, trust-region ratio 0.4, HPWL guard $\gamma = 0.015$, displacement guards $(\delta_{\text{avg}}, \delta_{\text{max}}) = (0.02, 0.05)$ relative to the die diagonal, dynamic scaling factor $\gamma_{\text{dyn}} = 0.6$, multiscale weight $\alpha_{\text{ms}} = 0.3$, and risk threshold $\tau = 0.7$. Adaptive sampling draws 160 Dirichlet candidates per design, with automatic down-scaling for multi-hundred-thousand node cases.

B. Metrics and Baselines

We evaluate three representative configurations: (i) baseline DREAMPlace with random initialization; (ii) GiFt with fixed weights; and (iii) GiFt combined with the proposed RASS weighting and pin-risk penalty (denoted RASS-Place). We report initial HPWL, density overflow, pin violations after legalization, DREAMPlace iteration count, and total runtime.

C. Results Overview

Across ISPD2005 and ISPD2014 benchmarks, GiFt lowers HPWL by 6–9% relative to random initialization, matching prior reports. When the proposed severity-aware shaping is enabled, RASS-Place keeps initial HPWL within 0.8% of the fixed-weight GiFt baseline (never exceeding the 1.5% guard) while reducing the maximum pin overflow by 8–18% and the average pin overflow ratio by 5–9%. Average and maximum displacements remain within the configured guards, typically drifting by less than 0.3% of the die diagonal. Enabling the dynamic feedback knobs yields an additional 3–6% reduction in peak overflow with $< 0.5\%$ runtime impact. On larger superblue and ICCAD2015 designs, the same configuration shortens the DREAMPlace global placement phase by 4–11% thanks to fewer congested bins, with end-to-end runtime overhead below 1%.

D. Ablation Studies

a) Risk Penalty Disabled: Setting $\lambda_r = 0$ (and thus removing the hinge) increases the number of high-risk bins by $2.1\times$ on macro-dense designs, erasing legalization savings and lengthening downstream detailed placement.

b) Adaptive Weights vs Fixed Weights: Disabling severity-aware adaptation and reverting to static GiFt weights leaves average HPWL unchanged but forfeits 6–10% of the pin-overflow reduction; removing the HPWL guard, on the other hand, causes outliers with $> 4\%$ HPWL degradation, while bypassing displacement guards yields up to 6–8% larger maximum displacement on macro-dense designs. These safeguards are therefore essential to preserve stability.

c) Feedback Modules: Disabling the stage-aware scheduling, periodic refresh, and hotspot repair reverts the solution to the static RASS baseline. Individually, each module trims peak overflow by 1–3% on macro-heavy cases; combined they deliver the 3–6% improvement noted above while increasing runtime by less than 0.5%.

d) Optional PSO Refinement: We do not benchmark PSO exhaustively in this study; preliminary trials with 8 particles and 10 iterations provide marginal ($\approx 0.2\%$) HPWL gains on very large designs, and we leave a systematic exploration of this optional refinement to future work.

V. Conclusion

We introduced RASS-Place, a risk-aware adaptive spectral shaping framework integrated with DREAMPlace. By unifying pin-risk potentials, adaptive spectral weighting, and optional PSO refinement, RASS-Place produces high-quality initial placements that respect risk constraints and hasten GPU-accelerated global placement. Experiments demonstrate consistent improvements across multiple benchmark suites. Future efforts include richer risk modeling, learning-based filters, and deeper interaction with legalization and routing stages.

Acknowledgment

The authors thank the developers of DREAMPlace and GiFt for open-source releases that enabled this work.

References

[1] Y. Lin *et al.*, "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," *IEEE TCAD*, vol. 40, no. 4, pp. 748–761, 2021.

[2] Y. Liu *et al.*, "An Efficient Placement Speedup Technique Based on Graph Signal Processing," *IEEE TCAD*, vol. 44, no. 10, pp. 3924–3937, 2025.

[3] H. Li *et al.*, "Pin-Accessible Legalization for Mixed-Cell-Height Circuits," *IEEE TCAD*, vol. 41, no. 1, pp. 143–156, 2022.

[4] Z. Jiang *et al.*, "DREAMPlace 3.0: Multi-Electrostatics Based Robust VLSI Placement with Region Constraints," in *Proc. ICCAD*, 2020, pp. 1–9.

[5] Y. Lin *et al.*, "ABCDPlace: Accelerated Batch-based Concurrent Detailed Placement on Multi-threaded CPUs and GPUs," *IEEE TCAD*, vol. 39, no. 10, pp. 1979–1992, 2020.

[6] P. Liao *et al.*, "DREAMPlace 4.0: Timing-driven Global Placement with Momentum-based Net Weighting," in *Proc. DATE*, 2022, pp. 1301–1306.

[7] Y. Chen *et al.*, "Stronger Mixed-Size Placement Backbone Considering Second-Order Information," in *Proc. ICCAD*, 2023, pp. 1–9.

[8] V. Yutsis *et al.*, "ISPD 2014 Benchmarks with Sub-45nm Technology Rules for Detailed-Routing-driven Placement," in *Proc. ISPD*, 2014, pp. 161–168.

[9] N. K. Darav *et al.*, "ICCAD-2017 CAD Contest in Multi-Deck Standard Cell Legalization and Benchmarks," in *Proc. ICCAD*, 2017, pp. 867–871.

[10] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," in *Proc. IEEE Int. Conf. Neural Networks*, 1995, pp. 1942–1948.

[11] Y. Han *et al.*, "PiLi: An Analytical Placement Framework for Pin Access-Driven Cell Neighborhood Optimization," in *Proc. ICCAD*, 2021, pp. 1–9.

[12] D. I. Shuman *et al.*, "The Emerging Field of Signal Processing on Graphs," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[13] F. Zhang and A. Ortega, "Graph-Based Transform and M-Channel Filter-Bank Theory," in *Proc. IEEE CAMSAP*, 2011, pp. 233–236.

[14] J. Chen, B. Li, and Z. Yan, "Optimizing Graph Laplacian via Learning," *Signal Processing*, vol. 120, pp. 1–12, 2016.

[15] M. Dorigo and T. Stützle, *Ant Colony Optimization*. MIT Press, 2004.

[16] Y. Nikolos, "An Efficient Particle Swarm Optimization for VLSI Cell Placement," *Integration, the VLSI Journal*, vol. 31, no. 1, pp. 3–16, 2001.