Thinkorplace: Smooth Proposals and Exact Local Search for Chip Macro Placement

Brendan Murrell* Xiaofei Zhang Andrew Murrell
*Lead and corresponding author, https://brendanmurrell.com

Technical report 1.2 · 15 November 2026 · Code: https://github.com/brmurrell3/macro-placer

Abstract

Macro placement decides where a chip's large blocks sit before its wires are routed. Thinkorplace, our entry in the Partcl/HRT Macro Placement Challenge 2026, ranked ninth on the final leaderboard (verified mean proxy 0.9974; the organizers report 132 verified eligible submissions) over 17 IBM designs. That is 31.6% below the published RePlAce baseline and 4.9% above the winner, at a listed 12 minutes per design against a one-hour limit. The solver separates proposing moves from accepting them. Gradient descent on a smooth surrogate of wirelength, density and routing congestion moves all blocks at once, and coordinate descent then keeps a move only if the exact score falls. An incremental evaluator makes each exact check take milliseconds instead of seconds. We show why the no-overlap rule keeps one-block moves near their starting arrangement, bound the effect of rounding error on exact acceptance, and give a two-block case in which an optimal reassignment raises wirelength. A post-competition rerun of the configuration most likely ranked, on a laptop CPU, gives a mean proxy of 0.98121 with every placement valid.

1 Introduction

A chip design starts as a *netlist*: components with connection points called *pins*, and *nets*, the groups of pins that must be wired together. The largest components are *macros*, pre-designed blocks such as memories, each a fixed rectangle far larger than a logic gate. Around them sit thousands of small *standard cells*, which the benchmarks group into *soft clusters* that may overlap one another. *Macro placement* decides where the macros go, and in this competition the soft clusters as well (Figure 2). Connected blocks should sit close together so that wires are short, but not so close that the wires between them cannot all be routed.

The Partcl/HRT Macro Placement Challenge 2026 [27] used 17 public designs from the ICCAD04 IBM suite [1], each with 178–786 hard macros and 894–1,985 soft clusters. Entries were ranked by a

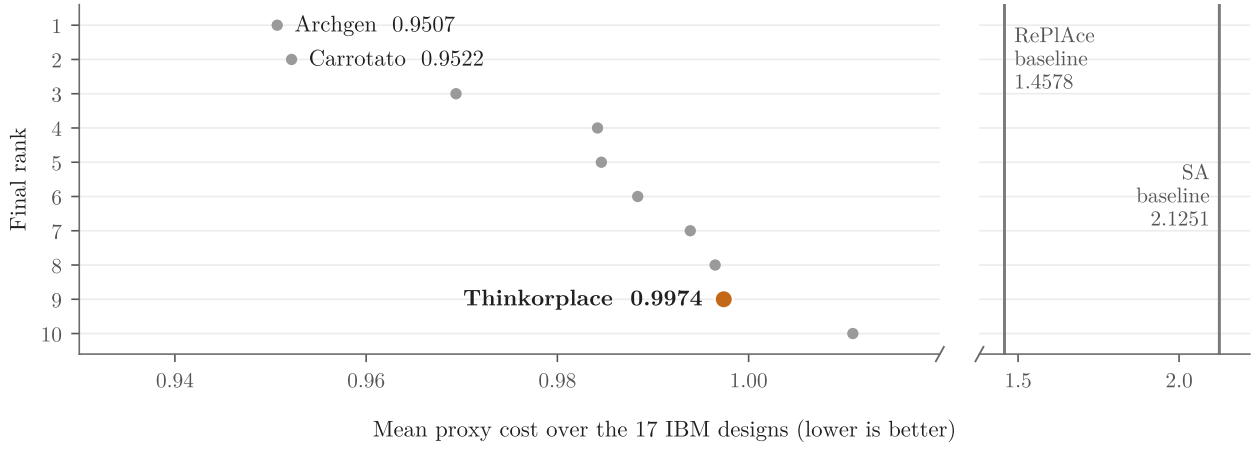


Figure 1: Mean proxy over the 17 IBM designs for the top ten of the final leaderboard and the published RePlAce and simulated-annealing (SA) baselines (Official; broken axis; lower is better).

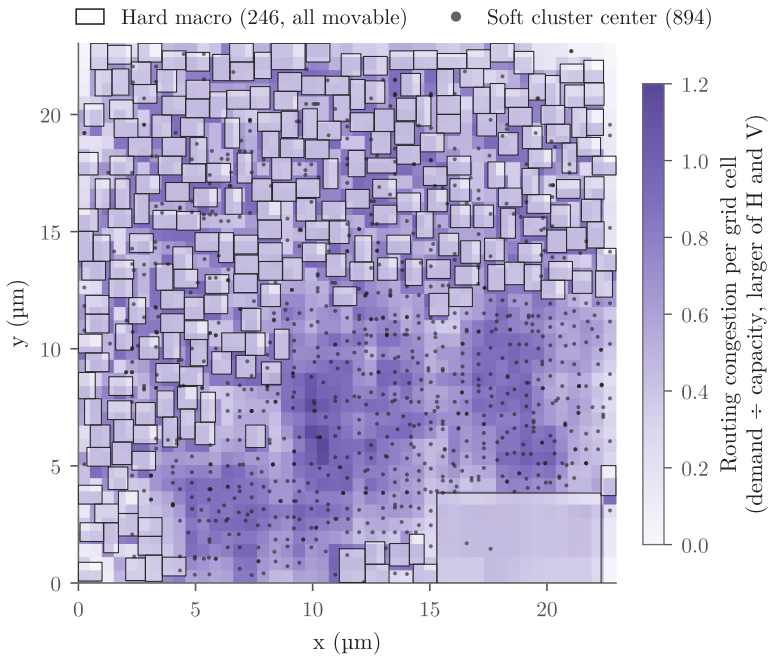


Figure 2: IBM01, the smallest design, with 246 hard macros (outlined) and 894 soft-cluster centers on a $22.95 \times 23.04\,\mu$m canvas, shaded by the evaluator's routing congestion per grid cell (darker is more congested). The placement is from a research run of our earlier v1 pipeline (Section 3.5), proxy 0.84528, rescored for this report (MEASUREMENT); the judged entry's coordinates were not saved.

proxy score, a fast stand-in for the quality of the routed chip: a weighted sum of wirelength, density and routing congestion, computed by an open evaluator [33] that implements the Circuit Training proxy [24]. Lower is better. Any overlap between two hard macros disqualified an entry, and each design had a one-hour limit.

A smooth approximation of the score can be minimized by gradient descent, which moves thousands of coordinates together, but it is only an approximation. The exact score is what the ranking uses, but it has kinks and jumps and is best used to judge one move at a time. Our solver uses the first to propose and the second to decide. A differentiable *surrogate* moves all blocks together; a *legalizer* then moves overlapping hard macros to nearby free spots; and *coordinate descent*, which moves one block along one axis at a time, keeps a move only if the exact score decreases. Rescoring only what a move changes makes each exact check about 10^3–2×10^4 times cheaper than recomputing the score, and that is what makes the last stage affordable.

Each result carries one of four labels, and Section 6 collects the caveats. OFFICIAL marks the organizers' final leaderboard. HISTORICAL marks May 2026 development records, which differ in hardware, budget and concurrency and mostly lack saved coordinates. MEASUREMENT marks post-competition diagnostics, and REPRODUCTION the post-competition rerun of the configuration most likely ranked (Section 4.2).

2 The problem as optimization

2.1 Variables and constraints

The canvas is the rectangle $[0, L_x] \times [0, L_y]$. Object i, a hard macro or a soft cluster, has center $p_i = (x_i, y_i)$, width w_i and height h_i; sizes and orientations are fixed. A placement p stacks all centers into one vector, of 2,156–5,536 coordinates on the IBM designs. Every object must lie inside the canvas,

$$\frac{w_i}{2} \le x_i \le L_x - \frac{w_i}{2}, \qquad \frac{h_i}{2} \le y_i \le L_y - \frac{h_i}{2}, \tag{1}$$

and fixed objects keep their positions (no IBM design has any). Soft clusters may overlap anything, but two hard macros $i \neq j$ must be separated horizontally or vertically, so at least one of four linear inequalities must hold:

$$\left(x_i + \tfrac{w_i}{2} \le x_j - \tfrac{w_j}{2}\right) \vee \left(x_j + \tfrac{w_j}{2} \le x_i - \tfrac{w_i}{2}\right) \vee \left(y_i + \tfrac{h_i}{2} \le y_j - \tfrac{h_j}{2}\right) \vee \left(y_j + \tfrac{h_j}{2} \le y_i - \tfrac{h_i}{2}\right). \tag{2}$$

Touching edges are legal. Fixing one of the four options for every pair turns the rule into ordinary linear inequalities. Let $\mathcal{B}$ be the set of placements that satisfy (1) and the fixed positions, $\mathcal{E}$ the set of unordered hard-macro pairs with $m = |\mathcal{E}|$, and $H_{e,b}$ the closed halfspace given by branch $b \in \{1,2,3,4\}$ of (2) for pair e.

Proposition 2.1 (Finite polyhedral decomposition)**.** *The feasible set is a union of at most 4^m compact convex polyhedra:*

$$\mathcal{F} = \mathcal{B} \cap \bigcap_{e \in \mathcal{E}} \bigcup_{b=1}^{4} H_{e,b} = \bigcup_{\sigma \in \{1,2,3,4\}^{\mathcal{E}}} P_\sigma, \qquad P_\sigma = \mathcal{B} \cap \bigcap_{e \in \mathcal{E}} H_{e,\sigma(e)}. \tag{3}$$

The polyhedra may overlap or be empty.

The proof is in Appendix D. Each piece P_σ fixes which side of each other the two macros of every pair lie on (Figure 3b). IBM01's 246 hard macros form $m = 30{,}135$ pairs, so there are up to $4^{30{,}135}$ pieces. Within a piece the constraints are linear and continuous optimization works. Changing a pair's arrangement means moving to another piece, and a path that never overlaps can do that only where two pieces touch. Along a single coordinate, with the others fixed, the legal values form a finite union of closed intervals (Appendix D), with gaps where other blocks are in the way.

2.2 The score

The competition proxy, introduced with Circuit Training [24] and computed by the TILOS MacroPlacement evaluator [33], is

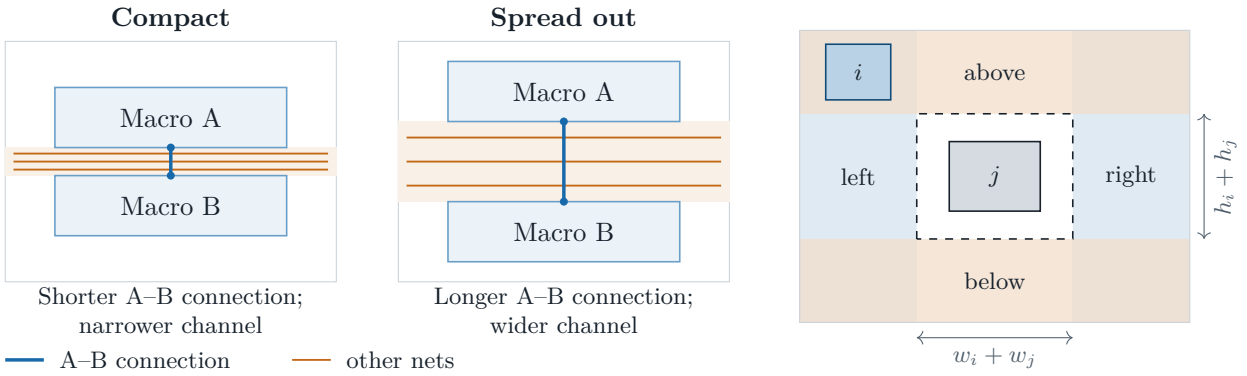
$$J(p) = W(p) + \tfrac{1}{2} D(p) + \tfrac{1}{2} C(p), \tag{4}$$

with every constant given in Appendix B.

Wirelength. The half-perimeter wirelength (HPWL) of a net n is half the perimeter of the smallest rectangle around its pins, a standard estimate of the wire needed to connect them. With each pin q at a fixed offset from its object's center and net weights ω_n,

$$\mathrm{HPWL}_n = \left(\max_{q \in n} x_q - \min_{q \in n} x_q\right) + \left(\max_{q \in n} y_q - \min_{q \in n} y_q\right), \qquad W = \frac{\sum_n \omega_n \, \mathrm{HPWL}_n}{(L_x + L_y) \sum_n \omega_n}.$$

Density and congestion. Both use a grid of G cells (810 to 2,255 per design) and both average the worst cells. For N cell values $z \in \mathbb{R}^N$, let $A_k(z)$ be the mean of the k largest. With ρ_b the total object area in cell b divided by the cell area (it exceeds 1 where objects overlap), $D = \frac{1}{2} A_{\lfloor 0.1G \rfloor}(\rho)$ with $N = G$, so the raw top-decile mean enters J with weight $\frac{1}{4}$. *Routing congestion* compares wiring demand with the routing tracks available. The evaluator maps each pin to the grid cell that contains it, routes each net between those cells with simple L- and T-shaped paths, divides each cell's horizontal and vertical demand by its capacity, spreads it over the cell and up to two neighbors on each side along one axis, and adds a fixed allowance for macros covering the cell. With $u \in \mathbb{R}^{2G}$ the horizontal and vertical values, $C = A_{\lfloor 0.05 \cdot 2G \rfloor}(u)$ with $N = 2G$, the mean of the worst 5%.



(a) Why the terms compete (schematic). **(b)** Where i's center may go (j fixed).

Figure 3: (a) More separation leaves room for wires crossing between two blocks but lengthens their connection (schematic; macro interiors obstruct routing). (b) Each branch of (2) confines i's center to a halfplane left of, right of, below or above j. Their union, the plane minus the dashed box, is not convex, and the corners satisfy two branches.

Lemma 2.2 (Tail representation and stability)**.** *For $z \in \mathbb{R}^N$ and an integer $1 \le k \le N$,*

$$A_k(z) = \max_{\substack{0 \le \beta_j \le 1/k \\ \sum_j \beta_j = 1}} \beta^\top z = \min_{\theta \in \mathbb{R}} \Big\{ \theta + \frac{1}{k} \sum_{j=1}^{N} (z_j - \theta)_+ \Big\}, \tag{5}$$

where $(a)_+ = \max\{a, 0\}$. A_k is convex and piecewise affine, and for any $\delta \in \mathbb{R}^N$,

$$|A_k(z+\delta) - A_k(z)| \le \min\Big\{ \|\delta\|_\infty,\ \frac{\|\delta\|_2}{\sqrt{k}},\ \frac{\|\delta\|_1}{k} \Big\}. \tag{6}$$

The threshold form is the conditional value-at-risk formula [29]; the proof is in Appendix D. The tails are convex in the cell values but kinked wherever the set of worst cells changes, and they are not convex in block positions, because cell values depend nonlinearly on positions. The ℓ_1 bound is the one the solver relies on: a move that changes few cells can shift the crowding terms only a little. The first sampled IBM01 move changed 12 of 1,845 cells and moved D by 0.0025, within its bound of 0.0153 (Measurement). The incremental evaluator of Section 3.1 exploits the same locality.

2.3 Why it is hard

The legal set is nonconvex and disconnected, so good placements in different pieces can be separated by overlapping states that no legal path crosses. The score is piecewise: pin extremes, clamps and top-k selection leave kinks, and because the congestion estimate assigns each pin to a grid cell, the score jumps whenever a pin crosses a grid line, which leaves its routing part with zero gradient almost everywhere. It is also expensive. One full evaluation with the reference evaluator took 1.4 s on IBM01 and 27 s on IBM10 (Measurement; Table 12), or about 2,600 and 130 evaluations per hour. Finally, the terms conflict. Thousands of coordinates interact through 7,269–83,132 nets, and packing connected blocks together shortens wires but narrows the routing channels between them (Figure 3a). Weighted congestion is about two-thirds of the score in every development record (Historical).

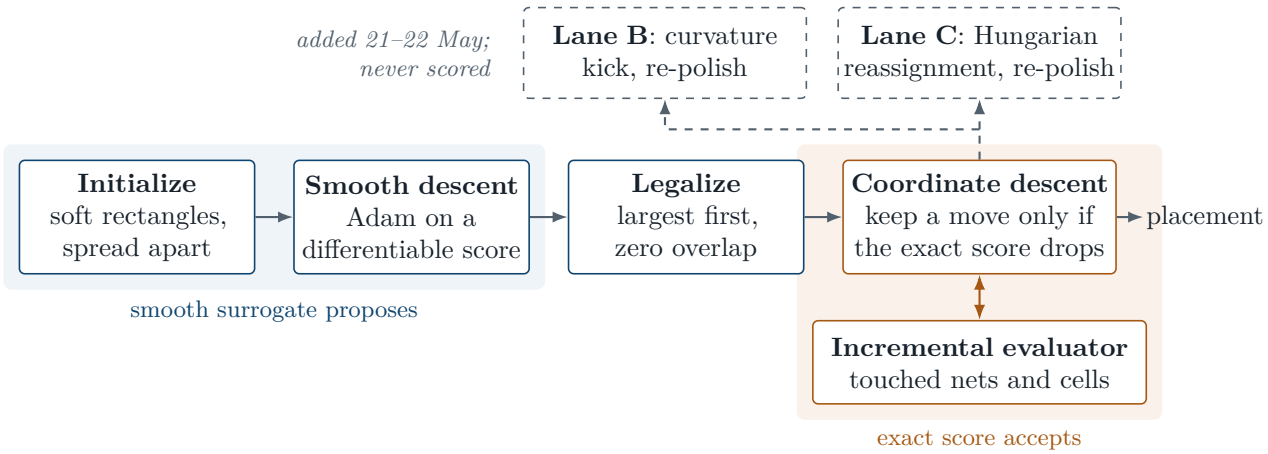


Figure 4: The solver. The smooth surrogate proposes (blue) and the exact, incrementally computed score accepts (orange). We infer that the ranked entry ran the solid pipeline; by the same inference, the dashed Lanes B and C, committed before the deadline, were never scored. The earlier v1 pipeline (Section 3.5) is not shown.

3 Approach

Figure 4 shows the solver. We describe the exact evaluator first, because the rest of the design depends on its speed.

3.1 Making the exact score cheap

The evaluator caches pin positions, net bounding boxes, cell densities, each net's routing contribution and the macro allowances. Moving object i changes only its pins, its nets, the cells under its footprint and the cells along those nets' routes, so the evaluator subtracts the old contributions, adds the new ones, re-reads the tails, and can revert. On 14 moves from IBM01 and IBM10, an update took 1.3–3.7 ms against 1.4–26.9 s for full recomputation with the reference Python evaluator (Measurement), roughly 10^3–2×10^4 times faster. A 600 s pass can therefore try about 10^5 candidates instead of 22–430. Part of the ratio comes from a faster implementation rather than from locality, and re-selecting the top k of G cells still takes time linear in the grid size. Floating-point updates do not cancel exactly, so the incremental score $\widehat{J}$ can drift from J; the largest relative component discrepancy we measured was 2.6×10^{-15}.

Proposition 3.1 (Acceptance under bounded score error)**.** *Let S_p and S_q be the incremental states at candidate endpoints p and q, and suppose $|\widehat{J}(S_p) - J(p)| \le \epsilon$ and $|\widehat{J}(S_q) - J(q)| \le \epsilon$. If $\widehat{J}(S_q) \le \widehat{J}(S_p) - \tau$, then*

$$J(q) - J(p) \le -\tau + 2\epsilon. \tag{7}$$

In particular, $\tau > 2\epsilon$ guarantees a strict decrease of the true score.

The proof is in Appendix D. Coordinate descent uses $\tau = 10^{-9}$, and the largest absolute discrepancy over 36 sampled endpoints was $\epsilon = 4.44 \times 10^{-16}$, so the error would have to be about a million times larger before an accepted move could raise the true score (Measurement). That margin holds on the sampled states only, and the sample exercises the single-move path, not the batched float64 path that coordinate descent uses.

3.2 Smooth proposals

The smooth stage minimizes a differentiable surrogate of (4),

$$\widetilde{J}(p) = \widetilde{W}_\gamma(p) + \tfrac{1}{2}\widetilde{D}(p) + \tfrac{1}{2}\widetilde{C}(p) + \lambda_t P_{\mathrm{ov}}(p) + \mu P_{\mathrm{out}}(p), \tag{8}$$

which smooths each kink of the score and adds penalties for hard-macro overlap (P_{ov}) and for leaving the canvas (P_{out}). It starts from a soft-rectangle initializer that spreads the hard macros apart by gradient steps; soft clusters start from the benchmark's input coordinates, which derive from a RePlAce placement (Appendix B).

Wirelength. HPWL uses max and min, which have no gradient where the extreme pin changes. Log-sum-exp replaces the maximum with a smooth one [25]: for values $x_1, \dots, x_r$ and smoothing length $\gamma > 0$,

$$\max_j x_j \;\le\; \gamma \log \sum_{j=1}^{r} e^{x_j/\gamma} \;\le\; \max_j x_j + \gamma \log r, \tag{9}$$

because the sum contains $e^{\max_j x_j/\gamma}$ and is at most r times it, and $\min_j x_j = -\max_j(-x_j)$. A large γ is smooth but biased and a small one accurate but stiff, so we anneal γ from 5×10^{-3} to 5×10^{-5} of the canvas scale.

Density. A rectangle's share of a cell changes at a rate that jumps when its edge crosses the cell boundary. We instead spread each object i of area $A_i = w_i h_i$ as a separable Gaussian with spreads $\sigma_{xi} = \max(w_i/2, \Delta_x/2)$ and $\sigma_{yi} = \max(h_i/2, \Delta_y/2)$, where $\Delta_x \times \Delta_y$ is the cell size. With Φ the standard normal distribution function, its mass in cell $b = [x_b^-, x_b^+] \times [y_b^-, y_b^+]$ is

$$m_{ib} = A_i \Big[\Phi\Big(\tfrac{x_b^+ - x_i}{\sigma_{xi}}\Big) - \Phi\Big(\tfrac{x_b^- - x_i}{\sigma_{xi}}\Big)\Big]\Big[\Phi\Big(\tfrac{y_b^+ - y_i}{\sigma_{yi}}\Big) - \Phi\Big(\tfrac{y_b^- - y_i}{\sigma_{yi}}\Big)\Big], \tag{10}$$

a smooth function of position, and $\widetilde{D}$ applies the top-decile rule to the cell densities $\tilde{\rho}_b = \sum_i m_{ib}/(\Delta_x \Delta_y)$. Top-$k$ selection and canvas clamps keep some kinks. This is the surrogate change that the development records associate with a gain (Table 2, row 9).

Congestion. RUDY spreads a net's demand uniformly over its bounding box [31]. The evaluator routes along rows and columns instead, and our surrogate follows it: each source–sink pin pair adds a horizontal stripe along the source's row and a vertical stripe along the sink's column, with soft (sigmoid) cell membership so the stripes move smoothly with the pins. The macro allowance, smoothing and top-5% tail follow the evaluator.

Overlap, optimizer and legalization. The weight λ_t on the penetration-area penalty P_{ov} rises over the first 70% of the steps, so the surrogate first finds an arrangement and then pushes blocks apart. Adam [13] takes 500 steps. Legalization then removes the remaining overlap while moving blocks as little as possible: in order of decreasing area, each hard macro stays where it is if that spot is free and otherwise moves to the nearest free one (Appendix C.3).

3.3 Exact local search

Coordinate descent [34] visits every movable object, hard macros and soft clusters alike, in random order. For each coordinate it takes the legal interval around the current position, bounded by the nearest blocking macros, and scores up to ten candidates there in one batched incremental call: the ends and midpoint, the current position, the nearest grid-line crossings, and targets suggested by the object's nets. The best is kept if it lowers the score by more than $\tau = 10^{-9}$. A pass stops at its time cap or, after half of it, once five consecutive sweeps each gain less than 0.001. A one-coordinate move can change a pair's arrangement only at a point of contact, so coordinate descent stays near the arrangement it starts from, and the quality of the smooth start matters (Section 4.4).

3.4 Escaping plateaus

When coordinate descent stalls, further gains need many blocks to move at once. We tried two ways of proposing such moves. Every candidate is legalized, re-polished by coordinate descent and kept only if the exact score drops.

Negative curvature. Near a placement p, a smooth function f behaves along a unit direction v like its second-order Taylor expansion,

$$f(p+tv) \approx f(p) + t\,\nabla f(p)^\top v + \tfrac{1}{2}t^2\,v^\top \mathbf{H} v, \qquad \mathbf{H} = \nabla^2 f(p). \tag{11}$$

Where small moves stop helping, the gradient term is small. If the curvature $v^\top \mathbf{H} v$ is negative, the surface bends down along v and a step of moderate length lowers f. The most negative curvature is the smallest eigenvalue of $\mathbf{H}$, which has about 5.2 million entries on IBM01. We never form it. Automatic differentiation gives a Hessian-vector product $\mathbf{H}v$ for a small multiple of the cost of a gradient [28], and the Lanczos method [18] uses a few dozen such products to approximate the extreme eigenvalues.

Proposition 3.2 (Signed curvature decrease)**.** *Let f be twice continuously differentiable on an open set U containing p, and let v be a unit vector with $v^\top \nabla^2 f(p) v \le -\eta < 0$. Choose $s \in \{-1, 1\}$ with $s\,\nabla f(p)^\top v \le 0$. Fix $t > 0$, and suppose the segment $\{p + usv : 0 \le u \le t\}$ lies in U and the Hessian is L-Lipschitz on it (Euclidean norm for vectors, induced norm for matrices). Then*

$$f(p+tsv) - f(p) \le -\frac{\eta}{2}t^2 + \frac{L}{6}t^3. \tag{12}$$

For $L > 0$ and $0 < t \le 3\eta/(2L)$ this is at most $-\eta t^2/4$; for $L = 0$ the bound is $-\eta t^2/2$ throughout the segment.

The proof is in Appendix D. The kicks we ran do not inherit this guarantee. Legalization moves the endpoint after the step, and f was an older surrogate whose density and congestion terms multiply horizontal by vertical overlap lengths, which builds in negative-curvature directions. At the diagnosed IBM01 endpoint that surrogate also changes branch within $10^{-7}\,\mu$m, far shorter than the 0.3–3 μm steps, so the Lipschitz assumption fails (Appendix E).

Assignment. Some gains need two blocks to trade places, which coordinate descent cannot do without passing through an overlap. The assignment move draws $K = 50$ macros at random from the 150 with the highest connectivity scores and takes 100 candidate slots that include the macros' current centers, so staying put is always possible. It scores each single move against a frozen background, $\Delta_{ij} = J(p^{i \leftarrow j}) - J(p^0)$, where $p^{i \leftarrow j}$ moves only macro i to slot j, and the Hungarian method [15] solves

$$\min_z \sum_{i,j} \Delta_{ij} z_{ij} \quad \text{subject to} \quad \sum_j z_{ij} = 1, \quad \sum_i z_{ij} \le 1, \quad z_{ij} \in \{0, 1\} \tag{13}$$

exactly, in time cubic in K. The moves are applied in order of predicted gain, skipping any that would overlap, and the batch is kept only if the incremental score falls by more than 10^{-7}.

The matrix sees each move alone, but moves interact (Figure 5). Two macros at $x_1 = 2$ and $x_2 = 12$ share a weight-2 net, and each has a unit-weight net to a fixed terminal at its start, so up to normalization their wirelength is

$$F(x_1, x_2) = 2|x_1 - x_2| + |x_1 - 2| + |x_2 - 12|. \tag{14}$$

For slots 3 and 11 the Hungarian method picks $(x_1, x_2) = (11, 3)$ with a predicted change of -18, but $F(2, 12) = 20$ and $F(11, 3) = 34$, so the real change is $+14$. Every placement here is legal for macros narrower than one unit. In the full score the moves also interact through the top-k tails. Independent-set matching avoids the wirelength part of this error by matching only cells that share no nets [4]. Our move picks connected macros on purpose and relies on the exact recheck.

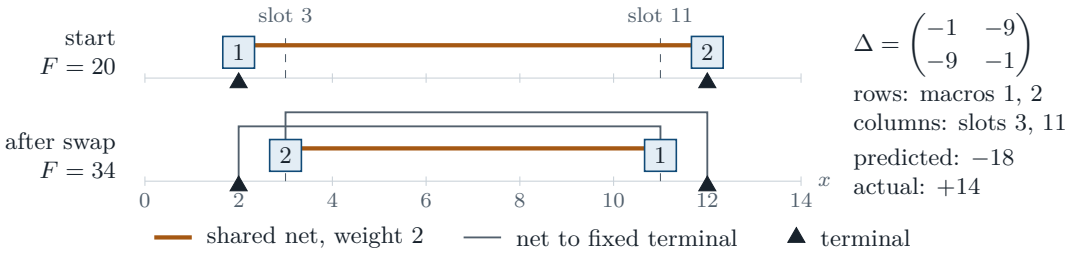


Figure 5: Each of the two crossing moves saves 9 on its own, so the matrix Δ predicts -18 for the swap. Made together, the moves cross and wirelength rises from 20 to 34.

3.5 What was scored

We did not record which commit the organizers ran. From the listed 12 minutes per design and the 20 May resubmission date (Appendix G), we infer that the entry verified at 0.9974 ran the solid pipeline of Figure 4, which we call the *single pipeline*, with a 720 s budget of which up to 600 s went to coordinate descent. Its smooth descent runs in single precision on a CUDA GPU, an Apple GPU or the CPU, whichever is available, so the trajectory depends on the device, and the device used for the judged run is not recorded.

On 21–22 May, before the deadline, we committed a three-lane system. Lane A is the same pipeline with a longer budget (2,700 s, up to 900 s of coordinate descent), Lanes B and C add the curvature kicks and the assignment batches, and the lowest exact score with zero overlaps wins. By the same inference this system was never scored, so 0.9974 says nothing about it.

Our earlier entry, v1 (13 May), polished two initial placements in turn with coordinate descent, large-neighborhood search and simulated annealing [14], then ran a curvature cascade. A 17 May run of a later v1 variant took about 53 minutes per design.

4 Results

4.1 Competition result

Our entry ranked ninth on the final leaderboard (verified mean proxy 0.9974; the organizers report 132 verified eligible submissions), with zero overlaps and per-design scores from 0.7829 to 1.2602 (Table 1 and Figure 1; Official). That is 31.6% below the published RePlAce baseline [7] and 4.9% above the winner. Ranks four to nine lie within 1.4% of one another. The listed 12 minutes per design is a fifth of the allowed hour; among the top-ten entries that list a runtime, only Vibe lists a shorter one (516 s, on its own hardware). Before the 20 May resubmission the leaderboard listed our 13 May entry (v1) at 1.0771, which is not a verified result.

Table 1: Official, final leaderboard (14 August 2026). Mean proxy cost over the 17 IBM designs; lower is better. The organizers report 132 verified eligible submissions. Runtimes are as listed by each entrant on its own hardware; the last column is the change relative to the RePlAce mean.

Rank	Entry	Mean proxy	Listed runtime	vs RePlAce
1	Archgen	0.9507	—	−34.8%
2	Carrotato	0.9522	∼12.4 min per design	−34.7%
9	**Thinkorplace**	**0.9974**	**12 min per design**	**−31.6%**
—	RePlAce baseline	1.4578	—	—
—	SA baseline	2.1251	—	+45.8%

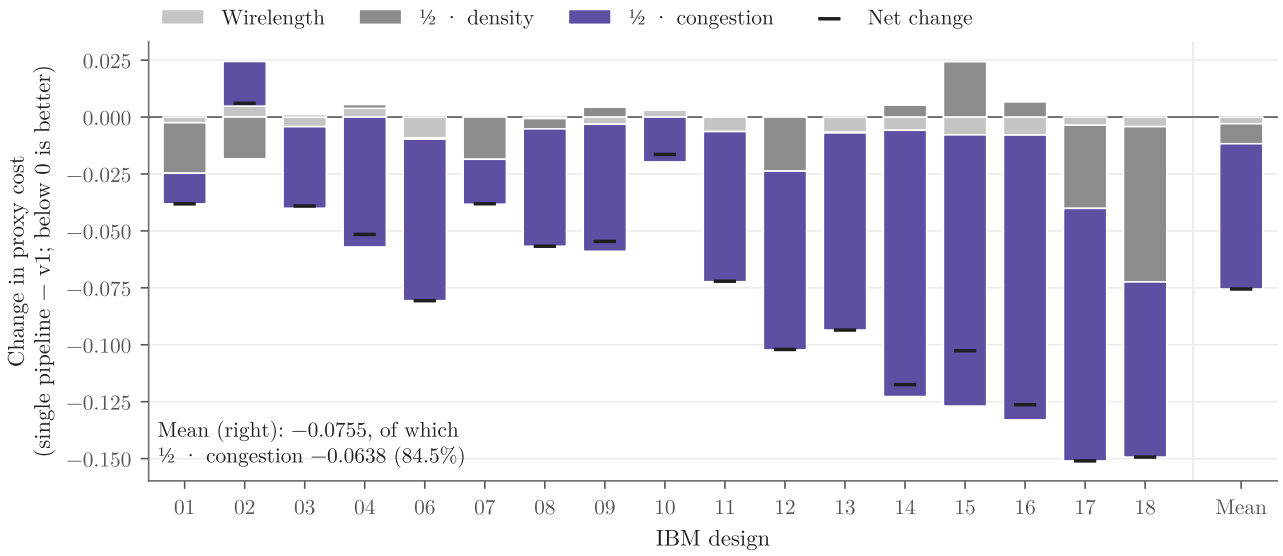


Figure 6: Change in each weighted score component per design, from the v1 record to the 0.98199 record (Historical). Bars below zero are improvements; congestion supplies 84.5% of the mean decrease.

4.2 Reproduction on a laptop CPU

After the competition we reran the configuration most likely ranked, at its 720 s budget, on all 17 designs in turn on an Apple M3 Max laptop CPU under the one-hour cap (Reproduction). The mean proxy is 0.98121; all 17 placements are valid with zero overlaps, 17 of 17 completed the full pipeline, and the median wall time is 396 s. The organizers measured 0.9974 on different hardware (our 20 May submission notes state an AMD EPYC with an NVIDIA RTX 6000 Ada GPU, which would run the descent), so the two numbers are not a matched comparison. Table 17 (Appendix G) gives each design, and Appendix F describes two runtime defects fixed before this run.

4.3 Where the gain came from

Three development records cover all 17 designs (Historical), and we name them by their mean proxy. The *v1 record* (1.05750) is a 17 May run of a later variant of the v1 pipeline, not the 13 May entry. The *early single-pipeline record* (1.00835) is a 20 May run of an early version of the single pipeline, and the *0.98199 record* is a 20 May run of the version the ranked entry most likely descends from. They differ in code, hardware, budget and concurrency, so they trace a trajectory but do not isolate causes. All three beat both published baselines on every design, by 14–46% against RePlAce (Appendix G).

From the v1 record to the 0.98199 record the mean fell by 0.07551 (7.14%), and 16 of 17 designs improved. Wirelength supplied 0.00294 of the decrease, weighted density 0.00878 and weighted congestion 0.06379, or 84.5% (Figure 6). That matches where the score sits: weighted congestion is 64.8–66.2% of the mean proxy in each record. The one regression, IBM02 (1.00659 to 1.01265), shows the tradeoff of Figure 3a, with density down but wirelength and congestion up. The decomposition says which terms moved, not which code change moved them. The smooth stage gained both Gaussian density and the routing trace over the same period, and no controlled ablation separates them.

4.4 A simpler, faster pipeline scored better

The v1 record spent 3,192 s per design on two initializers, three kinds of local search and a curvature cascade. The early single-pipeline record spent 697 s, and the 0.98199 record spent 524 s, a sixth of v1's time, for a 7.1% lower mean (Historical). The geometry of Section 2 suggests a reason. Exact local search stays near the arrangement it starts from (Section 3.3), so running more of it mostly refines that

arrangement, while a better smooth start changes the arrangement itself. Only a matched-compute comparison can test this.

5 What worked and what did not

We ran development as written hypotheses, each with a kill gate (a threshold, set in advance, for abandoning the idea) and a recorded outcome: 141 experiments, of which 39 were explicitly falsified (HISTORICAL). Table 2 summarizes ten. They are single, uncontrolled runs, so they show which ideas we kept and why rather than measure effect sizes. Kill gates and cross-design checks caught a bug (row 3) and a move that hurt industrial-style designs (row 5).

Table 2: What we tried during development (HISTORICAL). Each row summarizes single-run development records from April–May 2026. The runs are uncontrolled: hardware, time budget and concurrency varied, and most comparisons are against a separate reference run. Scores are proxy scores; lower is better. Row 1 gives the September re-measurement (MEASUREMENT) in place of the May figure. Verdict records the development decision at the time: *kept*, adopted into a later pipeline; *dropped*, abandoned after a negative result. Appendix A maps each numbered row to its record.

Idea (plain words)	What happened	Verdict
1. Rescore only what a one-block move changes, instead of recomputing the whole score	September re-measurement on two designs: 1.3–3.7 ms per move versus 1.4–26.9 s for a full rescore; largest relative difference 2.6×10^{-15}.	kept
2. Move one block at a time along one axis to its best legal position; stop when gains level off rather than at a fixed time	Mean over 17 designs 1.1055, versus 1.1193 with a fixed 600 s stop, using 69% more total time; hard designs gained 1.7–3.6%, and all 17 stopped before the one-hour cap.	kept
3. After local search stalls, accept some uphill moves at random (simulated annealing)	The first version was worse on 4 of 4 designs (by 5.7% on average) because it returned its last state rather than its best. Fixed, it gained 0.63% and became one stage of the earlier serial pipeline.	kept
4. After local search converges, take GPU gradient steps on a smooth approximation of the score to find a better nearby arrangement	0 of 350 attempts accepted on 4 designs at full budgets; a 2.16% gain seen with shortened budgets did not reappear.	dropped
5. Choose which blocks to remove and re-place by how much they add to routing congestion	Lower on 2 of 4 IBM designs (by 1.12% and 0.15%), but 1.45% worse on four open industrial-style designs (5.14% on one) and 0.39% worse over all 17 IBM designs.	dropped
6. Blend two good but different placements by moving every block partway along a straight line between them	On 2 designs, none of the 9 intermediate blends could be made overlap-free by the 50-iteration repair step; before repair, blends scored up to 61% worse.	dropped
7. Drive a mature open-source placer's optimizer (DREAMPlace) with the challenge score as its objective	On IBM01, 0.918 after repair and local search versus 0.852 for the existing pipeline (7.7% worse); the optimizer left 118 overlaps for repair to fix.	dropped
8. Make the smooth routing-congestion model agree more closely with the official scorer	On IBM17 the model's error fell from 14.4% to 8.0%, yet the final score was 0.98% worse; a separate bug fix that also reduced the error changed it by 0.12%, within noise.	dropped
9. Spread each block's area as a smooth bell-shaped (Gaussian) blur when estimating crowding, so the gradient changes smoothly	Lower on 4 of 4 crowded designs (by 2.71% on average) than the same pipeline with grid density. A later 17-design run averaged 0.98199, versus 1.05750 for the earlier serial pipeline, lower on 16 of 17.	kept
10. Jointly reassign up to 50 highly connected blocks to candidate slots with an optimal matching (the Hungarian method), then rerun local search	Lower in 5 of 5 single-design trials (0.35–1.25% against separate reference runs); part of the gain may come from the extra local-search pass. It went into an optional lane that is inferred never to have been scored.	kept

Smoothness mattered more than fidelity. A more accurate congestion model (row 8) and gradient polish of converged placements (rows 4 and 7) did not improve final scores, and an earlier smooth score that ranked saved placements well (Spearman 0.929) led descent from a good IBM01 placement to a result 81% worse after repair (HISTORICAL). The change that helped, Gaussian density (row 9), made the surface smoother; its fidelity was not measured.

Good placements can be separated by illegal states. No straight-line blend between two good placements was legal (0 of 9 on two designs; row 6). This is consistent with Proposition 2.1, which allows such gaps but does not require them.

Curvature kicks helped, for reasons we cannot pin down. In v1 research runs without the one-hour cap (HISTORICAL), curvature stages lowered the mean *plateau* score, the better of v1's two branches before any kick, from 1.08128 to 1.06119 (Appendix E.2). No design could have gotten worse, because a stage keeps its input unless a candidate scores lower, so the useful evidence is which kicks worked. Nearly all did. Of 204 first-stage attempts, 190 ended below the plateau with zero overlap, at almost the same rate along either eigenvector and at every step length. Generic perturb-and-repolish, or the extra polish alone, would produce this pattern, and the controls that would tell them apart, random-direction and zero-step kicks of equal cost, were not run. The computed curvature is also suspect. At one saved IBM01 placement the surrogate's density and congestion branch boundaries lie within $10^{-7}\,\mu$m, so the eigenvalue that automatic differentiation reports describes one smooth branch, not a bend in the score (MEASUREMENT; Appendix E).

Hungarian reassignment lowered the score in all five single-design trials (row 10), perhaps partly through its extra polish, and went into Lane C (Section 3.5).

Post-competition audit. A September audit of the competition code found ten defects, all since repaired (Appendix F). Most were in code that was never scored. Two could affect the single pipeline and cannot be ruled out, because no judged coordinates survive: float32 export can round a coordinate one step past the upper canvas edge, and input soft clusters outside the canvas were not explicitly repaired. The organizers' verification passed the entry.

6 Limitations

- The ranked configuration is inferred from its listed runtime and resubmission date. The judged commit, its coordinates and the device that ran its single-precision descent were not recorded.
- The REPRODUCTION result is one run on one host, on a CPU rather than the judges' GPU. Coordinate descent is capped by wall time, so its scores vary from run to run.
- HISTORICAL records differ in code, hardware, budget and concurrency, and most have no saved coordinates. Their component shares are arithmetic, not causal.
- By the same inference, Lanes B and C were never scored. The curvature evidence comes from uncapped runs without controls, and 12 of its 34 saved endpoints extend past a canvas edge by one single-precision rounding step (MEASUREMENT).
- Speed and error figures come from 14 moves on two designs through the single-move path, so the acceptance margin holds only on the sampled states.
- The smooth score keeps kinks from top-k selection and clamps, undercounts density at the canvas edge, and star-routes three-pin nets.
- Development reused the 17 public designs that set the ranking, and each competition design ran once. We report no place-and-route results (timing, power, area), and real flows discard the soft clusters that we place to lower the proxy.

7 Related work

The proxy and macro placers. The proxy comes from Circuit Training [24] and its open TILOS implementation [33], and the challenge's SA and RePlAce baselines [27] come from the updated assessment [7], which extends an earlier one [6]. Learned and black-box placers include MaskPlace, ChiPFormer and WireMask-BBO [16, 17, 30]; AutoDMP tunes an analytical placer for macros [2]; and RTL-MP and Hier-RTLMP use design hierarchy [11, 12], which the flat IBM designs [1] lack. We use no learned policy.

Analytical placement. Analytical placers minimize smoothed wirelength, by log-sum-exp [25] or weighted averages [10], under a differentiable density penalty: electrostatic in ePlace and ePlace-MS [22, 23], routability-aware in RePlAce [5], and GPU-accelerated in DREAMPlace and Xplace [19, 20]. Unlike RUDY's bounding boxes [31], our surrogate follows the evaluator's routing pattern, and it only proposes.

Detailed placement and escape moves. Abacus legalizes with minimal movement [32]. Detailed placers improve legal placements with global swaps [26] and independent-set matching, an assignment over cells that share no nets [4]; our assignment move drops that restriction, and Figure 5 shows why it exists. Minimum-mode and negative-curvature methods [3, 8, 9, 21] motivated the curvature kicks, which borrow their proposal direction but none of their guarantees.

8 Conclusion and future work

Thinkorplace splits macro placement into smooth global proposals and exact, incremental acceptance, and placed ninth on the final leaderboard in a listed 12 of the allowed 60 minutes per design (Official). The development records suggest that the score is mostly congestion, as was most of our improvement, that a fast pipeline from a good start beat close to an hour of elaborate search, and that curvature kicks need to be judged against cheaper perturbations. Testing these claims needs a controlled study of v1, the single pipeline and the three-lane system under matched hardware, budget and aggregate compute (51 runs with saved coordinates on the 17 designs), with random-direction and zero-step controls for the curvature kicks, a net-disjoint control for assignment, and place-and-route checks.

Acknowledgments

We thank the organizers of the Partcl/HRT Macro Placement Challenge 2026 for the benchmarks, the public evaluation harness, and the verified leaderboard, and the authors of the TILOS MacroPlacement evaluator, whose open implementation of the proxy score made exact local search possible.

References

[1] Saurabh N. Adya, S. Chaturvedi, Jarrod A. Roy, David A. Papa, and Igor L. Markov. Unification of partitioning, placement and floorplanning. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design*, pages 550–557, 2004. doi: 10.1109/ICCAD.2004.1382639.

[2] Anthony Agnesina, Puranjay Rajvanshi, Tian Yang, Geraldo Pradipta, Austin Jiao, Ben Keller, Brucek Khailany, and Haoxing Ren. AutoDMP: Automated DREAMPlace-based macro placement. In *Proceedings of the 2023 International Symposium on Physical Design*, pages 149–157, 2023. doi: 10.1145/3569052.3578923.

[3] Yair Carmon, John C. Duchi, Oliver Hinder, and Aaron Sidford. Accelerated methods for nonconvex optimization. *SIAM Journal on Optimization*, 28(2):1751–1772, 2018. doi: 10.1137/17M1114296.

[4] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1228–1240, 2008. doi: 10.1109/TCAD.2008.923063.

[5] Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, and Lutong Wang. RePlAce: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1717–1730, 2019. doi: 10.1109/TCAD.2018.2859220.

[6] Chung-Kuan Cheng, Andrew B. Kahng, Sayak Kundu, Yucheng Wang, and Zhiang Wang. Assessment of reinforcement learning for macro placement. In *Proceedings of the 2023 International Symposium on Physical Design*, pages 158–166, 2023. doi: 10.1145/3569052.3578926.

[7] Chung-Kuan Cheng, Andrew B. Kahng, Sayak Kundu, Yucheng Wang, and Zhiang Wang. An updated assessment of reinforcement learning for macro placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 45(8):3654–3668, 2026. doi: 10.1109/TCAD.2025.3644293.

[8] Weinan E and Xiang Zhou. The gentlest ascent dynamics. *Nonlinearity*, 24(6):1831–1842, 2011. doi: 10.1088/0951-7715/24/6/008.

[9] Graeme Henkelman and Hannes Jónsson. A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives. *The Journal of Chemical Physics*, 111(15):7010–7022, 1999. doi: 10.1063/1.480097.

[10] Meng-Kai Hsu, Yao-Wen Chang, and Valeriy Balabanov. TSV-aware analytical placement for 3D IC designs. In *Proceedings of the 48th Design Automation Conference*, pages 664–669, 2011. doi: 10.1145/2024724.2024875.

[11] Andrew B. Kahng, Ravi Varadarajan, and Zhiang Wang. RTL-MP: Toward practical, human-quality chip planning and macro placement. In *Proceedings of the 2022 International Symposium on Physical Design*, pages 3–11, 2022. doi: 10.1145/3505170.3506731.

[12] Andrew B. Kahng, Ravi Varadarajan, and Zhiang Wang. Hier-RTLMP: A hierarchical automatic macro placer for large-scale complex IP blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(5):1552–1565, 2024. doi: 10.1109/TCAD.2023.3346284.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

[14] Scott Kirkpatrick, C. Daniel Gelatt, Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671.

[15] Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955. doi: 10.1002/nav.3800020109.

[16] Yao Lai, Yao Mu, and Ping Luo. MaskPlace: Fast chip placement via reinforced visual representation learning. In *Advances in Neural Information Processing Systems 35*, pages 24019–24030, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/97c8a8eb0e5231d107d0da51b79e09cb-Abstract-Conference.html.

[17] Yao Lai, Jinxin Liu, Zhentao Tang, Bin Wang, Jianye Hao, and Ping Luo. ChiPFormer: Transferable chip placement via offline decision transformer. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 18346–18364, 2023. URL https://proceedings.mlr.press/v202/lai23c.html.

[18] Richard B. Lehoucq, Danny C. Sorensen, and Chao Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Society for Industrial and Applied Mathematics, 1998. doi: 10.1137/1.9780898719628.

[19] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement. In *Proceedings of the 56th Annual Design Automation Conference*, pages 117:1–117:6, 2019. doi: 10.1145/3316781.3317803.

[20] Lixin Liu, Bangqi Fu, Martin D. F. Wong, and Evangeline F. Y. Young. Xplace: An extremely fast and extensible global placement framework. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1309–1314, 2022. doi: 10.1145/3489517.3530485.

[21] Mingrui Liu and Tianbao Yang. On noisy negative curvature descent: Competing with gradient descent for faster non-convex optimization. *arXiv preprint arXiv:1709.08571*, 2017. URL https://arxiv.org/abs/1709.08571.

[22] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. ePlace: Electrostatics-based placement using fast Fourier transform and Nesterov's method. *ACM Transactions on Design Automation of Electronic Systems*, 20(2):17:1–17:34, 2015. doi: 10.1145/2699873.

[23] Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, and Chung-Kuan Cheng. ePlace-MS: Electrostatics-based placement for mixed-size circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(5):685–698, 2015. doi: 10.1109/TCAD.2015.2391263.

[24] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021. doi: 10.1038/s41586-021-03544-w.

[25] William C. Naylor, Ross Donelly, and Lu Sha. Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer. U.S. Patent 6,301,693, 2001. URL https://patents.google.com/patent/US6301693B1/en. Assigned to Synopsys, Inc.; filed 16 December 1998, issued 9 October 2001.

[26] Min Pan, Natarajan Viswanathan, and Chris Chu. An efficient and effective detailed placement algorithm. In *Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design*, pages 48–55, 2005. doi: 10.1109/ICCAD.2005.1560039.

[27] Partcl, Inc. Partcl/HRT Macro Placement Challenge 2026. GitHub repository, revision 996193c83eaa, 2026. URL https://github.com/partcleda/macro-place-challenge-2026/tree/996193c83eaad151e5ae3fb166cedbd83372d060. Final competition rules and leaderboard; accessed 2026-09-06.

[28] Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994. doi: 10.1162/neco.1994.6.1.147.

[29] R. Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2 (3):21–41, 2000. doi: 10.21314/JOR.2000.038.

[30] Yunqi Shi, Ke Xue, Lei Song, and Chao Qian. Macro placement by wire-mask-guided black-box optimization. In *Advances in Neural Information Processing Systems 36*, pages 6825–6843, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/15d6717f8bb33b3a74df26ce1eee0b9a-Abstract-Conference.html.

[31] Peter Spindler and Frank M. Johannes. Fast and accurate routing demand estimation for efficient routability-driven placement. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 1–6, 2007. doi: 10.1109/DATE.2007.364463.

[32] Peter Spindler, Ulf Schlichtmann, and Frank M. Johannes. Abacus: Fast legalization of standard cell circuits with minimal movement. In *Proceedings of the 2008 International Symposium on Physical Design*, pages 47–53, 2008. doi: 10.1145/1353629.1353640.

[33] TILOS AI Institute. MacroPlacement: Benchmarks, evaluators, and reproducible results from leading methods in open source. GitHub repository; challenge fork partcleda/MacroPlacement, revision 45a721d01dfe, 2026. URL https://github.com/partcleda/MacroPlacement/tree/45a721d01dfe56fc4800b95da52e9b4193b76d04. Accessed 2026-09-06.

[34] Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015. doi: 10.1007/s10107-015-0892-3.

Appendices

A Glossary and development chronology

The main text describes methods in plain words; these appendices also use the names under which the work was recorded, as numbered experiments run mostly between late April and 22 May 2026. The repository holds 151 experiment directories, 141 of them with a manifest recording the hypothesis, method, stopping rule and outcome; by their status fields, 39 were falsified, 24 marginal and 11 graduated, 49 were never closed, and 18 carry other labels. Table 3 defines the abbreviations and maps the main text's plain names for programs and records to the recorded ones; Table 4 then lists, in order, every experiment, pipeline and revision name that the report uses, with where its record lives.

Table 3: Abbreviations and terms used in the appendices, and the main text's plain names for programs and records. Table 4 lists the experiment, pipeline and revision names.

Term	Meaning
Official, Historical, Measurement, Reproduction	The four evidence labels: the organizers' final leaderboard; May 2026 development records; post-competition diagnostics on frozen inputs; the post-competition run of the configuration inferred to be ranked (Section 4.2)
Single pipeline	The main text's name for the inferred ranked configuration: soft-rectangle initialization, Gaussian smooth descent, legalization and CD under a 720 s budget (V4+Gaussian, commit `5485310`)
v1 record; early single-pipeline record; 0.98199 record	The main text's names for the three Historical suite records: the 17 May run of the final v1 configuration (1.05750), the 20 May early-v2 run with V3 (1.00835) and the 20 May E117 run with Gaussian density (0.98199)
CD, CD1, CD2	Coordinate descent; the first and second CD phases of a lane
LNS	Large-neighborhood search: remove a few macros and reinsert each at its best grid position
SA	Simulated annealing
LSE	Log-sum-exp, a smooth approximation of a maximum or minimum
RUDY	Rectangular uniform wire density: a net's routing demand spread evenly over its bounding box
HVP	Hessian-vector product, computed by automatic differentiation (AD)
J_B	The legacy smooth surrogate whose HVPs drive the curvature stages
Plateau	The better of the two v1 branches for each design, before any curvature stage
Cascade	Repeated curvature stages, each starting from the previous stage's result
Push-apart	The pairwise overlap-removal pass applied after legalization and after every curvature step
Valid, eligible, completed	Runner fields: a valid placement passes shape, finiteness, containment, fixed-object and overlap checks; an eligible one is also returned within the external limit with matching source; a completed run finished every configured phase
C, P, R, E, T	Outcome codes: completed pipeline; valid result after partial phases; valid checkpoint recovery; error; external timeout
Strict validity	Containment checked exactly, with no tolerance at the canvas edge
Float32 step	The spacing between adjacent single-precision numbers at a given magnitude, 3.81×10^{-6} at 32 to 64 μm

Read the chronology as a lineage rather than a ranking. Exact incremental scoring and coordinate descent (E1, E9) came first; local-search refinements built the two v1 branches and their plateau (E12–E48); curvature stages followed (E74, E84); the smooth single pipeline (V3, E117, V4+Gaussian) replaced v1 and is inferred to be the ranked entry; Lanes B and C were added after it. Scores in different rows come from different hardware, budgets and concurrency. Where a value recomputed from the frozen records differs from a figure quoted in May notes, as for the first curvature stage (1.06760 frozen against 1.0666 in May), the report uses the frozen value.

Table 4: Glossary and development chronology (HISTORICAL). Dates (2026) are decision or commit dates; manifest gives the first seven characters of the experiment manifest's git blob, and runtime archive names are prefixes of each archive's source-identity SHA-256. Scores are mean proxy over the 17 IBM designs unless stated (lower is better); frozen values are recomputed from the saved records in the publication data. The judged commit was not recorded: from the listed 12-minute runtime and the 20 May resubmission date, the single pipeline (V4+Gaussian) is inferred to be the ranked entry, so Lanes A, B and C and the three-lane system, all committed before the deadline, are inferred never to have been scored. Row n refers to the numbered rows of Table 2.

Name	Date	What it is	Record
April to early May 2026			
E1	26 Apr	Incremental evaluator: after a one-block move, update wirelength, density and congestion from cached state instead of rescoring.	Row 1; manifest `478c9af`; re-measured in September
E9	27 Apr	Coordinate descent (CD), one block and one axis at a time, stopped by a plateau rule.	Row 2; manifest `9da7ae0`
E12	28 Apr	Large-neighborhood search after CD: remove a few blocks and try every grid cell for each.	manifest `97c4a47`; mean 1.0990 vs 1.1055
E14	29 Apr	Annealing polish that returned its last state rather than its best.	Row 3; manifest `fd7f50d`
E24	29 Apr	The same annealing polish with best-so-far tracking and a lower temperature.	Row 3; manifest `fa32221`
E25	29 Apr	Branch 1 of v1: soft-rectangle start (SDF), then CD, neighborhood search and annealing.	manifest `84d69da`; mean 1.0954
E41	30 Apr	Branch 2 of v1: smooth-score start (DPO), then CD, neighborhood search, annealing and three-block joint moves.	manifest `4a930aa`; mean 1.0848
E48	1 May	Best of branches 1 and 2 for each design: the plateau before the curvature stages.	manifest `8bb8dc8`; May 1.08151; frozen start points 1.08128
E53	2 May	GPU smooth-score polish after converged local search.	Row 4; manifest `0fa976e`
E54	2 May	Neighborhood search that removes the blocks contributing most to congestion.	Row 5; manifest `871381b`
E62	2 May	The challenge's reference starter placer as a third starting branch.	manifest `505095e`; 1.44% worse on 4 designs
E65	3 May	Straight-line blends between the placements of branches 1 and 2.	Row 6; manifest `1586365`
E69	3 May	Pair swaps guided by a sequence-pair encoding (left/right and above/below order of every pair).	manifest `ceee9da`; about 0.10% gain on 4 designs, below threshold
E74	4–5 May	First curvature stage: step both ways along negative-curvature directions of a smooth surrogate, then re-polish with CD.	manifest `6f75879`; frozen mean 1.06760 (May note 1.0666)
May 2026			
E84	10–11 May	Repeated curvature stages starting from the first stage's results (the cascade).	manifest `5794d74`; frozen mean 1.06119
v1	13 May	Serial pipeline: two starting branches, each with CD, neighborhood search and annealing, then the curvature cascade. Entered on 13 May (the earlier submission).	frozen 17 May run of its final configuration: mean 1.05750
stacked periphery	16–17 May	Final v1 configuration: adds curvature searches under three reweighted scores and a last step nudging blocks toward the canvas edge.	manifest `848218d`; frozen mean 1.05750
V3	19–20 May	Smooth placer: smooth wirelength, grid density and per-net routing-trace congestion optimized with Adam, then legalization and CD.	frozen EPYC run of v2 with V3: mean 1.00835

Table 4, continued

Name	Date	What it is	Record
E116	20 May	DREAMPlace's optimizer driven by the challenge score.	Row 7; manifest `db11cc2`
E122	20 May	Three-pin routing shapes in the smooth congestion model.	Row 8; manifest `06f8838`
E124	20 May	Fix for spurious demand on degenerate routes in the smooth congestion model.	Row 8; manifest `d9dc97d`
E117	20 May	Gaussian density in the smooth placer.	Row 9; manifest `b7bab52`; frozen mean 0.98199
V4+Gaussian	20 May	V3 with faster smooth-score kernels (V4) and Gaussian density, then legalization and CD; 720 s budget, 600 s of CD.	commit 5485310; inferred to be the ranked entry
v2	20–22 May	Entry name of the 20 May resubmission, ranked ninth (0.9974); the name was reused on 22 May for the three-lane system.	Official leaderboard
E159	21 May	Hungarian reassignment of up to 50 blocks, then a second CD pass.	Row 10; manifest `cab75cd`
Lane A	21–22 May	The single pipeline (smooth descent, legalization, CD) run as one lane of the three-lane system.	never scored as a lane (inferred)
Lane B	21–22 May	Lane A followed by curvature proposals and a second CD pass.	never scored (inferred)
Lane C	21–22 May	Lane A followed by Hungarian reassignment and a second CD pass.	never scored (inferred)
A+B+C	22 May	Runs the three lanes in parallel and returns the lowest full score; committed before the deadline.	commit 456f94f; never scored (inferred)
Implementation names, source revisions and runtime archives			
SDF	—	Implementation name of the soft-rectangle initializer used by branch 1 of v1 and by v2.	
DPO	—	Implementation name of the smooth-score initializer used by branch 2 of v1.	
`5485310`	20 May	Commit that made V4+Gaussian the v2 entry; the judged commit was not recorded.	git history
`2ab92a33d352`	22 May	Last competition-era commit; its source closure is the frozen competition archive.	competition-source.tar
`45a721d01dfe`	11 Jan	Pinned revision of the TILOS MacroPlacement evaluator (git submodule).	submodule pin
`996193c83eaa`	14 Aug	Upstream challenge revision carrying the final leaderboard.	upstream README
`5730d13762ce`	6 Sep	Runtime archive for the 12-run pilot.	frozen snapshot and archive
`ca4343aaefda`	6 Sep	Runtime archive for the post-pilot repair batch.	frozen snapshot and archive
`5aa4c719cf18`	6 Sep	Runtime archive with the final repairs, used for the NG45 integration check.	frozen snapshot and archive

B The canonical proxy cost

This appendix states the score exactly as the pinned evaluator computes it. The formulation follows Circuit Training [24]; the evaluator is the open TILOS `PlacementCost` class in `plc_client_os.py` (MacroPlacement revision `45a721d`); the challenge harness wraps it in `compute_proxy_cost` (`objective.py`). Function names below refer to those two files. Every rule was checked against that source for this report.

Notation. The canvas is $[0, L_x] \times [0, L_y]$, divided into a grid of n_c columns and n_r rows with $G = n_c n_r$ cells of width $\Delta_x = L_x/n_c$ and height $\Delta_y = L_y/n_r$. Object i (hard macro or soft cluster) has center $p_i = (x_i, y_i)$, width w_i and height h_i. A hard-macro pin sits at its macro's center plus a fixed offset;

a soft-cluster pin sits at the cluster center; a port (an I/O terminal on the canvas edge) has a fixed position. The grid dimensions, the canvas size and the routing constants are read from the header of each design's `initial.plc` (`restore_placement` with comment parsing). All 17 IBM headers carry the same constants: horizontal and vertical routing capacities $r_H = 65.957$ and $r_V = 106.957$ routes per μm, macro routing allocations $a_H = 30.304$ and $a_V = 71.304$ per μm, and smoothing range $s = 2$ cells. The header's overlap threshold (0.004) does not enter the score.

Table 5: Benchmark statistics of the 17 IBM designs, parsed from the challenge inputs with the canonical evaluator's loader. Nets is the evaluator's net count; the grid is the density and congestion grid.

Design	Movable hard macros	Soft clusters	Nets	Canvas $W \times H$ (μm)	Grid columns × rows
IBM01	246	894	7,269	22.95 × 23.04	45 × 41
IBM02	271	1,075	10,944	32.56 × 32.48	30 × 27
IBM03	290	1,148	10,247	35.12 × 35.04	32 × 29
IBM04	295	1,085	13,555	34.09 × 34.08	31 × 30
IBM06	178	900	14,085	32.65 × 32.64	31 × 28
IBM07	291	1,040	19,920	38.51 × 38.40	35 × 32
IBM08	301	1,030	20,694	41.05 × 40.96	38 × 34
IBM09	253	1,048	18,341	46.74 × 46.88	36 × 38
IBM10	786	1,982	39,140	77.05 × 77.12	55 × 41
IBM11	373	1,195	25,273	51.53 × 51.52	39 × 45
IBM12	651	1,985	40,996	67.97 × 68.00	47 × 47
IBM13	424	1,301	28,201	55.94 × 56.00	43 × 43
IBM14	614	1,529	57,333	59.85 × 60.00	49 × 44
IBM15	393	1,138	46,467	67.64 × 67.52	57 × 38
IBM16	458	1,315	67,118	81.03 × 81.12	45 × 48
IBM17	760	1,844	83,132	72.60 × 72.64	51 × 44
IBM18	285	1,029	53,763	64.82 × 64.96	55 × 39

No design has fixed hard macros. IBM05 is excluded because it has no macros; the benchmark set used by the challenge omits it. The macro and net counts summarized in the challenge README agree with these parsed values only for IBM01.

Weights. The score is

$$J(p) = W(p) + \tfrac{1}{2}\, D(p) + \tfrac{1}{2}\, C(p), \tag{15}$$

with the three components defined below (`compute_proxy_cost`). The density component already contains a factor $\frac{1}{2}$, so the raw density tail enters J with coefficient $\frac{1}{4}$.

Wirelength. Each pin or port with a non-empty list of sinks is the driver of one net n, whose pins are the driver and its sinks. The net weight ω_n is the driver's weight attribute (default 1; ports count with weight 1). With the half-perimeter wirelength

$$\mathrm{HPWL}_n = \big(\max_{q\in n} x_q - \min_{q\in n} x_q\big) + \big(\max_{q\in n} y_q - \min_{q\in n} y_q\big),$$

the component is

$$W(p) = \frac{\sum_n \omega_n\, \mathrm{HPWL}_n(p)}{(L_x + L_y)\, N_\omega}, \qquad N_\omega = \sum_n \omega_n \tag{16}$$

(`get_cost` and `get_wirelength`). The normalizer N_ω is the evaluator's weighted net counter `net_cnt`, accumulated while the netlist is parsed; it is not the number of entries in the net dictionary.

Density. Let R_i be the rectangle of object i and B_b cell b. Hard macros and soft clusters both contribute, through exact intersection areas:

$$\rho_b(p) = \frac{1}{\Delta_x \Delta_y} \sum_i |R_i \cap B_b| \tag{17}$$

(`get_grid_cells_density`). For $G \geq 10$, let $k = \lfloor 0.1\, G \rfloor$. The evaluator sorts the nonzero loads in decreasing order, sums the largest k of them (or all of them, if fewer than k are nonzero) and divides by k:

$$D(p) = \frac{1}{2} \cdot \frac{1}{k} \sum_{j=1}^{\min(k, n_{\text{occ}})} \rho_{(j)}, \tag{18}$$

where n_{occ} counts nonzero cells. Dropping zeros while keeping the denominator k is the same as padding with zero loads. For a grid with $G < 10$ cells the rule changes: D is half the mean over the occupied cells only (`get_density_cost`). Every IBM grid has between 810 and 2,255 cells, so the small-grid rule never applies to the challenge designs.

Congestion: net routing. Congestion uses a pattern-routing estimate on the same grid (`get_routing`). Each pin is mapped to the cell containing it, $(\lfloor y/\Delta_y \rfloor, \lfloor x/\Delta_x \rfloor)$; the challenge harness patches this lookup to clamp indices into the grid, so off-canvas pins land in border cells. For net n, let S_n be the set of distinct cells of its pins and s_n the driver's cell. Each cell along a route segment gains demand $\bar{\omega}_n$, equal to the driver weight when it exceeds 1 and to 1 otherwise:

- $|S_n| = 1$: no demand.
- $|S_n| = 2$: an L-shaped route. Horizontal demand runs along the driver's row from the smaller to the larger column, excluding the last; vertical demand runs along the sink's column from the smaller to the larger row, excluding the last (`__two_pin_net_routing`).
- $|S_n| = 3$: an L- or T-shaped Steiner route chosen from the relative order of the three cells (`__three_pin_net_routing`).
- $|S_n| > 3$: a star of two-pin routes from s_n to every other cell of S_n (`__split_net`).

The accumulated vertical and horizontal counts are divided by the per-cell capacities $c_V = \Delta_x r_V$ and $c_H = \Delta_y r_H$.

Congestion: smoothing and macro blockage. The normalized net demand is smoothed along one axis only (`__smooth_routing_cong`). Each vertical value is spread uniformly over the cells within $\pm s$ columns of the same row, and each horizontal value over the cells within $\pm s$ rows of the same column; at the grid boundary the window is clipped and the value is divided by the clipped window size, so smoothing conserves demand. Hard macros then add routing blockage (`__macro_route_over_grid_cell`). For each cell a hard macro covers, the vertical map gains (horizontal overlap length)$\times a_V$ and the horizontal map gains (vertical overlap length)$\times a_H$, both divided by the same capacities. A macro that fully covers a cell therefore adds $a_V/r_V = 0.667$ to its vertical value and $a_H/r_H = 0.459$ to its horizontal value. One edge correction applies: if a macro spans several rows and covers its lowest or highest row only partly, the vertical blockage of its entire top row is removed; if it spans several columns and covers its first or last column only partly, the horizontal blockage of its entire rightmost column is removed. The blockage is added after smoothing, so it is not smoothed:

$$V = \mathcal{S}_{\text{col}}(V_{\text{net}}) + V_{\text{mac}}, \qquad H = \mathcal{S}_{\text{row}}(H_{\text{net}}) + H_{\text{mac}}. \tag{19}$$

Congestion: tail statistic. The score treats the two maps as one list of $2G$ values, all vertical values followed by all horizontal ones, and takes the mean of its largest $\lfloor 0.05 \cdot 2G \rfloor$ entries, zeros included (`abu` with fraction 0.05, called by `get_congestion_cost`). If that count is zero, the maximum is used. The evaluator's separate per-direction functions (`get_V_congestion_cost`, `get_H_congestion_cost`) do not enter the score.

What the definition implies for search. All three components are piecewise: HPWL through pin extrema, density through rectangle–cell intersections, congestion through cell indices, routing patterns and the two tail selections. A move of one object changes the pins, nets and cells it touches, but both tails are recomputed over the whole grid, so the cost of scoring a move remains linear in G (Remark D.3). Congestion depends on cell indices rather than exact coordinates; a small move that keeps every pin in its cell leaves the net-routing demand unchanged, and the score jumps when a pin crosses a grid line, so the routing part is piecewise constant rather than merely kinked.

The benchmarks. The 17 designs are the ICCAD04 IBM mixed-size benchmarks [1] as converted by the TILOS MacroPlacement project [33]; Table 5 lists their statistics, parsed from the pinned inputs. The conversion groups each design's standard cells into *soft clusters* with the Circuit Training grouping flow [24], starting from a RePlAce initial placement [5]. A soft cluster is a fixed-size rectangle that may overlap other clusters and macros; it carries pins and so contributes to all three components. The score therefore depends on where the soft clusters sit, and Thinkorplace moves them together with the hard macros. They exist only inside the proxy: the organizers lock their sizes and state that the soft-macro abstraction does not carry over to place-and-route. No IBM design has fixed hard macros, and hard macros and soft clusters together occupy 80% of the canvas area in every design. The suite skips IBM05 because, as the TILOS ICCAD04 testcase notes state, that design has no macros.

C Algorithm details

This appendix specifies the competition-era programs at the level needed to reimplement them. Three programs matter (Figure 4):

- the *ranked single pipeline* of 20 May, inferred to be the entry the organizers scored: soft-rectangle initialization, Gaussian smooth descent, legalization and coordinate descent (CD) under a 720 s budget;
- the *three-lane system* committed on 21–22 May and, by the same inference, never scored: Lane A repeats the ranked pipeline with longer budgets, Lane B adds a curvature perturbation and Lane C an assignment step;
- the *v1 serial pipeline*, the lineage of the 13 May entry.

The modules that implement initialization, descent, legalization, incremental scoring and CD are identical between the inferred ranked commit `5485310` and the three-lane snapshot `2ab92a3` apart from import paths, so Sections C.1–C.4 describe both. Constants are the constructor defaults of the snapshot; Table 6 collects them.

C.1 Soft-rectangle initializer ("SDF")

The initializer (`SDFPlacer`, called through `sdf_init`) places hard macros only; in the competition code soft clusters keep their input coordinates. The name comes from the occupancy model, which uses distances to rectangle edges; the model is a product of logistic functions, not a signed distance field. Starting from the input coordinates clamped into the canvas, it runs 500 Adam steps (learning rate 3.0 μm, cosine-annealed to 3% of that) on

$$L = \frac{E(p)}{E(p^0)} + d_t\,\overline{r}_{10\%} + o_t\,\frac{O(p)}{L_x L_y}, \qquad r(u,v) = \sum_i \sigma\Big(\frac{w_i/2 - |u - x_i|}{\tau_t}\Big)\,\sigma\Big(\frac{h_i/2 - |v - y_i|}{\tau_t}\Big). \tag{20}$$

Here E is a clique wirelength $\sum_{(i,j)} c_{ij}\|p_i - p_j\|_1$ over pairs of hard macros that share a net, where a net touching m hard macros adds $1/(m-1)$ to each of its pairs; p^0 is the starting placement. The density term $\overline{r}_{10\%}$ is the mean of the largest 10% of the occupancy values r at a 32×32 lattice of points spanning the canvas, and σ is the logistic function. O is the total pairwise overlap area of hard macros. With progress $t \in [0,1]$, the temperature τ_t decays geometrically from 0.05 to 0.002 of the canvas diagonal, $d_t = 0.4(1.5 - 0.8t)$ and $o_t = 80(0.05 + 0.95\sqrt{t})$. Every 20 steps the state with the

Table 6: Competition-era defaults of the ranked single pipeline (most likely commit `5485310`; inferred) and the three-lane system (snapshot `2ab92a3`). An em dash marks a stage the program does not have.

Parameter	Ranked pipeline	Three-lane system	Notes
Global budget	720 s	2,700 s	Three-lane: join timeout, unbounded after expiry
Descent steps	500	500	Third retry uses 300
Learning rate	$0.005\,L_x$	$0.005\,L_x$	Adam, no schedule
Wirelength smoothing	$5\times10^{-3}L_x \to 5\times10^{-5}L_x$	same	Linear in step index
Overlap weight	$0 \to 10$; retries 50, 100	same	Ramp over first 70% of steps
Boundary weight μ	50	50	After canvas-area normalization
Gaussian width	max(half size, half cell)	same	Per axis
Legalizer	2% steps, 80 rings	same	Then push-apart, at most 50 sweeps
CD budget	cap 600 s, minimum 300 s	CD1: 900/450 s; CD2: 360/180 s	Plateau: five sweeps below 0.001
CD candidates	at most 10 per axis	same	Acceptance 10^{-9}
Curvature stage	—	120 s, two iterations	$t \in \{0.3, 1, 3\}\,\mu$m; polish 30–180 s
Eigensolver	—	$k = 1$, SA, tol. 10^{-2}, 50 iterations	Iteration cap, not a time cap
Assignment	—	300 s, $K = 50$ of top 150, 100 slots	Patience 15; acceptance 10^{-7}
Seeds	42	lanes 42, 142, 242	See Table 7

lowest $E/E(p^0) + 0.1\,O/(L_xL_y)$ is retained. A greedy pass then places macros in decreasing order of area: a macro that clears every placed macro by $0.05\,\mu$m stays; otherwise the pass searches square rings of step $0.2\max(w_i, h_i)$ around its position (up to 249 rings) and takes the nearest clearing point of the first ring that has one. The macro-pair list came from splitting pin names at the first "/"; Appendix F records what that did on hierarchical designs.

C.2 Smooth proposal surrogate

The descent optimizes the centers of all movable objects, hard and soft, with fixed objects' gradients set to zero. Every term is evaluated on positions clamped into the canvas, so a coordinate outside the canvas feels only the boundary penalty. The loss at step t of N is

$$\widetilde{J}_t = \widetilde{W}_{\gamma_t} + \tfrac{1}{2}\widetilde{D} + \tfrac{1}{2}\widetilde{C} + \lambda_t P_{\text{ov}} + \mu P_{\text{out}}. \tag{21}$$

- *Wirelength.* Each net's pin extrema are replaced by log-sum-exp terms [25], $\gamma \log \sum_q e^{x_q/\gamma}$ for the maximum and the mirrored form for the minimum, weighted by the net weight and normalized by $(L_x + L_y)N_\omega$ exactly as in Appendix B. The smoothing length is $\gamma_t = g_t L_x$, with g_t linear in the step index from 5×10^{-3} to 5×10^{-5}.
- *Density.* Object i becomes a separable Gaussian with $\sigma_{x,i} = \max(w_i/2, \Delta_x/2)$ and $\sigma_{y,i} = \max(h_i/2, \Delta_y/2)$; its mass in a cell is $w_i h_i$ times the product of the two normal-CDF differences over the cell's extent. Cell density is total mass over cell area, and $\widetilde{D} = \frac{1}{2}$ of the mean of the largest $\max(1, \lfloor 0.1G \rfloor)$ cells. Because the evaluator's density component also carries the factor $\frac{1}{2}$, the raw Gaussian tail enters $\widetilde{J}$ with coefficient $\frac{1}{4}$, matching Equation (15). Mass falling outside the canvas is not renormalized.
- *Congestion.* A differentiable trace of the evaluator's routing (Appendix B). Every net is a star from its driver to each sink; each pair contributes horizontal demand on the driver's row and vertical demand on the sink's column. A pin's row and column are soft assignments, a softmax over cells of $-((u - c_k)/(0.5\Delta))^2$ with cell centers c_k; the endpoints of each segment are their expected indices, combined by a log-sum-exp minimum and maximum with sharpness 6 per cell; and cell k lies on the

segment with weight $\sigma(4(k-k_{\min}+\frac{1}{2}))-\sigma(4(k-k_{\max}+\frac{1}{2}))$. Demand is divided by the cell capacities, smoothed over $\pm s$ cells along one axis as in the evaluator, and added to a macro-blockage map built from clamped rectangle–cell overlaps. The score is the mean of the top 5% of the concatenated vertical and horizontal maps. Three simplifications remain: three-cell nets are star-routed rather than given L- or T-shapes; stars run to every sink pin rather than to distinct cells, with the raw net weight; and the evaluator's partial-overlap correction at macro edges is omitted.

- *Penalties.* P_{ov} is the total pairwise penetration area $\sum_{i<j}(\bar{w}_{ij}-|x_i-x_j|)_+(\bar{h}_{ij}-|y_i-y_j|)_+$ over hard macros, with required separations $\bar{w}_{ij}=(w_i+w_j)/2$ and $\bar{h}_{ij}=(h_i+h_j)/2$, divided by the canvas area L_xL_y. Its weight λ_t rises linearly from 0 to λ_{end} over the first 70% of the steps, then stays constant. P_{out} is the sum of squared distances of centers outside their legal box, also divided by L_xL_y, with $\mu=50$.

The optimizer is Adam [13] with learning rate $0.005\,L_x$, default moment parameters and no schedule, for $N=500$ steps in single precision on CUDA, Apple MPS or CPU, in that order of preference. The trajectory therefore depends on the device, and GPU reductions need not be bitwise deterministic. A retry ladder guards feasibility. Each attempt runs the descent and the legalizer of Section C.3; if hard overlaps remain, it applies the push-apart pass; if they still remain, the next attempt starts. The attempts use $\lambda_{\text{end}}=10$, 50 and 100, the last with 300 steps. If all three fail, the lane falls back to the soft-rectangle placement plus push-apart. The ranked pipeline stops retrying once fewer than 60 s of its budget remain.

C.3 Legalization

The legalizer (`greedy_macro_legalize`) treats hard macros only. Fixed macros are placed first. Movable macros follow in decreasing order of area; each tries its descent position, then offsets on square rings of step 2% of the canvas width and height, up to 80 rings, clamped into the canvas. The first candidate whose penetration with every placed macro is at most $10^{-4}\,\mu$m on some axis is accepted. A macro with no such candidate keeps its position. If the canonical overlap check, the challenge's `compute_overlap_metrics`, which counts a pair as overlapping when its penetration is positive on both axes with no tolerance, still finds overlaps, the push-apart pass (`project_overlaps`) runs up to 50 sweeps: for each overlapping pair it removes the penetration along the axis where it is smaller, splitting the move equally when both macros are movable, and clamps centers to the canvas.

C.4 Coordinate descent

CD (`run_cd_adaptive`) visits every movable object, hard or soft, in an order reshuffled at each sweep, and optimizes x and then y.

- *Legal interval.* For a soft cluster it is the canvas interval. For a hard macro, the blockers are the hard macros whose perpendicular separation is less than the required separation plus $10^{-4}\,\mu$m; the nearest blocker on each side bounds the interval with a clearance of $10^{-4}\,\mu$m. If the bounds cross, the coordinate is pinned. This is a subinterval of the component of the slice in Remark D.1 that contains the current value (blockers taken with a $10^{-4}\,\mu$m tolerance, ends pulled in by the clearance).
- *Candidates.* The interval ends and midpoint; the current value; the 25%, 50% and 75% quantiles of pin targets, where each incident net with pins outside the object supplies the low end, high end and midpoint of those pins' extent, shifted by the object's pin offset; and the three cell-aligned positions (an edge of the object on a grid line) nearest to the current value. Values are clipped to the interval and deduplicated at 10^{-6}.
- *Scoring and acceptance.* All candidates for one axis are scored in a single batched call to the incremental evaluator (`delta_cost_axis_batch`). The best is applied if it lowers the incremental score by more than 10^{-9} and moves the coordinate by more than $10^{-7}\,\mu$m.

- *Unreachable branch.* A golden-section line search runs only when the candidate list exceeds 12 entries. The generator produces at most ten values and truncates its output to 12, so the branch never executes; it is noted here only because it appears in the source.
- *Stopping.* After each sweep the decrease of the incremental score is recorded. CD stops at its hard cap, or once it has run for half the cap and the last five sweeps each lowered the score by less than 0.001. The cap is checked between objects, so a sweep can be cut short.

The incremental evaluator caches pin positions, per-net extrema, density loads, raw net-routing demand per direction and macro blockage. A move updates the structures the object touches, then recomputes smoothing and both tails over the grid (Remark D.3); a one-step revert restores the previous state.

C.5 Lane B: curvature perturbation

Lane B runs CD1 (the Lane A phase), then a bounded curvature stage if CD1 is free of overlaps, then CD2 with cap $\max(60, 0.4 \times 900) = 360\,\text{s}$. The stage endpoint replaces the CD1 placement only if its canonical score is no higher.

Operator. The curvature stage uses the legacy surrogate J_B of the v1 lineage, not the Gaussian surrogate. J_B is the sum of log-sum-exp HPWL with fixed $\gamma = 5 \times 10^{-4} L_x$, normalized by $(L_x + L_y)$ times the unweighted number of nets; $\frac{1}{2}$ of half the top-10% mean of exact rectangle–cell density; and $\frac{1}{2}$ of a bounding-box RUDY estimate [31], which spreads each net's weight uniformly over its smoothed bounding box, sums horizontal and vertical demand per cell and averages the top 5% of the G cells, without macro blockage or smoothing. Positions are clamped into the canvas and arithmetic is float32. The Hessian-vector product [28] is `torch.autograd.functional.hvp` of J_B restricted to movable coordinates.

Eigensolver. SciPy's interface to ARPACK [18] (`eigsh`) seeks one eigenvector with `which="SA"` (smallest algebraic), tolerance 10^{-2}, at most 50 iterations and 22 Lanczos vectors, without a user starting vector. If it returns nothing, a shifted power iteration on $\alpha I - H$ takes over, with $\alpha = 2\|Hv\| + 1$ estimated over five products and 50 further iterations from a NumPy random start. The stage stops when the smallest eigenvalue is at least -10^{-3}.

Steps. The eigenvector v is normalized over all coordinates. Six candidates $p + stv$, with $s \in \{+1, -1\}$ and step length $t \in \{0.3, 1, 3\}\,\mu\text{m}$, are formed from the iteration's starting placement, so t is the Euclidean length of the whole displacement, not a per-macro step: spread evenly over IBM01's 2,280 movable coordinates, $t = 0.3\,\mu\text{m}$ would move each by about $0.006\,\mu\text{m}$. Each candidate is clipped to the canvas, has fixed objects restored, receives the push-apart pass, is discarded if overlaps remain, and is otherwise polished by CD (at least 30 s, cap 180 s, patience 3, threshold 0.001). A polished candidate is accepted when it is free of overlaps and its canonical score is below the best so far by more than 10^{-7}; the stage state moves to it at once.

Cascade. The stage repeats for at most two iterations within 120 s. A new iteration starts only if at least $\max(60, 1.2\bar{t})$ seconds remain, where $\bar{t}$ is an exponentially weighted average of iteration time; no new candidate starts in the last 30 s; and the stage stops when an iteration improves the best score by less than 10^{-5}. One candidate polish may last 180 s, longer than the whole stage budget, so the stage is not bounded in wall-clock time.

C.6 Lane C: assignment

Lane C runs CD1, then repeated assignment steps on the incremental evaluator for at most 300 s or until 15 consecutive steps are rejected, then CD2. One step is:

1. *Cluster.* Score every movable hard macro by $\sum_{n \ni i} 1/\max(1, |n| - 1)$ over its nets, where $|n|$ counts pins, and draw $K = 50$ macros without replacement from the 150 highest-scoring ones, with NumPy generator seed $42 + k$ at step k.

2. *Slots.* Take the 50 macros' current centers, then grid-cell centers inside the cluster's bounding box padded by max(cell size, 10% of its span), skipping cells too close to the canvas edge for the cluster's median macro, until 100 slots exist; the whole canvas is used if the box is too small.
3. *Cost matrix.* Entry (i, j) is the incremental score change from moving macro i alone to slot j against the current background, computed by move, score and revert; it is 0 if the macro already sits there, and $+\infty$ if the slot overlaps a hard macro outside the cluster.
4. *Solve.* Rows with no finite entry are set to zero; every other infinite entry becomes 10^6 times the larger of 1 and the largest finite entry; and SciPy's `linear_sum_assignment` solves the rectangular 50×100 problem with the Hungarian method's guarantees [15].
5. *Sequential commit.* Moves whose original entry is finite are applied in order of predicted gain, each after a legality check against the current placement, including cluster macros already moved. An illegal move is skipped, so an assigned swap into a slot still occupied is dropped.
6. *Batch acceptance.* The batch stays if at least one macro moved, the canonical overlap check finds no overlap, and the incremental score fell by more than 10^{-7}; otherwise every move is reverted.

After the loop, the best state is rescored canonically. If it is no more than 0.5% worse than the CD1 score it initializes CD2 (cap 360 s); otherwise CD2 starts from CD1. The lane returns the overlap-free candidate with the lowest canonical score among CD1, the assignment endpoint and CD2. The competition worker logged only accept or reject per step, not the rejection reason.

C.7 Supervisor and the ranked pipeline

The three-lane supervisor spawns one process per lane. Each process sets OpenMP, MKL and OpenBLAS thread counts to one third of the host's logical CPUs, unless the environment already sets them. The supervisor joins the lanes in order with timeout equal to the time left before its 2,700 s deadline minus 15 s; a non-positive remainder becomes `timeout=None`, an unbounded wait (Appendix F). A lane still alive after its join is terminated. Completed lanes write one result file each at normal completion. The supervisor sorts successful lanes by their recorded canonical score, rechecks hard overlaps and returns the best overlap-free placement; if every lane fails it returns the soft-rectangle placement after push-apart.

The ranked pipeline has no subprocesses. It runs the descent ladder, then CD with the cap

$$T_{\mathrm{CD}} = \max(30,\ \min(T_{\mathrm{rem}} - 10,\ 600))\ \mathrm{s},$$

where T_{rem} is the time left in its 720 s budget, and it raises an error if the final placement has overlaps. Its listed runtime on the leaderboard, about 12 minutes per design, matches that budget.

C.8 The v1 serial pipeline

The v1 serial pipeline is the family that preceded the smooth descent. The 13 May entry was one variant of it, the adaptive wall-safe cascade; the snapshot holds the final stacked-periphery configuration, whose 17 May run is the v1 record (column "v1 (17 May)" of Table 16; mean 1.05750). With the default budget of 3,300 s, 210 s are reserved for the periphery stage and the remaining $B = 3{,}090$ s are split as follows.

1. *E25 branch.* Soft-rectangle initialization and push-apart; CD over all movable objects (cap $0.2B = 618$ s, minimum 300 s, patience 3, threshold 0.001); grid-cell large-neighborhood search (LNS, $0.06B$), which removes the $\max(1, \min(30, \lfloor 0.05\, n_{\mathrm{hard}} \rfloor))$ hard macros whose trial move to the canvas center scores best and reinserts each at the best legal grid-cell center; and simulated annealing [14] ($0.06B$) on hard macros with random breakpoint moves, Metropolis acceptance, temperature decaying exponentially from 5×10^{-4} to 10^{-6} over the budget, and best-state tracking.
2. *E41 branch*, skipped if fewer than 300 s remain. A differentiable proxy optimization (DPO) initializer runs three Adam phases on log-sum-exp wirelength, density and RUDY with growing overlap weights from the soft-rectangle start, and keeps the lower-scoring of the two. CD, LNS and annealing

follow ($0.05B$ each for the last two), then joint reinsertion of three macros at a time: triples of high-adjacency macros, five single-move-ranked grid-cell candidates each, and all of the up to 125 combinations checked for legality and scored ($0.05B$).
3. *Plateau.* The branch with the lower canonical score is kept (ties go to E25).
4. *Curvature cascade* (target $0.2B$). The Lane B procedure with an unbounded eigensolver (tolerance 10^{-4}, up to 500 iterations), up to five iterations, and the same six steps per iteration.
5. *Portfolio* (the remaining time, skipped below 90 s). The same procedure with the curvature operator built from three non-canonical component weightings, $(1, 0, 1)$, $(1, 1, 0)$ and $(0, 1, 1)$ in the order wirelength, density, congestion; step lengths drawn from a half-Cauchy distribution; CD polishes of 20–60 s. Polishing and acceptance always use the canonical score.
6. *Periphery.* Each movable object moves 1% of its gap toward its nearest canvas edge, along that axis only; push-apart and CD (cap 180 s) follow; the result is kept only if it is free of overlaps and strictly better.

The lowest-scoring overlap-free candidate among the branch, cascade and portfolio results enters the periphery stage. Initialization, canonical scoring and the eigensolves are not charged to any stage budget, and per-candidate polish caps are not clipped to the stage budget, so the 3,300 s budget is advisory.

C.9 Seeds

Table 7: Sources of randomness in the competition snapshot and the effect of the nominal lane seed. "Global" means the process-wide Torch, Python and NumPy generators.

Component	Seed actually used	Lane seed effect
Soft-rectangle initializer	42; also resets the global generators to 42	None
Smooth descent	Lane seed +100×attempt, consumed only by non-default initializers	None
Legalizer, push-apart	Deterministic	None
CD sweep order	NumPy generator seeded with the sweep index	None
Lane B eigensolver	ARPACK's own start vector; fallback uses the global NumPy generator	None
Lane C cluster draw	NumPy generator seeded with $42 + k$ at step k	None
Ranked pipeline	Seed 42; same components as Lane A	—
v1	Initializer 42; DPO 42; annealing, joint reinsertion and portfolio steps 42; LNS deterministic	—

Table 7 traces every random choice in the snapshot. The lane seeds 42, 142 and 242 reach the descent constructor, which uses a seed only for the DPO or random initializers; the default soft-rectangle path ignores it and fixes its own seed at 42. Adam, the legalizer and push-apart are deterministic given their input, and the CD sweep order depends only on the sweep index. The three lanes therefore share one initialization and descent, and their CD1 phases follow the same sweep sequence, differing only where wall-clock stopping cuts them at different points. The engineering runs confirm this: wherever initializer and legalized-descent checkpoints survive for all three lanes, their coordinates are bit-identical to each other and to standalone Lane A (Appendix F). Lane C's sampling is seeded from a separate base of 42, and Lane B's eigensolver start is not set by any seed. Repeated runs of the default configuration are repeatability trials, not independent starts. An independent-start protocol would vary an initializer that consumes the seed and seed the eigensolver start vector and the Lane C base explicitly.

D Proofs

This appendix proves the results stated in the main text and adds five remarks, each with its proof, that the main text uses informally. Coordinates are real numbers, sizes and orientations are fixed, and touching rectangles are legal. Hard macro i has center (x_i, y_i), width w_i and height h_i on a $L_x \times L_y$ canvas; for a pair $i \neq j$ write $\bar{w}_{ij} = (w_i + w_j)/2$ and $\bar{h}_{ij} = (h_i + h_j)/2$.

D.1 Feasible set and coordinate slices

Proof of Proposition 2.1. Let $\mathcal{B}$ be the set of placements that satisfy every canvas-containment inequality $w_i/2 \le x_i \le L_x - w_i/2$, $h_i/2 \le y_i \le L_y - h_i/2$ and every fixed-coordinate equality. For an unordered pair $e = \{i, j\}$ of hard macros, the pair does not overlap exactly when one of four closed halfspaces contains the placement:

$$H_{e,1} = \{x_j - x_i \ge \bar{w}_{ij}\}, \quad H_{e,2} = \{x_i - x_j \ge \bar{w}_{ij}\}, \quad H_{e,3} = \{y_j - y_i \ge \bar{h}_{ij}\}, \quad H_{e,4} = \{y_i - y_j \ge \bar{h}_{ij}\}.$$

With $\mathcal{E}$ the set of pairs and $m = |\mathcal{E}|$, the feasible set is $\mathcal{F} = \mathcal{B} \cap \bigcap_{e \in \mathcal{E}} \bigcup_{b=1}^{4} H_{e,b}$. A finite intersection of finite unions equals the union, over all ways of choosing one member from each union, of the intersections of the chosen members. Choosing branch $\sigma(e)$ for every pair gives

$$\mathcal{F} = \bigcup_{\sigma \in \{1,2,3,4\}^{\mathcal{E}}} P_\sigma, \qquad P_\sigma = \mathcal{B} \cap \bigcap_{e \in \mathcal{E}} H_{e,\sigma(e)}.$$

Each P_σ is the intersection of finitely many closed halfspaces and hyperplanes, hence a closed convex polyhedron. The containment inequalities bound every coordinate, so P_σ is bounded and therefore compact. There are 4^m choices of σ; different choices may give overlapping, identical or empty polyhedra, so 4^m is an upper bound on the number of distinct pieces. □

Remark D.1 (Legal coordinate slices). Fix a legal placement and vary only x_i for a movable hard macro i. Let

$$S_i = \{j \neq i \text{ hard} : \ |y_i - y_j| < \bar{h}_{ij}\}$$

be the macros that overlap i's horizontal band. The legal values of x_i form

$$I_i = \left[\tfrac{w_i}{2},\, L_x - \tfrac{w_i}{2}\right] \setminus \bigcup_{j \in S_i} (x_j - \bar{w}_{ij},\, x_j + \bar{w}_{ij}), \tag{22}$$

a finite union of closed intervals, possibly including single points.

Proof. Only the pairs that contain i change. For $j \notin S_i$, the placement already satisfies $H_{e,3}$ or $H_{e,4}$, which does not involve x_i, so j imposes no constraint. For $j \in S_i$ both vertical branches fail, so the pair is legal exactly when $|x_i - x_j| \ge \bar{w}_{ij}$, which removes the open interval in Equation (22). Removing finitely many open intervals from a closed interval leaves finitely many closed intervals; endpoints are legal because contact is allowed. The y_i slice follows by exchanging axes. A soft cluster has no overlap constraint, so its slice is the canvas interval, and a fixed object's slice is a single point. □

The CD search of Appendix C.4 does not enumerate I_i. It keeps the component that contains the current value, bounded by the nearest blocker on each side and shrunk by a clearance of $10^{-4}\,\mu$m; a move across a blocker is left to the proposal stages. Successive moves of other macros change S_i, so CD sweeps can pass between polyhedra P_σ even though each single move stays in one interval.

D.2 Incremental scoring

Let $\Psi(p)$ be the cache obtained by computing everything from scratch at placement p: pin positions, per-net extrema and, for each net n and object i, the weighted HPWL term $a_n(p)$, the density-load vector $d_i(p)$ of object i, the raw routing vectors $r_n^o(p)$ of net n and the macro-blockage vectors $b_i^o(p)$ for orientation $o \in \{H, V\}$, with $b_i^o = 0$ for soft clusters. The cache stores the aggregates

$$T = \sum_n a_n(p), \qquad \Lambda = \sum_i d_i(p), \qquad R^o = \sum_n r_n^o(p), \qquad M^o = \sum_i b_i^o(p), \tag{23}$$

and every score is a fixed function of these aggregates: normalization, smoothing and the two tails (Appendix B).

Remark D.2 (Cache commutation). Let U_i move object i and $\widehat{U}_i$ be the cache update that recomputes i's pins, the extrema of the nets incident to i, and then subtracts the old and adds the new a_n, r_n^o for those nets and d_i, b_i^o for i. In exact arithmetic, $\widehat{U}_i\Psi(p) = \Psi(U_i p)$, and the same holds for any sequence of moves and for reverting them.

Proof. A net term a_n or r_n^o depends only on the positions of the pins of n; it changes under U_i only if n is incident to i. The terms d_j and b_j^o depend only on object j's rectangle and change only for $j = i$. Hence, for instance, $T(U_i p) = T(p) - \sum_{n \ni i} a_n(p) + \sum_{n \ni i} a_n(U_i p)$, which is exactly the update; the other three aggregates follow in the same way, provided the update recomputes each incident net's extrema from all its pins when the moved pin was extreme. Equal aggregates give equal scores. Induction on the number of moves gives the sequence statement, and a revert applies the same substitution with the roles of old and new exchanged. □

The remark is conditional on complete dependency tracking and exact arithmetic. In floating point, subtracting and adding terms need not cancel, and the summation order differs from a fresh computation; the measured discrepancies are reported in the main text.

Proof of Proposition 3.1. Let $\widehat{J}$ be the incremental score and J the canonical score, with $|\widehat{J}(S_p) - J(p)| \le \epsilon$ and $|\widehat{J}(S_q) - J(q)| \le \epsilon$ at the two endpoints, and suppose $\widehat{J}(S_q) \le \widehat{J}(S_p) - \tau$. Adding and subtracting the incremental values,

$$\begin{aligned} J(q) - J(p) &= [J(q) - \widehat{J}(S_q)] + [\widehat{J}(S_q) - \widehat{J}(S_p)] + [\widehat{J}(S_p) - J(p)] \\ &\le \epsilon - \tau + \epsilon = -\tau + 2\epsilon. \end{aligned}$$

If $\tau > 2\epsilon$ the right side is negative, so the canonical score strictly decreases. □

If the three components carry absolute errors at most ϵ_W, ϵ_D and ϵ_C, the triangle inequality and the weights $1, \frac{1}{2}, \frac{1}{2}$ give $\epsilon = \epsilon_W + \frac{1}{2}\epsilon_D + \frac{1}{2}\epsilon_C$. The proposition needs the bound at the two states compared; the sampled errors in the main text satisfy it with a wide margin on the sampled states, which is the sense in which the CD threshold of 10^{-9} and the assignment threshold of 10^{-7} are safe.

D.3 Tail statistics and the cost of a scored move

For $z \in \mathbb{R}^N$ and an integer $1 \le k \le N$, write $z_{(1)} \ge \cdots \ge z_{(N)}$ for the sorted entries and $A_k(z) = k^{-1} \sum_{j=1}^k z_{(j)}$ for the top-k mean; $(a)_+ = \max\{a, 0\}$ and $\mathcal{Q}_k = \{\beta \in \mathbb{R}^N : 0 \le \beta_j \le 1/k,\ \sum_j \beta_j = 1\}$.

Proof of Lemma 2.2. Threshold form. For any $\theta \in \mathbb{R}$,

$$\sum_{j=1}^k z_{(j)} = k\theta + \sum_{j=1}^k (z_{(j)} - \theta) \le k\theta + \sum_{j=1}^k (z_{(j)} - \theta)_+ \le k\theta + \sum_{j=1}^N (z_j - \theta)_+.$$

At $\theta = z_{(k)}$ both inequalities are equalities: the first k differences are nonnegative, and every entry outside the top k is at most $z_{(k)}$, so its positive part vanishes. Dividing by k gives $A_k(z) = \min_\theta\{\theta + k^{-1}\sum_j (z_j - \theta)_+\}$, attained at $\theta = z_{(k)}$.

Maximum form. For $\beta \in \mathcal{Q}_k$ and $\theta = z_{(k)}$,

$$\beta^\top z = \theta + \sum_j \beta_j (z_j - \theta) \le \theta + \sum_j \beta_j (z_j - \theta)_+ \le \theta + \frac{1}{k}\sum_j (z_j - \theta)_+ = A_k(z),$$

using $\sum_j \beta_j = 1$ and $\beta_j \le 1/k$. The vector with entries $1/k$ on the indices of the k largest entries lies in $\mathcal{Q}_k$ and attains $A_k(z)$, so $A_k(z) = \max_{\beta \in \mathcal{Q}_k} \beta^\top z$.

Convexity. The maximum is attained at one of the finitely many vectors that put $1/k$ on a k-subset, so A_k is a maximum of finitely many linear functions: convex and piecewise affine, and differentiable wherever the k-th and $(k+1)$-th largest entries differ.

Stability. For any $\beta \in \mathcal{Q}_k$, $\beta^\top(z + \delta) \le A_k(z) + \max_{\beta' \in \mathcal{Q}_k} \beta'^\top \delta$; maximizing over β and exchanging z and $z + \delta$ gives $|A_k(z + \delta) - A_k(z)| \le \max_{\beta \in \mathcal{Q}_k} |\beta^\top \delta|$. Every $\beta \in \mathcal{Q}_k$ has $\|\beta\|_1 = 1$, $\|\beta\|_\infty \le 1/k$ and $\|\beta\|_2^2 \le \|\beta\|_\infty \|\beta\|_1 \le 1/k$. Hölder's inequality in the three dual pairs $(1, \infty)$, $(2, 2)$ and $(\infty, 1)$ yields the bounds $\|\delta\|_\infty$, $\|\delta\|_2/\sqrt{k}$ and $\|\delta\|_1/k$. □

Both canonical tails are of this form. For $G \ge 10$ the density component is $\frac{1}{2}A_k(\rho)$ with $N = G$ and $k = \lfloor 0.1G \rfloor$: the loads are nonnegative, so summing the largest k nonzero loads and dividing by k equals A_k of the full vector with its zeros. The congestion component is $A_{k'}$ of the $N = 2G$ concatenated values with $k' = \lfloor 0.05 \cdot 2G \rfloor$, and A_1 (the maximum) when $k' = 0$. Convexity in the loads does not make the score convex in coordinates, because the loads are themselves piecewise functions of the coordinates. For the solver the $\|\delta\|_1/k$ bound is the useful one: a move changes the loads in few cells, so its effect on a tail is at most the total load it shifts divided by k.

Remark D.3 (Cost of a scored move). Let $T_{\text{mut}}(i)$ be the work to update object i's pins, its incident nets and its own grid contributions, and $T_{\text{sel}}(n, k)$ the cost of selecting the k largest of n values, with $T_{\text{sel}}(0, 0) = 0$. For $G \ge 10$, a call that moves object i and returns its score costs

$$T_{\text{move}}(i) = T_{\text{mut}}(i) + \Theta(G) + T_{\text{sel}}\big(n_{\text{occ}}, \min\{k, n_{\text{occ}}\}\big) + T_{\text{sel}}(2G, k'),$$

where $n_{\text{occ}} \le G$ counts nonzero density cells. The work is therefore at least linear in G, however few cells the move touches.

Proof. The mutation is local by Remark D.2. Producing the score then requires, from the aggregates of Equation (23): dividing every density load by the cell area and every routing value by its capacity; smoothing both net-demand maps, which touches each cell a bounded number of times for the fixed range $s = 2$; adding the blockage maps; and the two selections. Each of the first three steps visits all G cells, giving the $\Theta(G)$ term. The implementation keeps no incremental order-statistic structure, so the selections are recomputed in full. □

Full recomputation, by contrast, re-derives every pin position, re-routes every net and re-rasterizes every object. The microbenchmarks in the main text measure the saving at the IBM grid sizes ($810 \le G \le 2{,}255$); they do not establish a cost that depends on the moved object's degree alone.

D.4 Negative curvature

Proof of Proposition 3.2. Recall the setting: f is twice continuously differentiable on an open set U containing p, v is a unit vector with $v^\top \nabla^2 f(p) v \le -\eta < 0$, the sign $s \in \{-1, 1\}$ satisfies $s\nabla f(p)^\top v \le 0$, and the segment $\{p + usv : 0 \le u \le t\}$ lies in U, with the Hessian L-Lipschitz on it in the operator norm. Put $\phi(u) = f(p + usv)$ for $u \in [0, t]$. Then $\phi'(0) = s\nabla f(p)^\top v \le 0$ and $\phi''(u) = v^\top \nabla^2 f(p + usv) v$,

because $s^2 = 1$. The Lipschitz condition and $\|sv\|_2 = 1$ give $|\phi''(u) - \phi''(0)| \le Lu$, so $\phi''(u) \le -\eta + Lu$. Taylor's formula with integral remainder gives

$$\phi(t) - \phi(0) = t\,\phi'(0) + \int_0^t (t-u)\,\phi''(u)\,du \le \int_0^t (t-u)(-\eta + Lu)\,du = -\frac{\eta}{2}t^2 + \frac{L}{6}t^3.$$

For $L > 0$ and $0 < t \le 3\eta/(2L)$, $\frac{L}{6}t^3 \le \frac{\eta}{4}t^2$, so the decrease is at least $\eta t^2/4$. For $L = 0$ the bound is $-\eta t^2/2$ on the whole segment. □

The bound concerns the unconstrained step. If the favorable sign points out of the feasible set, it says nothing: $f(x) = x - x^2$ on $[0, \frac{1}{4}]$ has $f'(0) = 1$ and $f''(0) = -2$, yet $x = 0$ is its strict constrained minimum because the descent sign leaves the interval. Lane B also clips, legalizes and polishes each step before scoring it with the canonical proxy, so the bound applies to the proposal, not to the accepted endpoint, and Appendix E shows that its smoothness hypothesis fails for the operator Lane B differentiates.

Remark D.4 (Eigen-residual bound). Let $\mathbf{H}$ be a real symmetric matrix, v a unit vector, $\hat{\lambda} = v^\top \mathbf{H} v$ and $r = \mathbf{H}v - \hat{\lambda}v$. Then $\operatorname{dist}(\hat{\lambda}, \operatorname{spec}\mathbf{H}) \le \|r\|_2$, and $\hat{\lambda} < 0$ implies $\lambda_{\min}(\mathbf{H}) < 0$.

Proof. Expand $v = \sum_j a_j u_j$ in an orthonormal eigenbasis, $\mathbf{H}u_j = \lambda_j u_j$, with $\sum_j a_j^2 = 1$. Then $r = \sum_j a_j(\lambda_j - \hat{\lambda})u_j$ and $\|r\|_2^2 = \sum_j a_j^2(\lambda_j - \hat{\lambda})^2 \ge \min_j(\lambda_j - \hat{\lambda})^2$. The second claim is the Rayleigh bound $\lambda_{\min}(\mathbf{H}) \le v^\top \mathbf{H} v$. □

For the IBM01 probe of Appendix E, the automatic-differentiation operator is the Hessian of the branch of J_B active at the endpoint, so it is symmetric in exact arithmetic, as the remark requires; in float32 it is symmetric only up to rounding. The residual 3.49×10^{-7} then places an eigenvalue of that operator within 3.49×10^{-7} of -0.442458, and its smallest eigenvalue is at most -0.442458. The remark says nothing about whether the operator is the Hessian of the scored function, and a small residual can locate an interior eigenvalue rather than the smallest one.

D.5 Assignment interactions

Let p^0 be the current placement, $p^{i \leftarrow j}$ the placement with only macro i moved to slot j, and p^z the placement after all moves of an assignment z. For an exact objective F, write $\Delta_{ij} = F(p^{i \leftarrow j}) - F(p^0)$ and define the interaction residual $R_F(z) = F(p^z) - F(p^0) - \sum_{i,j} \Delta_{ij} z_{ij}$, so the joint move improves F exactly when $\sum_{ij} \Delta_{ij} z_{ij} + R_F(z) < 0$. For the proxy, the weights give $R_J = R_W + \frac{1}{2}R_D + \frac{1}{2}R_C$. A net that contains pins of at most one moved macro contributes nothing to R_W, because its HPWL changes identically under the joint move and under that macro's single move; shared nets can contribute with either sign. Independent-set matching in detailed placement restricts each matching to cells that share no nets [4], which removes this HPWL residual; Lane C imposes no such restriction and draws its macros by individual connectivity, so cluster members can share nets. Density and congestion interact even across disjoint nets, through the shared tail selections.

Remark D.5 (An optimal assignment that worsens HPWL). Two movable macros sit at $x_1 = 2$ and $x_2 = 12$ on a common horizontal line. Each has one zero-offset pin on a shared net of weight 2 and a second, coincident pin on a unit-weight net to a fixed point terminal at its own starting position. Up to the positive normalization, the HPWL is

$$F(x_1, x_2) = 2|x_1 - x_2| + |x_1 - 2| + |x_2 - 12|.$$

With candidate slots 3 and 11, the optimal assignment predicts a change of -18 but realizes $+14$.

Proof. All pins share one vertical coordinate, so only horizontal spans count. Initially $F(2,12) = 2 \cdot 10 + 0 + 0 = 20$. The four single moves, each with the other macro in place, give

$$F(3,12) = 2 \cdot 9 + 1 + 0 = 19, \qquad F(11,12) = 2 \cdot 1 + 9 + 0 = 11,$$
$$F(2,3) = 2 \cdot 1 + 0 + 9 = 11, \qquad F(2,11) = 2 \cdot 9 + 0 + 1 = 19,$$

so, with rows for macros 1 and 2 and columns for slots 3 and 11,

$$\Delta = \begin{pmatrix} -1 & -9 \\ -9 & -1 \end{pmatrix}.$$

The two complete assignments have predicted totals $-1 - 1 = -2$ (each macro to its nearer slot) and $-9 - 9 = -18$ (the swap), so the optimum is the swap, $(x_1, x_2) = (11, 3)$. Its true value is $F(11,3) = 2 \cdot 8 + 9 + 9 = 34$, a realized change of $34 - 20 = +14$ and a residual $R_F = 14 - (-18) = 32$. The whole residual comes from the shared net: each single-move entry credits it with a shortening from 10 to 1, while the swap shortens it only from 10 to 8.

Feasibility: take both macros of width and height $\frac{1}{2}$, centered on the line $y = \frac{1}{2}$ of a 14×1 canvas. Two macros on one horizontal line are legal when their centers are at least $\frac{1}{2}$ apart. The center gaps are 10 initially, 9, 1, 1 and 9 in the four single moves, and 8 after the swap, so every placement involved is legal; all centers lie in $[\frac{1}{4}, \frac{55}{4}]$, inside the canvas, and the terminals are points, which impose no overlap constraint. Lane C's sequential commit would realize the full swap here, because either order of the two moves is legal, and its batch check would then reject the batch, since the incremental score rises. □

Lane C scores its matrix entries with the incremental evaluator against the current background, replaces infinite entries before solving, and commits the assigned moves one at a time, skipping illegal targets (Appendix C.6). Its batch check measures the interactions within the incremental evaluator, and the phase endpoint is rescored canonically; the example shows why those checks are needed even when the assignment is solved exactly and every entry is exact.

E Curvature diagnostic

The curvature proposals differentiate a legacy surrogate J_B (Appendix C.5) twice with automatic differentiation (AD). This appendix reports a MEASUREMENT of that operator at one saved endpoint, IBM01 after the first curvature stage (E74), and the HISTORICAL stage records it comes from. The AD Hessian-vector product is the Hessian of the branch of J_B that is active at the endpoint, the smooth piece of the piecewise function that contains it. Symmetric finite differences straddle density and RUDY branch boundaries that lie within $10^{-7}\,\mu$m of the endpoint, so they measure jumps in the gradient rather than curvature; on the smooth wirelength term the two agree to about 4×10^{-9}. J_B therefore has no classical Hessian at this endpoint. Its negative eigenvalue, -0.442458, belongs to the active branch, which still makes its eigenvector a usable candidate direction.

Negative active-branch eigenvalues are expected from the form of J_B. Within one branch, the overlap of a rectangle with a grid cell is a product $o_x(x_i)\, o_y(y_i)$ of two piecewise-linear factors, whose 2×2 Hessian block in (x_i, y_i) has the form $\left(\begin{smallmatrix} 0 & c \\ c & 0 \end{smallmatrix}\right)$ with eigenvalues $\pm|c|$. The density and RUDY terms are sums of such products, and only the convex wirelength term can offset them. The 13 Ritz values of the probe's Krylov space run from -0.442 to $+0.354$, nearly symmetric about zero.

E.1 IBM01 endpoint probe

Table 8 gives the probe's finite-difference checks.

The probe evaluates the frozen legacy curvature surrogate J_B at the saved IBM01 endpoint of the first curvature stage, in float32 and without any optimization or legalization. At that point $J_B = 0.86905$ and $\|\nabla J_B\|_2 = 8.09 \times 10^{-2}\,\mu\text{m}^{-1}$. A Lanczos run capped at 14 Hessian-vector products (HVPs) of

the automatic-differentiation (AD) operator returned Rayleigh quotient -0.442458 with residual norm 3.49×10^{-7}. The historical cascade recorded the smallest eigenvalue -0.442458 at the same endpoint; the two values differ by 1.49×10^{-7}.

Let x be the saved coordinate vector, $g_{\mathrm{AD}}(x)$ its AD gradient, and $H_{\mathrm{AD}}v$ the AD HVP for a unit vector v over the movable coordinates. The central differences are

$$d_{\mathrm{FD}}(\epsilon) = \frac{J_B(x+\epsilon v) - J_B(x-\epsilon v)}{2\epsilon}, \qquad k_{\mathrm{FD}}(\epsilon) = v^\top \frac{g_{\mathrm{AD}}(x+\epsilon v) - g_{\mathrm{AD}}(x-\epsilon v)}{2\epsilon}.$$

Positions and ϵ are in micrometers, so slopes are in $\mu\mathrm{m}^{-1}$ and curvatures in $\mu\mathrm{m}^{-2}$; the surrogate itself is dimensionless.

Table 8: Float32 central differences along two seeded random unit directions at the saved IBM01 endpoint, all movable coordinates included. The AD columns are the slope and curvature of the branch active at x; the FD columns sample both sides of nearby branch boundaries. Slopes are in $\mu\mathrm{m}^{-1}$ and curvatures in $\mu\mathrm{m}^{-2}$. The directions are independent of the Lanczos vector.

Dir.	ϵ (μm)	$g_{\mathrm{AD}}^\top v$	d_{FD}	$v^\top H_{\mathrm{AD}} v$	k_{FD}
1	1.00×10^{-3}	-1.33×10^{-4}	-5.36×10^{-4}	3.44×10^{-3}	-2.88×10^{-1}
1	3.00×10^{-3}	-1.33×10^{-4}	-5.66×10^{-4}	3.44×10^{-3}	-1.01×10^{-1}
1	1.00×10^{-2}	-1.33×10^{-4}	-5.33×10^{-4}	3.44×10^{-3}	-1.28×10^{-2}
1	3.00×10^{-2}	-1.33×10^{-4}	-5.00×10^{-4}	3.44×10^{-3}	9.50×10^{-3}
1	1.00×10^{-1}	-1.33×10^{-4}	-5.35×10^{-4}	3.44×10^{-3}	1.82×10^{-2}
2	1.00×10^{-3}	-7.68×10^{-4}	-3.28×10^{-4}	4.93×10^{-4}	-3.12×10^{-2}
2	3.00×10^{-3}	-7.68×10^{-4}	-2.09×10^{-4}	4.93×10^{-4}	2.07×10^{-3}
2	1.00×10^{-2}	-7.68×10^{-4}	-1.88×10^{-4}	4.93×10^{-4}	1.49×10^{-2}
2	3.00×10^{-2}	-7.68×10^{-4}	-2.17×10^{-4}	4.93×10^{-4}	1.71×10^{-2}
2	1.00×10^{-1}	-7.68×10^{-4}	-3.15×10^{-4}	4.93×10^{-4}	2.09×10^{-2}

For the full vectors, with $h_{\mathrm{FD}} = [g_{\mathrm{AD}}(x+\epsilon v) - g_{\mathrm{AD}}(x-\epsilon v)]/(2\epsilon)$, the relative difference $\|h_{\mathrm{FD}} - H_{\mathrm{AD}}v\|_2 / \|H_{\mathrm{AD}}v\|_2$ ranges from 7.96 to 776 over the ten rows, and the cosine between the two vectors from -1.36×10^{-1} to 6.74×10^{-2}. At the endpoint, 21 coordinates sit at or beyond a canvas clamp. For direction 1 at $\epsilon = 10^{-3}\,\mu$m the forward and backward cost slopes are -8.34×10^{-4} and $-2.38 \times 10^{-4}\,\mu\mathrm{m}^{-1}$, so the two sides already lie on different branches.

Precision and branch follow-up. A separate saved probe restricts both directions to the 2,233 coordinates whose canvas clearance exceeds $0.1\,\mu$m in both precisions, so no perturbation can cross a canvas clamp. All forward floating operations run in float64; stored float32 constants are promoted without regeneration. With P selecting these coordinates, the comparison uses $PH_{\mathrm{AD}}Pv$ and differences of Pg_{AD}. The frozen composite is also split into its three weighted primitives, which reproduce the composite value, slope and curvature to 10^{-14}. Table 9 reports the gradient jump $\|P[g(x+\epsilon v) - g(x-\epsilon v)]\|_2$ in $\mu\mathrm{m}^{-1}$.

Inside the guarded region the wirelength gradient changes in proportion to ϵ, and its finite-difference HVP agrees with the AD product to 3.91×10^{-9} and 3.45×10^{-9} at $\epsilon = 10^{-5}\,\mu$m. The density and RUDY gradients jump by amounts that persist down to $\epsilon = 10^{-7}\,\mu$m, where the $\pm\epsilon$ samples already differ in 311 recorded min/max/clamp comparisons of the composite in each direction. At $\epsilon = 10^{-3}\,\mu$m the composite jump norms are 3.86×10^{-2} and 4.42×10^{-2}, and the relative HVP differences are 752.48 and 670.50.

Table 9: Guarded float64 follow-up at the saved IBM01 endpoint: gradient jump across $\pm\epsilon$ (in μm^{-1}) and the relative difference between the finite-difference and AD Hessian-vector products at $\epsilon = 10^{-5}\,\mu$m. A smooth term has a jump proportional to ϵ; a jump that does not shrink with ϵ marks a branch boundary.

Dir.	Term	$\epsilon = 10^{-7}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-3}$	Rel. HVP diff.
1	Wirelength	1.51×10^{-10}	1.51×10^{-8}	1.51×10^{-6}	3.91×10^{-9}
1	$\frac{1}{2}\times$density	6.01×10^{-3}	2.20×10^{-2}	3.85×10^{-2}	4.52×10^{5}
1	$\frac{1}{2}\times$RUDY	2.50×10^{-3}	2.61×10^{-3}	2.61×10^{-3}	5.17×10^{3}
1	Composite J_B	6.51×10^{-3}	2.22×10^{-2}	3.86×10^{-2}	4.33×10^{4}
2	Wirelength	1.54×10^{-10}	1.54×10^{-8}	1.54×10^{-6}	3.45×10^{-9}
2	$\frac{1}{2}\times$density	6.01×10^{-3}	2.15×10^{-2}	3.84×10^{-2}	4.11×10^{5}
2	$\frac{1}{2}\times$RUDY	2.50×10^{-3}	2.61×10^{-3}	2.17×10^{-2}	3.98×10^{3}
2	Composite J_B	6.51×10^{-3}	2.16×10^{-2}	4.42×10^{-2}	3.28×10^{4}

E.2 Stage records and endpoint validity

Table 10 gives the per-design records of the two curvature stages of the v1 lineage, from staged research runs without the one-hour limit. On top of the time needed to reach the plateau, the first stage took up to 4,007 s on one design and the cascade up to 3,454 s. The first stage lowered the mean plateau score from 1.08128 to 1.06760 and the cascade reached 1.06119 (Figure 7a). In the first stage, 201 of 204 perturb-and-polish attempts could be legalized and 190 ended below the plateau with zero overlap, with nearly equal success for the first and second eigenvectors (94 and 96 of 102) and for total displacement norms of 0.3, 1 and 3 μm summed over all blocks (64, 65 and 61 of 68; Figure 7b). The attempts are 12 on each of 17 designs, so they are not independent.

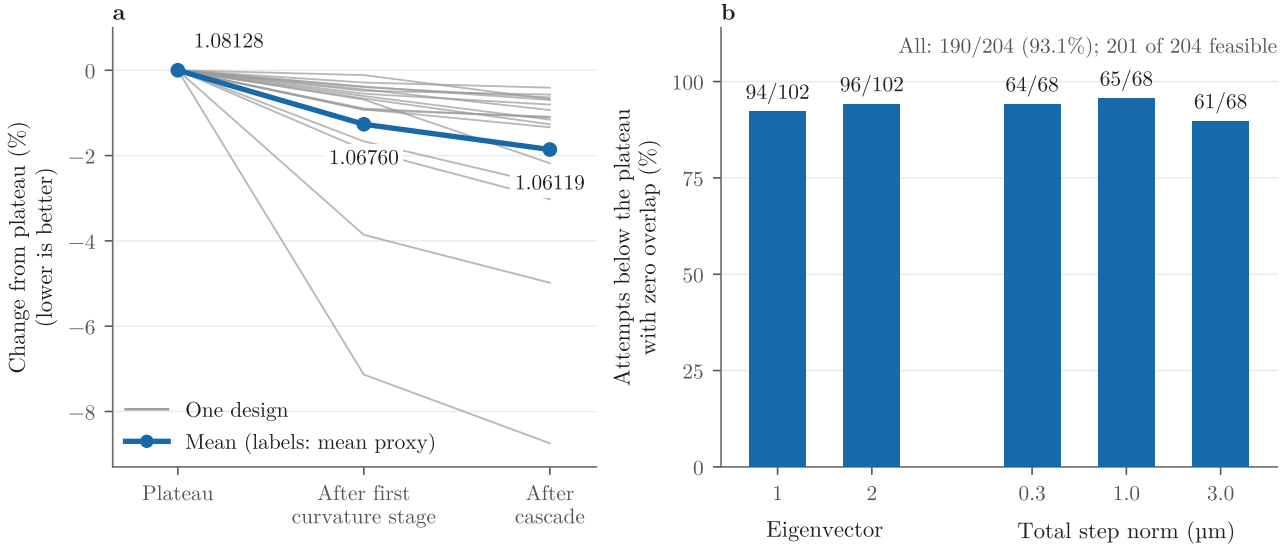


Figure 7: Curvature kicks (perturb along a computed negative-curvature direction, then legalize and re-polish) in v1 research runs without the one-hour cap (HISTORICAL). (a) Score change relative to the plateau, the best v1 result before any kick, per design and on average. (b) Share of first-stage attempts ending below the plateau with zero overlap, by eigenvector and step length.

All 34 saved endpoints (first stage and cascade on 17 designs) were rescored with the canonical evaluator for this report. The fresh scores reproduce the recorded ones to within 9.87×10^{-8}, so every recorded decrease persists, and none of the 34 placements has a hard-macro overlap. A strict geometry check, which also requires every rectangle to lie inside the canvas, accepts 22 of the 34: 12 of the 17 first-stage endpoints and 10 of the 17 cascade endpoints. In each of the 12 failures an object extends

Table 10: Historical records from uncapped research runs (no one-hour limit). Proxy cost after each curvature stage (lower is better); the plateau is the better of the two v1 branches. Validity is a strict canvas check of the saved endpoints (Measurement); every failure exceeds a canvas edge by at most $7.63 \times 10^{-6}\,\mu$m and no endpoint has a hard-macro overlap.

Design	Plateau	After first curvature stage	After cascade	Strict validity (first / cascade)	Cascade stage time (s)
IBM01	0.88960	0.85527	0.84528	✓ / ✓	2,913
IBM02	1.11890	1.03907	1.02104	✓ / ✓	3,285
IBM03	0.95849	0.95214	0.94634	✓ / ×	3,431
IBM04	1.00156	0.99225	0.99066	✓ / ✓	375
IBM06	1.15276	1.13127	1.11792	✓ / ✓	3,298
IBM07	1.09397	1.07574	1.06449	× / ×	3,381
IBM08	1.11106	1.10606	1.10069	✓ / ✓	3,319
IBM09	0.82828	0.82735	0.82269	× / ×	3,360
IBM10	1.00051	0.99155	0.98940	✓ / ✓	2,880
IBM11	0.87824	0.87293	0.86808	✓ / ✓	3,454
IBM12	1.20534	1.20058	1.19766	✓ / ✓	3,432
IBM13	0.95540	0.94883	0.93458	× / ×	3,413
IBM14	1.21609	1.20940	1.20630	✓ / ✓	3,284
IBM15	1.16172	1.15612	1.15510	× / ×	2,733
IBM16	1.12347	1.11320	1.10847	× / ×	3,287
IBM17	1.33708	1.33323	1.33162	✓ / ×	3,423
IBM18	1.34934	1.34419	1.33994	✓ / ✓	3,340
Mean	1.08128	1.06760	1.06119	12 / 10 of 17	3,095

past the upper canvas edge by exactly one float32 rounding step at the canvas size: $3.81 \times 10^{-6}\,\mu$m on canvases of 32 to 64 μm (IBM03, IBM07, IBM09, IBM13) and $7.63 \times 10^{-6}\,\mu$m on canvases of 64 to 128 μm (IBM15, IBM16, IBM17). The excess is consistent with clipping in float64 followed by conversion to float32; the corrected runtime instead steps a rounded-up coordinate one representable value inward (Appendix F). The coordinates were left as saved, so the stage means include the 12 strictly invalid placements. On the 10 designs whose two endpoints both pass, the mean falls from 1.074258 to 1.067697 (−0.61%), lower on all 10.

Figure 8 shows what the cascade changed on IBM01, a design whose two endpoints are both strictly valid. It moved 104 of the 246 hard macros, by at most 1.01 μm on a canvas 23 μm wide; the median displacement among the moved macros is 0.085 μm. It also moved 407 of the 894 soft clusters, by up to 8.4 μm (not drawn). The score fell by 0.00999 through many small, scattered adjustments.

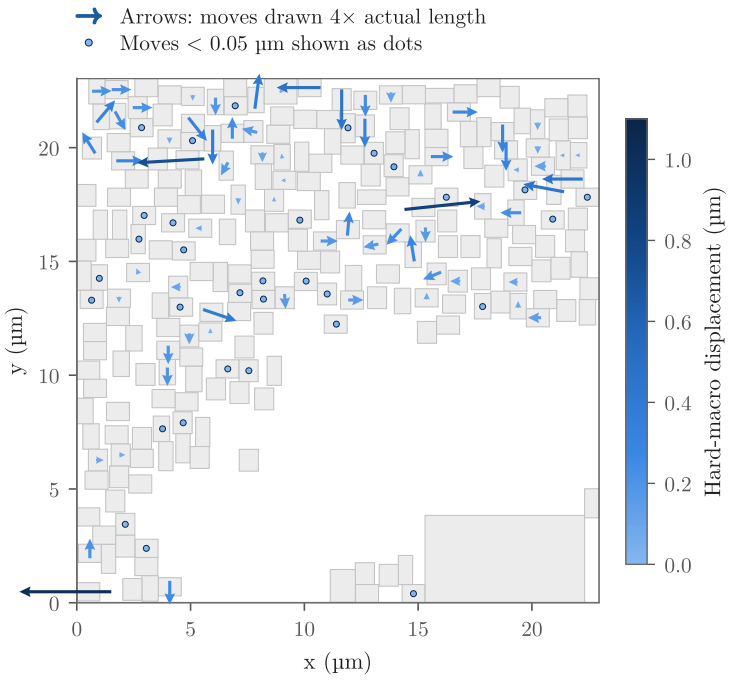


Figure 8: Hard-macro displacement on IBM01 between the saved endpoints of the first curvature stage (proxy 0.85527) and the cascade (proxy 0.84528), HISTORICAL. Arrows start at the first-stage position and are drawn four times longer than the actual displacement, moves under 0.05 μm are drawn as dots, and color gives the length. The difference includes the CD polish after every accepted step, so it is not a trajectory.

F Post-competition verification

In September 2026 we audited the competition code against the challenge rules and our own runtime assumptions, repaired what we found in a separate post-competition runtime, and ran the repairs in short engineering runs on an Apple M3 Max. Below are the defects and whether each could have affected the judged entry, the evaluator microbenchmarks behind the main text's speed and agreement figures, and the engineering runs, including two stopped reproduction attempts.

F.1 Defects and their scope

Table 11 lists each defect, where it lived in the competition code, whether it could have affected the judged 20 May IBM entry, the repair and the evidence that the repair works. The judged entry is inferred to be the single pipeline of Appendix C.7; its listed runtime of about 12 minutes per design matches its 720 s budget. Most rows are settled by two facts. That pipeline runs in one process without lane workers, and every hard-macro name in the 17 IBM netlists is flat, so the first "/" of a pin name always separates the owning macro. A frozen record checks this on all 17 pinned IBM inputs (`writeup/data/publication/ibm_repair_equivalence.json`, written by `scripts/measure_publication_ibm_repair_equivalence.py`; MEASUREMENT). It compares the extraction code of 20 May (commit `5485310`) with the corrected runtime without optimizing or scoring: the two ownership rules agree on all 411,642 macro pins, and the smooth net tensors and the initializer's hard-macro edge lists are identical on 17 of 17 designs. The same record finds no fixed macros, and soft clusters outside the canvas in the inputs of 15 designs (1 to 11 per design), which the corrected runtime's initial clamp moves. The "Evidence" column refers to the focused tests of Appendix H and to the frozen diagnostics.

F.2 Matched incremental-evaluator measurements

We measured identical prescribed move sequences through two evaluation paths: incremental updates returning wirelength, density, congestion, and proxy; and the canonical placement setter followed by the three full component computations. Neither timed path includes hard-overlap validation. Both use float64 positions on the Mac CPU with one PyTorch thread. The order of the two engines alternates between repetitions; initialization and a warm-up component evaluation precede each batch.

Table 11: Defects found by the post-competition audit. "Judged entry" answers whether the defect could have affected the 20 May IBM entry: *no*, *unknown*, or *note* for an item that is not a defect. Tests are those of Appendix H; "IBM input record" is `ibm_repair_equivalence.json`.

Defect	Where	Judged entry	Fix	Evidence
Lane join became `timeout=None` after the deadline	Three-lane supervisor	No: the single pipeline has no lanes to join	Shared monotonic deadline; zero join timeout after expiry; external process-group timeout	Join and termination tests; IBM17 runs stopped at 240.0–240.1 s (limit 240 s)
Stage polish caps exceed the stage budget	Curvature stage; v1 cascade and portfolio	No: neither stage is in the single pipeline	Caps charged for eigensolver and legalizer time; deadline checked between HVPs	Three budget tests
Lane result written only on normal completion	Lane worker	No: the single pipeline returns its placement directly	Atomic, validated checkpoint per phase; best valid one recovered	Checkpoint tests; three recoveries in each 12-run batch
Temporary lane directories not removed	Three-lane supervisor	No: no lane directories, and disk residue cannot change a placement	Cleanup on every exit path	Cleanup tests
Pin owner taken from the pin name before its first "/"	Surrogate net extraction (`_extract_net_data`)	No: IBM names are flat, verified on 17 of 17 designs (0 owner mismatches over 411,642 macro pins; identical net tensors); IBM01 keeps 2,978 hard-pin occurrences and 5,993 retained nets	Owner from the parser's pin object	IBM input record; ariane133: all 5,498 used hard-pin occurrences restored; four tests
Same split in the initializer's macro graph	`sdf_init.py`	No: IBM names are flat; edge lists identical on 17 of 17 designs (IBM01: 644 edges)	Map from the parser's hard-macro pins	IBM input record; ariane133: 0 to 859 edges; replayable record; four tests
`qualified` set from the overlap count alone	Local result writer	No: the organizers scored with their own harness	Validity and eligibility recorded separately	22 of 34 overlap-free historical endpoints pass strict containment
Input soft clusters outside the canvas not repaired	Initializer; push-apart	Unknown: no judged coordinates survive; 15 of 17 IBM inputs have 1 to 11 such clusters, which the descent pushes inward without guaranteeing containment; the organizers verified the entry	Every movable center projected into the canvas	IBM input record; v1 failed on IBM10 and IBM17 in the pilot and passed on IBM10 after repair; soft-bound tests
Float32 export can round past the upper canvas edge	Final conversion after clipping	Unknown: the single pipeline also exports float32; the organizers verified the entry	Step one representable value inward	12 of 34 historical endpoints exceed by one step; one test
NG45 lookup failed on the `_ng45` name suffix	`find_benchmark_dir`	No: IBM names have no suffix	Explicit aliases, exact name first	Pilot ariane133 failures; later batches valid; three tests
Golden-section search unreachable	CD axis search	Note: it never runs	None needed	At most ten candidates; list capped at 12

Table 12: CPU component-kernel microbenchmarks. Times are means per move. Each ratio is the full-recomputation time divided by the incremental time for one paired batch; the table gives the median and range over three paired batches for the IBM01 sample and two paired batches for each IBM10 sample (the median of two batches is their mean). Setup and overlap checks are excluded; these are kernel measurements, not end-to-end placer speedups.

Design / sample	Moves × repeats	Inc. time (ms)	Full time (ms)	Median ratio	Range
IBM01 / mixed	8×3	1.29	1,401.6	1,040×	1,035–1,197
IBM10 / soft	3×2	3.67	26,682.1	7,269×	7,055–7,483
IBM10 / hard	3×2	1.49	26,865.4	18,023×	17,250–18,795

Every measured component agreed within relative tolerance 10^{-5}, using $|a-b|/\max(|a|,|b|,10^{-30})$; the largest observed relative discrepancy was 2.6×10^{-15}. Evaluator setup is excluded from the ratios and retained in the raw records. The sequences are small: IBM01 has three hard-macro moves and five soft-cluster moves per repetition; the first IBM10 sequence happened to select only soft clusters, so a separate hard-only sequence was measured. Moves may create overlaps. The ratios compare localized cache updates with full recomputation in the reference Python evaluator, so they reflect both the update strategy and the implementation, and they exclude the rest of the placer.

Mean incremental setup times were IBM01 mixed: 0.33 s, IBM10 soft: 1.48 s, IBM10 hard: 1.45 s. A separate overlap-metrics call on each final state is also timed in the raw data; it is excluded from both component paths and is not the full geometry-validation cost. Repetitions share the same seed and moves, so their spread measures timing repeatability, not independent placement outcomes.

F.3 Engineering runs

Three batches exercised the repaired runtime with the corrected runner. They check execution, recording and legality, with solver budgets of 60 to 180 s, far below the competition's hour, and say little about placement quality. All ran on an Apple M3 Max (14 logical CPUs, 36 GiB) under macOS 26.6.2 with CPython 3.9.6 and PyTorch 2.8.0, nominal seed 42, and every planned run is reported. Each batch froze its source as a runtime archive before execution: `5730d13762ce` for the pilot, `ca4343aaefda` for the repair batch and `5aa4c719cf18` for the NG45 integration check.

Pilot. The predeclared designs were IBM01 (small), IBM10 (large), IBM17 (historically hardest) and ariane133, a hierarchical NG45 design. Each configuration used 500 analytical steps and a 30 s CD phase cap, and runs went one at a time. Table 13 gives the outcomes. The seven errors have three causes. The v1 runs on IBM10 and IBM17 failed canvas containment after local refinement, because their inputs contain movable soft clusters outside the canvas that the original initializer does not repair. On IBM17 both v2 configurations exhausted their budget without a valid checkpoint. All three ariane133 runs failed to locate the design directory, before any optimization. No run reached the external limit. Two recoveries show how the checkpoints behave: on IBM01 the three-lane run returned Lane B's checkpoint after its curvature step (0.82787) when the lanes were stopped at the solver budget, and on IBM10 the standalone and three-lane runs returned the same legalized-descent state (1.16686). Final validation and export took 26.5–28.2 s on IBM10 against about 1.5 s on IBM01.

Coverage by variant: v1 1/4 eligible, 0/4 completed; A 2/4 eligible, 1/4 completed; A+B+C 2/4 eligible, 0/4 completed. No variant has an eligible result on every planned design, so no mean is reported. Where initializer or legalized-descent checkpoints survive for all three A+B+C lanes and for standalone A (IBM01 initializer, IBM01 legalized descent, IBM10 legalized descent), the four coordinate tensors are bit-identical; the nominal lane seeds therefore did not diversify the shared prefix. Full SHA-256 values are in the JSON summary.

Table 13: Limited-budget pilot on four designs (180 s solver budget, 240 s external limit, 1 concurrent job, 3 CPU threads per run). C: completed pipeline; P: valid result after partial phases; R: valid checkpoint recovery; E: error. Returned: lane and phase whose saved coordinates equal the returned placement. Threads: CPU threads per solver process as recorded. Every planned run is shown.

Design	Variant	Proxy	Solver (s)	Wall (s)	Outcome	Returned	Threads
IBM01	v1	0.92786	140.8	145.1	P	E25	3
IBM01	A	0.83617	51.3	54.4	C	A/final	3
IBM01	A+B+C	0.82787	180.0	183.0	R	B/curvature	1
IBM10	v1	—	—	70.1	E	—	3
IBM10	A	1.16686	180.0	209.5	R	A/descent	3
IBM10	A+B+C	1.16686	180.0	211.0	R	ABC/descent	1
IBM17	v1	—	—	71.0	E	—	3
IBM17	A	—	—	183.8	E	—	—
IBM17	A+B+C	—	—	183.6	E	—	—
ariane133	v1	—	—	2.1	E	—	3
ariane133	A	—	—	3.1	E	—	—
ariane133	A+B+C	—	—	3.0	E	—	—

Repair batch. The second batch adds the all-movable bounds repair, fixed-reference preservation, the NG45 aliases and retained phase diagnostics, and runs the same 12 cells three at a time, still with three CPU threads per run (Table 14). Standalone A used all three threads in its one lane; the three-lane runs gave one thread to each lane. That allocation, and the phase each run reached, explain the IBM10 contrast: A returned its legalized descent (1.14944) while the three-lane run returned its initializer (1.41116), so the two values measure progress under different resources, not the value of the extra lanes. The ariane133 rows still use the original soft-rectangle edge extractor; its repair came after this batch.

Table 14: Post-pilot repair validation on four designs (180 s solver budget, 240 s external limit, 3 concurrent jobs, 3 CPU threads per run). C: completed pipeline; P: valid result after partial phases; R: valid checkpoint recovery; T: external timeout. Returned: lane and phase whose saved coordinates equal the returned placement. Threads: CPU threads per solver process as recorded. Every planned run is shown.

Design	Variant	Proxy	Solver (s)	Wall (s)	Outcome	Returned	Threads
IBM01	v1	0.93106	151.8	155.2	P	E25	3
IBM01	A	0.83681	53.6	56.9	C	A/final	3
IBM01	A+B+C	0.82961	180.0	183.2	R	B/curvature	1
IBM10	v1	1.22215	125.3	169.1	P	E25	3
IBM10	A	1.14944	180.0	225.6	R	A/descent	3
IBM10	A+B+C	1.41116	180.0	216.2	R	ABC/init.	1
IBM17	v1	—	—	240.1	T	—	3
IBM17	A	—	—	240.0	T	—	—
IBM17	A+B+C	—	—	240.0	T	—	—
ariane133	v1	0.65572	172.8	181.6	P	periphery	3
ariane133	A	0.64930	81.1	89.0	C	A/final	3
ariane133	A+B+C	0.64911	180.0	187.8	P	A/final	1

Coverage by variant: v1 3/4 eligible, 0/4 completed; A 3/4 eligible, 2/4 completed; A+B+C 3/4 eligible, 0/4 completed. No variant has an eligible result on every planned design, so no mean is reported. Where initializer or legalized-descent checkpoints survive for all three A+B+C lanes and for standalone A (IBM01 initializer, IBM01 legalized descent, IBM10 initializer, ariane133 initializer,

ariane133 legalized descent), the four coordinate tensors are bit-identical; the nominal lane seeds therefore did not diversify the shared prefix. Full SHA-256 values are in the JSON summary.

NG45 integration. Table 15 lists the three runs. This check ran all three entry configurations on ariane133 from runtime archive `5aa4c719cf18`, which adds the SDF graph repair. The profile used a 60-second solver budget, a 120-second external limit, three concurrent jobs and three CPU threads per run. A and A+B+C used five analytical steps and zero-second CD phase budgets; v1 kept its budget-derived refinement stages. All three runs returned valid, study-eligible placements within the external limit, and A completed its configured pipeline. The check exercises connectivity extraction and finalization on a hierarchical netlist; its settings are too short for a quality comparison, so proxies are omitted.

Table 15: Final ariane133 integration after the SDF graph repair (60 s solver budget, 120 s external limit). C: completed pipeline; P: valid result after partial phases. Returned and Threads as in Table 13.

Variant	Solver (s)	Wall (s)	Outcome	Returned	Threads
v1	34.6	44.0	P	E25	3
A	45.7	53.7	C	A/final	3
A+B+C	60.0	67.8	P	A/final	1

IBM17 timeouts and finalization cost. The three IBM17 runs of the repair batch are the only runs that reached the external limit, and the saved artifacts show where the time went. Both v2 configurations logged lane selection before the limit, so the remaining time was spent after the placer returned, where the worker validates the placement and calls the canonical scorer once more before saving it. The v1 run wrote its first checkpoint at 192.7 s, although its inner log reports 66.4 s of search; the gap covers differing timer origins and the canonical scoring that precedes every checkpoint write. One canonical score of IBM17 takes about 73 s on this host with one thread (72.8 and 73.3 s for the two IBM17 endpoints rescored in Appendix E), against 1.3 s for IBM01. That single call is longer than the 60 s between the 180 s solver budget and the 240 s limit. For the planned one-hour runs, the study configuration therefore stops the solver at 3,300 s and reserves the last five minutes for validation and export.

Reproduction attempts (engineering record). The reproduction of Section 4.2 runs the single pipeline at 720 s per design on all 17 IBM designs, one design at a time on the CPU with three threads, on the same M3 Max, by then under macOS 27.0.1. It took three attempts. The operator stopped the first two after each exposed a defect in the corrected runtime; their records are engineering records, not a suite result, and are not compared with any other record.

Attempt 1 (30 September 2026; commit `cb554f1`, runtime `5aa4c719cf18`) was stopped during IBM06, which has no result. IBM01, IBM02 and IBM04 completed the pipeline (0.82996, 1.00200 and 0.91909). On IBM03 coordinate descent completed, but its final checkpoint failed strict validation ("Macros outside horizontal canvas bounds"): the per-phase checkpoint checked canvas bounds without the one-step inward projection applied at export, so a float32 rounding overshoot at the upper canvas edge discarded the coordinate-descent result, and the run returned its checkpoint from before coordinate descent (1.10002). The rejected candidate was not saved, so its excess is unknown. The fix (commit `64b50a7`, runtime `cbfbff0c81c3`) projects the placement one step inward at the checkpoint when no coordinate moves by more than $10^{-4}\,\mu$m, records that shift with each checkpoint, and still rejects larger violations.

Attempt 2 (1 October 2026; commit `64b50a7`, runtime `cbfbff0c81c3`) finished 15 designs before the operator stopped it during IBM17; IBM17 and IBM18 have no result. IBM03 completed the pipeline (0.91446), as did 12 other designs. IBM10 and IBM16 exposed a second defect. On this CPU, smooth

descent took about 118 s on IBM10 and 154 s on IBM16, and coordinate descent was then budgeted all the remaining time to the 720 s deadline (576 s and 526 s), leaving none for its final checkpoint and score, two canonical evaluations of about 27 s each on IBM10. The supervisor stopped the lane at the deadline, so both runs returned their last valid checkpoint, from before coordinate descent (1.14944 and 1.14491). The fix (commit `d9f2577`, runtime `cdd01ecc7c98`) times every checkpoint and caps each coordinate-descent phase so that 10 s plus twice the slowest measured checkpoint still fits before the deadline.

The reproduction reported in Section 4.2 is attempt 3, run on 1–2 October 2026 from commit `d9f2577` with runtime `cdd01ecc7c98`. The records of all three attempts (manifests, requests, results, logs, placements and checkpoints) are kept under `writeup/runs/`, in `ranked-reproduction-20260930/`, `ranked-reproduction-20261001/` and `ranked-reproduction-20261001-r3/`.

G Suite records and evidence ledger

G.1 Historical suite records

Three complete 17-design records survive from May 2026 (Table 16). Each is an evaluator output with per-design components, overlap counts, runtimes, a timestamp, four concurrent jobs and one Torch thread; none stores coordinates, a source revision or host details, so the scores can be recomputed from their components but not from placements. Contemporaneous notes place the 17 May v1 run on the M3 Max and the early v2 run on an AWS EPYC instance; the 0.98199 record names no host. The records differ in program, hardware and time budget, so a difference between columns cannot be assigned to any one change.

Table 16: Historical records; hardware, budgets and concurrency differ. Proxy cost per design (lower is better) for three development records and the two published baselines listed by the challenge. Each record is below both baselines on all 17 designs.

Design	SA	RePlAce	v1 (17 May)	Early v2	Single pipeline (20 May lineage)
IBM01	1.3166	0.9976	0.85963	0.84658	0.82152
IBM02	1.9072	1.8370	1.00659	0.99484	1.01265
IBM03	1.7401	1.3222	0.92742	0.90815	0.88836
IBM04	1.5037	1.3024	0.97285	0.94491	0.92131
IBM06	2.5057	1.6187	1.13361	1.07574	1.05296
IBM07	2.0229	1.4633	1.05294	1.03333	1.01486
IBM08	1.9239	1.4285	1.08163	1.04295	1.02490
IBM09	1.3875	1.1194	0.82494	0.78161	0.77033
IBM10	2.1108	1.5009	0.99575	0.98969	0.97934
IBM11	1.7111	1.1774	0.86041	0.80567	0.78831
IBM12	2.8261	1.7261	1.19196	1.14974	1.08989
IBM13	1.9141	1.3355	0.93633	0.85447	0.84279
IBM14	2.2750	1.5436	1.19081	1.10989	1.07330
IBM15	2.3000	1.5159	1.15820	1.08974	1.05557
IBM16	2.2337	1.4780	1.12240	1.03277	0.99607
IBM17	3.6726	1.6446	1.32264	1.24247	1.17164
IBM18	2.7755	1.7722	1.33932	1.23934	1.18998
Mean	2.1251	1.4578	1.05750	1.00835	0.98199

Against the published baselines, the margin below RePlAce ranges from 13.8% (v1, IBM01) to 45.8% (early v2, IBM02). The 0.98199 record is lower than the v1 record on 16 of 17 designs. The exception is IBM02: density falls from 0.61156 to 0.57491, but wirelength rises from 0.08418 to 0.08897 and

congestion from 1.23324 to 1.27245, so the total rises from 1.00659 to 1.01265. The largest change is on IBM17, where congestion falls from 1.88679 to 1.66501 and density from 0.60911 to 0.53604, a total decrease of 0.15100; IBM17 nevertheless remains among the costliest designs.

Repeatability. One direct repeatability observation survives, as a HISTORICAL note rather than a frozen record: a development note (commit `4a202de`) records an identical rerun of the 17 May v1 configuration on the same M3 Max over a four-design subset. The subset mean moved from 0.89844 to 0.89517 (−0.36%); IBM01 moved from 0.85963 to 0.84691 (−1.48%), IBM04 was unchanged, and IBM09 and IBM13 moved by −0.20% and +0.14%. This is one repeat pair, of a different pipeline from the ranked one and too few for a confidence interval, but it shows that run-to-run differences of a few tenths of a percent in a subset mean, and over 1% on one design, occur without any code change.

Lineage of the judged entry. The final leaderboard row reads "Resubmitted 5/20 (thinkorplace-v2). Was 1.0771" and names no commit, archive or result digest. We infer that the judged program was the 20 May single pipeline, most likely commit `5485310`, from three premises: the resubmission date; the listed runtime of about 12 minutes per design, which matches that pipeline's 720 s budget rather than the three-lane system's 2,700 s; and the upstream leaderboard of 21 May, which listed the pending resubmission with a self-reported score consistent with the single-pipeline lineage and not with the early-v2 record at 1.00835 (the provenance note in Appendix G.3 gives the figure). The three-lane commits of 22 May (Eastern time) still preceded the Pacific-time deadline, so chronology alone cannot exclude them; the runtime and the resubmission date do.

G.2 Reproduction

Table 17 gives each design of the rerun in Section 4.2 (REPRODUCTION), rendered from its run records by `scripts/render_publication_reproduction.py`.

Table 17: Reproduction (post-competition), ranked configuration rerun on the lead author's Apple M3 Max CPU, one design at a time; all values from the canonical evaluator. Proxy = wirelength + $\frac{1}{2}$ density + $\frac{1}{2}$ congestion. Ratio to RePlAce: each proxy over the published RePlAce value for its design; in the mean row, the mean proxy over the RePlAce mean (1.4578). Wall: supervised time per design, including final validation (720 s solver budget, 3,600 s limit). ✓: full pipeline completed.

Design	Proxy	Wirelength	$\frac{1}{2}$ Density	$\frac{1}{2}$ Congestion	Ratio to RePlAce	Wall (s)	Outcome
IBM01	0.82996	0.08442	0.26335	0.48218	0.832	329	✓
IBM02	1.00200	0.08971	0.28751	0.62479	0.545	364	✓
IBM03	0.91446	0.10122	0.25211	0.56113	0.692	352	✓
IBM04	0.91909	0.08946	0.25343	0.57620	0.706	346	✓
IBM06	1.03906	0.08057	0.25438	0.70411	0.642	348	✓
IBM07	1.00657	0.08705	0.26236	0.65716	0.688	361	✓
IBM08	1.01842	0.08599	0.25798	0.67445	0.713	375	✓
IBM09	0.77703	0.06776	0.25831	0.45097	0.694	365	✓
IBM10	0.98044	0.08049	0.25927	0.64068	0.653	743	✓
IBM11	0.78998	0.06582	0.25516	0.46900	0.671	396	✓
IBM12	1.08597	0.07966	0.26455	0.74177	0.629	721	✓
IBM13	0.84098	0.06592	0.26020	0.51486	0.630	404	✓
IBM14	1.06636	0.06678	0.26922	0.73037	0.691	726	✓
IBM15	1.05107	0.07465	0.29701	0.67941	0.693	573	✓
IBM16	1.00323	0.06560	0.27215	0.66549	0.679	754	✓
IBM17	1.17400	0.07126	0.26957	0.83316	0.714	782	✓
IBM18	1.18197	0.07536	0.29480	0.81180	0.667	539	✓
Mean	0.98121	0.07834	0.26655	0.63633	0.673	499	

G.3 Evidence ledger

Table 18 maps each quantitative claim of the report to its evidence label and record.

Table 18: Claim-to-evidence map. OFFICIAL: the organizers' final leaderboard. HISTORICAL: May 2026 development records. MEASUREMENT: post-competition diagnostics on frozen inputs. REPRODUCTION: the post-competition run of the configuration inferred to be ranked.

Claim	Label	Record	Scope
Ninth on the final leaderboard, verified mean 0.9974; best 0.7829, worst 1.2602	OFFICIAL	Upstream README at `996193c`	Inferred to be the 20 May single pipeline; judged commit not recorded
Winner 0.9507; SA and RePlAce baseline means 2.1251 and 1.4578	OFFICIAL	Same README; baselines from Cheng et al. [7]	Organizers' numbers, not reruns
v1 (17 May) mean 1.05750	HISTORICAL	Frozen evaluator JSON	Components only; no coordinates
Early v2 mean 1.00835	HISTORICAL	Frozen evaluator JSON	Earlier smooth placer (V3); different host
0.98199 record, lower than v1 on 16 of 17	HISTORICAL	Frozen evaluator JSON	Closest surviving full-suite record of the ranked lineage; unmatched conditions
Congestion supplies 84.5% of the decrease from the v1 record to the 0.98199 record	HISTORICAL	Component fields of the two records	Arithmetic attribution, not a mechanism
Plateau 1.08128, first curvature stage 1.06760, cascade 1.06119	HISTORICAL	Frozen stage records and 34 placements	Uncapped research runs
Rescoring within 9.87×10^{-8}; 22 of 34 endpoints strictly valid	MEASUREMENT	Rescore and geometry records	Coordinates left unrepaired
190 of 204 first-stage attempts below the plateau	HISTORICAL	Frozen stage records	No random-direction or zero-step control; 12 attempts per design
Incremental update 1.29–3.67 ms versus 1.40–26.87 s; largest error 4.44×10^{-16}	MEASUREMENT	Frozen microbenchmarks	14 sampled moves on IBM01 and IBM10, single-move path only; not a uniform bound
IBM01 curvature: active-branch eigenvalue -0.442458; wirelength HVPs agree to 4×10^{-9}	MEASUREMENT	Frozen diagnostic and follow-up	One surrogate, one endpoint
Ten defects and repairs (5,498 pin occurrences, 859 edges)	MEASUREMENT	Topology records; focused tests	Extraction and execution, not placement quality
Repairs leave all 17 IBM inputs unchanged; 15 have soft clusters outside the canvas	MEASUREMENT	IBM repair-equivalence record	Input extraction only; no optimization or scoring
Engineering batches	MEASUREMENT	Frozen run records	60–180 s budgets; no suite mean
Repeatability -0.36% (subset), -1.48% (IBM01)	HISTORICAL	Development note, not frozen	One repeat pair of v1
Rerun of the configuration inferred to be ranked, on a laptop CPU: mean 0.98121	REPRODUCTION	Its own run records	Reported in Section 4.2 and Table 17
Full-suite score of the three-lane system	—	None	Never scored (inferred)

Provenance only: three further numbers appear in May-era notes and listings. The upstream leaderboard of 21 May showed the pending resubmission with a self-reported 0.991; 0.98387 is an extended-CD standalone run; and 0.97745 is an offline per-design minimum across component runs. None has a frozen record or is a result of this report.

H Reproducibility

The code, frozen evidence and report sources are in the public repository https://github.com/brmurrell3/macro-placer. Commands below run from the repository root with the project's virtual environment.

Source revisions. The report depends on four source revisions. The competition snapshot `2ab92a33d352` (22 May 2026) is the last competition-era commit; its Python source closure is frozen as `writeup/data/publication/competition-source.tar`, described by `source_snapshot.json` with the archive digest and file list. The upstream challenge revision `996193c83eaa` (14 August 2026) carries the final leaderboard. The TILOS MacroPlacement submodule is pinned at `45a721d01dfe`; six canonical evaluator, loader and validation files are byte-identical between the local base and the final challenge revision. The ranked single pipeline is most likely commit `5485310`; Appendix G gives the basis for that inference.

Package extraction. The competition entries imported code from the experiment tree, so copying a submission directory onto a clean checkout of the challenge was not enough to run it. The post-competition cleanup extracts the active dependency closure into the package `thinkorplace/` with three parts, `thinkorplace.common`, `thinkorplace.v1` and `thinkorplace.v2`, using relative imports; the submission entry points remain as thin adapters. Runtime imports no longer touch `experiments/` or `writeup/`. `thinkorplace/MIGRATION.json` records the origin and source digest of every module. The package keeps the optimization formulas and adds the repairs of Appendix F; a test checks that the extracted primitives preserve the original algorithms' syntax trees.

Frozen evidence and generators. `writeup/data/publication/` holds the evidence, and its `README.md` states the scope of each record:

- `raw/`: the three historical suite records, the 34 curvature-stage records and the 34 saved placement tensors, with SHA-256 values in `historical_summary.json`;
- the evaluator microbenchmarks, curvature diagnostic and follow-up, rescoring and geometry records, topology diagnostics, tail-bound and error-margin arithmetic, and the frozen leaderboard;
- `ibm_repair_equivalence.json`, written by `scripts/measure_publication_ibm_repair_equivalence.py`: on all 17 IBM inputs, the 20 May extraction code (commit `5485310`) and the corrected runtime agree on pin ownership, smooth net tensors and initializer edges, and the record counts the input soft clusters that the initial clamp moves; no optimization or scoring is run;
- summaries of the three engineering batches, whose primary run records (manifests, requests, results, logs, placements and checkpoint hashes) are under `writeup/runs/`, as are those of the three reproduction attempts (`writeup/runs/ranked-reproduction-*/`);
- `reproduction_ranked.json`, written by `scripts/render_publication_reproduction.py` from the reported reproduction's run records: configuration, commit and runtime identity, host, the SHA-256 of every result record, and per-design scores, components, validity, outcomes and wall times;
- `corrected-runtime-*.tar.gz`: deterministic archives of each post-competition runtime, with per-file digests in the matching `corrected_runtime_snapshot_*.json`.

Generators read these records and write only derived summaries, tables and figures; the historical freezer refuses to replace a frozen raw record with different bytes. No generator runs a solver, and the scripts that produced the diagnostics are separate from the generators.

Runner. The study runner (`scripts/run_publication_study.py` with `scripts/publication_worker.py`) implements the following workflow:
1. create an immutable run manifest before any solver is constructed;
2. record input and initialization hashes and the explicit device choice;
3. save canonical incumbents atomically at every meaningful phase;
4. propagate one monotonic solver deadline and enforce an external process-group timeout;
5. validate the final candidate independently and retain every failure;
6. preserve final coordinates and phase metadata for deterministic post-processing.

A run's identity covers executable source, dependencies, inputs and constructor settings, so editing the paper does not invalidate a run while any runtime change does. Source is hashed again after execution. Validity (shape, finite values, containment, fixed objects, zero hard overlaps), study eligibility (return within the external limit with matching source and environment) and pipeline completion are separate fields: a valid recovered checkpoint can be an eligible anytime result without a completed pipeline.

Tests. Runtime archive `5aa4c719cf18` passed 64 focused tests in five modules; the receipt `verification/final-tests-20260906.json` binds the archive to the test log:
- `test_submission_safety.py` (30): deadline propagation, zero join timeouts after expiry, child termination and cleanup, rejection of invalid lanes, checkpoint retention and atomic replacement, phase-outcome reporting, hierarchical pin ownership in four code copies, and budget guards for the curvature stages and the eigensolver;
- `test_thinkorplace_package.py` (7): imports against a pristine upstream without the research tree, legalizer clamping of soft clusters, parity of runtime validation with the canonical check on float32 endpoints and randomized geometry, and preservation of the extracted algorithms;
- `test_postpilot_repairs.py` (12): soft-cluster bounds, fixed obstacles, float32 export, infeasible fixed references, unresolved hard geometry, NG45 directory aliases, and retained diagnostics for rejected initializers and fatal results;
- `test_sdf_topology.py` (4): initializer edge extraction for flat and hierarchical netlists in both code copies;
- `test_publication_runner.py` (11): explicit parameters, worker validation, checkpoint-recovery semantics, process-group timeouts, resume identity, dry runs and checkpoint inventories.

These are engineering checks of execution and validation, not measurements of placement quality.

Commands. Regenerate the tables and figures from the frozen records:

```
.venv/bin/python scripts/freeze_publication_evidence.py
.venv/bin/python scripts/render_publication_measurements.py
.venv/bin/python scripts/render_publication_curvature.py
.venv/bin/python scripts/render_publication_pilot.py
.venv/bin/python scripts/render_publication_pilot.py --profile repair
.venv/bin/python scripts/render_publication_integration.py
.venv/bin/python scripts/render_publication_devrecords.py
.venv/bin/python scripts/render_publication_v1.py
```

Build and check the report:

```
make -C writeup/latex
.venv/bin/python scripts/check_publication_paper.py
```

Run the focused tests:

```
.venv/bin/python -m pytest test/test_submission_safety.py \
  test/test_thinkorplace_package.py test/test_postpilot_repairs.py \
  test/test_sdf_topology.py test/test_publication_runner.py
```

Run a study from a configuration under `writeup/study/`, for example the pilot:

```
.venv/bin/python scripts/run_publication_study.py \
  --config writeup/study/pilot.json
```

The reported reproduction (Section 4.2) ran from commit `d9f2577` with the runtime archive `corrected-runtime-cdd01ecc7c98.tar.gz`, writing its records to `writeup/runs/ranked-reproduction-20261001-r3/`; a repeat should write to a new directory. It was run, and is rendered, with:

```
.venv/bin/python scripts/run_publication_study.py \
  --config writeup/study/ranked_reproduction.json \
  --output writeup/runs/ranked-reproduction-20261001-r3
.venv/bin/python scripts/render_publication_reproduction.py \
  --run writeup/runs/ranked-reproduction-20261001-r3
```

The renderer writes `reproduction_ranked.json`, the summary macros and Table 17; it withholds the suite mean, and exits nonzero, unless all 17 designs have a valid, usable, study-eligible result. Benchmark inputs and the evaluator come from the pinned submodule; every run records their hashes.