LEARNING WARM STARTS FOR THE SIMPLEX METHOD FROM MEASURED SOLVER COST

Anonymous authors
Paper under double-blind review

ABSTRACT

Learned initial bases for the simplex method predict which variables are basic at optimality, before handing the resulting basis to an off-the-shelf solver. Prior work has shown that while solutions built from a well-tuned GNN-based architecture can successfully reduce iteration-count, the time they save may fall well short of the iterations they save. Two effects explain the gap: the iterations removed by a good start tend to be cheap, and closeness to an optimal basis does not rank starts by cost. We prove that two starts with equal overlap with the optimal basis and equal pivot counts can differ in linear-algebra work by a factor that grows with the square root of the number of rows, and across the starts of one LP, the pivot count explains 71% of the variance of log solve time while the CPU instruction count explains 98.5%. We show that on 25 classes of linear programs, a strong supervised basis predictor removes two thirds of the iterations of the HiGHS dual simplex but only a third of its time. We introduce, COSTSTART, which uses supervised prediction for the basis and trains a small policy on measured instruction counts to choose, per LP, how many predicted columns to use, which algorithm and pricing rule to run, or no warm start. On unseen LPs of the training classes, including LPs four times the training size, COSTSTART roughly halves the solve time of HiGHS and is 1.4 to 2.2 times as fast as two published methods retrained on the same data.

1 INTRODUCTION

The simplex method remains the algorithm of choice for solving linear programs (LPs) in practice, particularly when a basic optimal solution is needed, such as in integer optimization or sensitivity analysis, or when many similar LPs from one application must be solved. It iteratively pivots between neighboring bases until it reaches an optimal one, so a good starting basis that is near optimal can significantly shorten the path. Solvers have long built starting bases with crash heuristics (Bixby, 1992; Maros, 2003), and recent work has had success learning them using a graph neural network (GNN) that scores each column for membership in the optimal basis plus a decoder to turn the scores into a valid basis, with large reductions in the number of simplex iterations required (Fan et al., 2023; Lu & Yan, 2025).

For research on the simplex method, the number of iterations (pivots) is a natural measure as it does not depend on hardware or implementation, it is what the theory of the method studies, and a start close to an optimal basis should need few iterations. In practice, however, users pay for solve time, and the work of an iteration is far from constant. It depends on the sparsity of the basis and of the vectors each iteration computes, numerical stability of the basis factorizations, and other subtle factors, and varies by orders of magnitude within one solve (Hall & McKinnon, 2005). The published results show the difference: in the HiGHS experiments of Fan et al. (2023), the learned start takes $0.25\times$ the iterations of the default start but $0.41\times$ its time, Lu & Yan (2025) report $0.60\times$ and $0.73\times$ and note that "a closer initial basis does not always result in greater acceleration" (Appendix C). We therefore minimize "charged" time, the overall solve time with every overhead of the learned start included, and treat the iteration count as one of its factors.

On our data, the difference is large (Figure 1a). On unseen LPs from 25 problem classes (with 134 variants), the strongest supervised predictor we could train removes about two-thirds of the iterations of the HiGHS dual simplex but only about a third of its time, and two published methods save little or no time. The difference has two parts. First, the iterations that a good start removes are often the computationally cheap ones. Second, closeness to an optimal basis, which predictors are trained for, does not rank the candidate starts of an LP by cost. We prove that two starts that contain the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

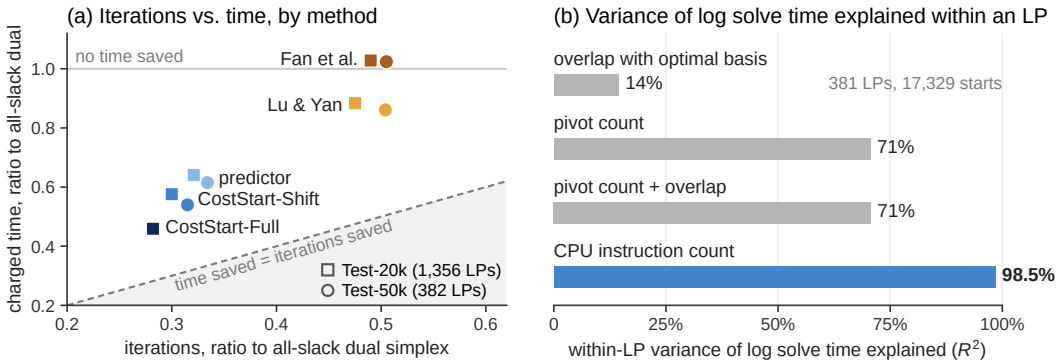

Figure 1: Iteration savings overstate time savings. (a) Iterations and charged time of learned warm starts, both relative to the HiGHS dual simplex from the all-slack basis (class-balanced geometric means over the unseen test sets of §5; the variants of CostStart are defined in §4.2). Every method lies far above the diagonal on which time saved would equal iterations saved. (b) Share of the variance of log solve time explained by each quantity across the starts of one LP (381 LPs, 17,329 distinct starts); the instruction count is a nearly noise-free measure of the solver's work and is CostStart's training label.

same fraction of an optimal basis (their overlap) and need the same number of pivots can differ in linear-algebra work by a factor that grows without bound, and on real LPs across different initial bases, the overlap explains just 14% of the variance in log solve time, and even the pivot count only explains 71% (Figure 1b).

These findings suggest that we need a method that can optimize for solver time, not just overlap between the predicted basis and an optimal basis. Supervised prediction remains the best way we found to learn which columns belong in a start, but the decisions about how to use it should be trained on the solver's measured cost and should include decisions that change the cost of each iteration. Our method, CostStart, adds a small decision policy that chooses, per LP, how far to shift the predicted scores (and so how many predicted columns enter the start), whether to run dual or primal simplex or leave the LP to the solver, and which pricing rule to use. It is trained on the CPU instruction counts of every action on every training LP, which is a proxy for solver time with less noise, with a loss equal to the exact expected cost of its choice. Its variants CostStart-Shift (shift only), CostStart-Alg (adds the algorithm) and CostStart-Full (adds the pricing rule) have growing action sets.

A warm start is useful in practice only if it can be trusted on the LPs it will meet. Where the published methods train one model per problem family and test on the same generator, we train one model for over 130 problem variants across 25 categories and test it on instances generated after the system was frozen, on LPs up to four times the training size, on held-out variants, and on unseen classes, with guardrails that limit losses on unfamiliar and badly scaled LPs.

Contributions.

- **Analysis.** We explain why iteration savings overstate time savings for advanced simplex warm start methods: the removed iterations are cheap, and neither overlap nor pivot count ranks the starts of an LP by cost (§3).
- **Method.** CostStart pairs a supervised basis predictor with a decision policy trained on measured instruction counts, whose actions change the cost of each iteration as well as their number (§4).
- **Evidence.** On fresh test sets, CostStart roughly halves the charged time of HiGHS at the training size and on LPs up to four times larger and is 1.4 to 2.2 times as fast as two published methods; trained on iteration counts instead, its policy needs 1.4 times the instructions on short solves (§6).
- **Deployment.** Guardrails limit losses on unfamiliar LPs, and 25 labeled LPs or an online bandit recover part of the loss on new classes (§6.3).
- Data. We publish a large-scale, highly realistic LP instance generator library that can be used for training and experimentation at `https://anonymous.4open.science/r/SyntheticLPs-674F/`.
- Code. Our full code and trained models are available at `https://anonymous.4open.science/r/CostStart-9464/`, providing a foundation for future work on this problem.

2 RELATED WORK

Learned initial bases for the Simplex method The canonical work for the simplex initial basis prediction task (Fan et al., 2023) trains a GNN on constraint-variable bipartite graph LP representations (Gasse et al., 2019) to predict each structural and slack variable's status at optimality: basic, at lower bound, or at upper bound. The GNN is trained with inverse-frequency-weighted cross-entropy and the predicted basis is composed of the m highest-scoring basic-status columns, repaired via the crash algorithm (Bixby, 1992) in the frequent event that the resulting matrix is singular. Recent work (**?**) improves the supervision on the same training pipeline via a duality-inspired tripartite graph that carries objective coefficients and bounds as edges to a global node, a loss applied only to the mistakes the exact-cardinality decode makes, multi-level labels sampled from feasible bases along the solver's own path, and label preprocessing for equality rows. Both works serve as are our reference points, and we rebuild and retrain both on identical data and report the reproduction.

Crash bases and measures for warm starts Crash bases built from the structure of the constraint matrix are classical (Bixby, 1992; Gould & Reid, 1989; Maros, 2003), as is the observation that they can cut iterations without cutting time (Koberstein, 2005) due to the pivot computational complexity of pivoting on dense vectors (Hall & McKinnon, 2005) and pricing rule complexity. Both Fan et al. (2023) and Lu & Yan (2025) report iterations and time relative to the solver started from its own default. (**?**) observe that fewer pivots does not always equate to faster solve time, and this pivot-based performance comparison has recently been shown to be unreliable elsewhere. Suri et al. (2026) find that reported 30–46% iteration reductions for learned interior-point warm starts rest on benchmarking against a flat start rather than the solver's actual default, and that against the corrected baseline no primal-only method reduces iterations at all. Taheri & Molzahn (2026) independently report that most partial warm starts increase solve time or reduce reliability for the same class of problems. We explore this finding in more depth for the simplex case and supply a mechanism. The learning-augmented algorithms literature asks what a prediction buys in the worst case. Sakaue & Oki (2022) give a discrete-convex-analysis framework whose running times depend on the ℓ_∞ distance from the prediction to an optimum, covering weighted matroid intersection among other problems, later extended to M-convex minimization (Oki & Sakaue, 2023). We prove analytically the range of computational complexity for warm-starts with the same pivot-distance to the optimal solution for a example class of problems.

3 ITERATION SAVINGS OVERSTATE TIME SAVINGS

We solve LPs $\min\{c^\top x : Ax = b,\ \ell \le x \le u\}$ with m rows and n structural columns with HiGHS 1.13.1 (Huangfu & Hall, 2018), on one thread with presolve off. A basis is a set of m columns of $[A\ I]$ with a nonsingular submatrix; the columns of I are slacks, and the all-slack basis is I itself. A warm start installs a basis B_0 before the solve, and its overlap is $|B_0 \cap B^\star|/m$ for an optimal basis $B^\star$. Our main comparator, native HiGHS, receives no basis and uses HiGHS' "choose" strategy: primal simplex if the all-slack basis is primal feasible, dual simplex otherwise (HiGHS' default, always dual, is slower on our data). Learned warm starts make B_0 resemble $B^\star$ and are evaluated through the number of iterations N, but the user also pays for producing and installing the start and for the work of each iteration. To compare a method a with a reference r on one LP, we split the log time ratio into iterations, time per iteration and fixed overhead,

$$\log \frac{T_a}{T_r} \;=\; \log \frac{N_a}{N_r} \;+\; \log \frac{\bar{w}_a}{\bar{w}_r} \;+\; \Big(\log \frac{T_a}{T_a^{\text{run}}} - \log \frac{T_r}{T_r^{\text{run}}}\Big), \tag{1}$$

where T is the charged time, T^{run} the time of the solver's own run and $\bar{w} = T^{\text{run}}/N$ its mean time per iteration.

3.1 OVERLAP AND PIVOT COUNT DO NOT DETERMINE WORK

An optimal basis would make the solve nearly free, but for other starts overlap and pivot count do not determine the work, even for a fixed pivot rule. Take classical dual steepest edge (DSE), which picks the leaving row p maximizing δ_p^2/γ_p, where δ_p is the row's primal infeasibility and $\gamma_p = \|B^{-\top} e_p\|_2^2$. Each iteration solves for $\rho_p = B^{-\top} e_p$, $\alpha_q = B^{-1} a_q$ and, for DSE, $\tau = B^{-1}\rho_p$. We call the total number of nonzeros of these vectors along a path its solve-vector work K. K depends only on

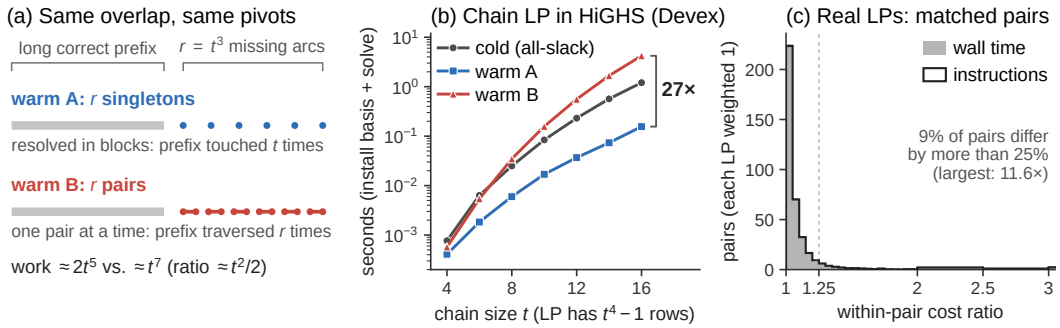

Figure 2: Equal overlap and pivot count do not imply equal work. (a) The warm starts of Theorem 1: DSE resolves the singletons of warm A in blocks and the pairs of warm B one at a time. (b) The chain LP in HiGHS (presolve and scaling off, Devex): warm B takes 27 times as long as warm A. (c) Real LPs: pairs of starts of one LP with overlap within 0.005 and pivots within 2% differ in time by more than 25% in 9% of cases (class-balanced; the histogram gives each LP equal weight).

the bases on the path and bounds the work of any implementation from below; on the LP below, output-sensitive solves and heap-based row selection achieve $\Theta(M + K)$ up to logarithmic terms (Appendix A).

Theorem 1 (Equal overlap and pivot count, unbounded work ratio)**.** For every even $t \geq 4$ let $M = t^4$ and $r = t^3$, and consider the minimum-cost-flow LP on a path with return arcs $\min \sum_{i=1}^{M} u_i$ subject to $u_j - u_{j+1} - v_j = b_j$ $(1 \leq j < M)$, $u, v \geq 0$, with an explicit rational $b \in [0.97, 1.03]^{M-1}$. It has a unique nondegenerate optimal basis, and three dual-feasible starts satisfy, under exact DSE,

Start	Overlap with $B^\star$	Pivots	Work K
Cold (all-slack: every v_j basic)	0	$M - 1$	$\sim \frac{3}{2}t^6$
Warm A (last r chain arcs missing: singletons)	$1 - r/(M-1)$	r	$\sim 2t^5$
Warm B (every second of the last $2r$ arcs missing: pairs)	$1 - r/(M-1)$	r	$\sim t^7$

Every choice on the three paths is unique, every basis factorizes without fill, and the two warm bases have equal nonzero counts after every pivot. Hence $K_B/K_A \sim t^2/2 = \Theta(\sqrt{M})$, and $K_B/K_{\text{cold}} = \Theta(M^{1/4})$.

Corollary 1 (The proxy-optimal start can be the costlier one)**.** With $r - 1$ missing arcs, warm B has strictly higher overlap and one fewer pivot than warm A, and still $\sim t^2/2$ times its work.

Only the arrangement of the missing columns differs (Figure 2a). Most of warm B's work is the solve with the entering column, which spans the long correct prefix each time one of its isolated pairs is attached, whereas warm A's singletons are first assembled into blocks that touch the prefix rarely; every dual simplex iteration forms this vector, so the separation is a property of the path. Work on algorithms with predictions bounds the running time of warm-started algorithms by the distance of the prediction from an optimal solution (Dinitz et al., 2021; Sakaue & Oki, 2022); for the simplex method, the theorem rules out such a bound in terms of overlap, even together with the pivot count.

HiGHS with Devex pricing and with scaling and cost perturbation off takes the same numbers of pivots (Figure 2b): at 65,535 rows it takes 1.2 s from the cold start, 0.16 s from warm A and 4.2 s from warm B, which needs a sixteenth of the cold pivots. Under its default edge weights, a non-slack start that is not near optimal first computes exact DSE weights with one BTRAN per row, $\Theta(M^2)$ work here, which hides the difference (on our real LPs, a median 0.4% of the solve).

3.2 On realistic LPs

To investigate this empirically, we solved 381 LPs of Dev-20k (25 categories, 134 variants), each from up to 53 starts: 42 from our decoder, ten high-overlap starts and the all-slack basis (Appendix B).

Overlap and pivots rank starts poorly; instructions rank them almost perfectly. Within an LP, the pivot count explains 71% of the variance of log time, adding overlap explains nothing more, and

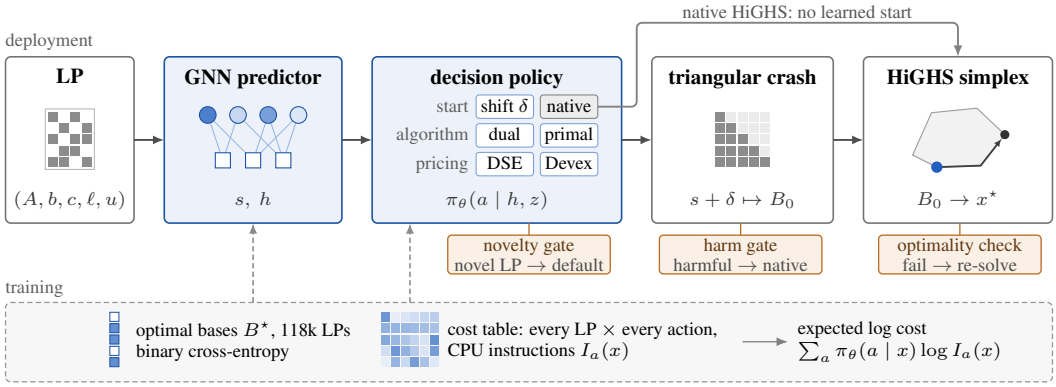

Figure 3: COSTSTART at deployment (top) and in training (bottom). Blue boxes are learned and orange tags are guardrails, all charged in the timings; the solver is unmodified. From the predictor's embedding h and score statistics z, the policy picks one of up to 20 actions: a shift δ of the scores s with an algorithm and pricing rule, or native HiGHS. Training uses optimal bases for the predictor and instruction counts of every action for the policy.

the instruction count explains 98.5% (Figure 1b; Table 4). Among the decoder's 42 starts, the one with the highest overlap costs $1.28\times$ the time of the fastest and the one with the fewest pivots $1.09\times$; and when one start has both higher overlap and fewer pivots than another, it is still more than 5% costlier in 12% of such pairs, the empirical counterpart of Corollary 1.

The removed iterations are the cheap ones. An iteration from a learned start costs about 1.3 to 1.9 times one from the all-slack start. Traces of single iterations show why: on the dual simplex path from the all-slack basis, the early iterations that the predictor's start skips cost $0.44\times$ [0.28, 0.66] as much as the later ones it keeps, and the learned path's iterations cost about as much as those (17 development LPs; Appendix B). Kernel statistics point, as in the theorem, to denser solve vectors, and no static property of the start predicts this cost.

These findings shape the method. Starts of similar closeness differ widely in cost, so the label should be measured cost; one random basis swap changes the cost by 4% at the median and by more than 30% one time in ten, so cost should enter through a choice among a few well-separated options rather than through gradients on every score; and a start near $B^\star$ trades fewer iterations for costlier ones, so the options should include ways to make each iteration cheaper.

4 COSTSTART: SUPERVISED PREDICTION, COST-TRAINED DECISIONS

An overview of the COSTSTART system is shown in Figure 3, which includes a supervised predictor (§4.1), a decision policy trained on measured cost (§4.2), and three guardrails (§4.3). The pipeline adds 30 to 50 ms to a 20,000-variable LP, all of it charged.

4.1 SUPERVISED PREDICTOR AND DECODER

Predictor. We encode the LP as a graph with a node per structural column, per slack column and per constraint, and an edge for every nonzero of A (Gasse et al., 2019). An 8-layer GATv2 network (Brody et al., 2022) with 1.8M parameters scores every column of $[A\ I]$ for membership in the optimal basis that HiGHS returns, trained with binary cross-entropy first on TRAIN-SMALL and then on 3,457 LPs at the 20k size. It is a scaled-up version of the predictor of Fan et al. (2023) (Appendix D); its forward pass also yields an embedding h of the LP.

Decoder. A triangular crash builds a basis greedily from the scores (Algorithm 1 in Appendix D). It repeatedly makes the structural column j basic in the row i with the largest positive gain $(s_j - s_{n+i})/\deg(j)^{1/2}$, where s_{n+i} is the score of row i's slack and $\deg(j)$ the number of entries of column j, and removes every candidate that would break triangularity; rows without a structural column keep their slack. The result is always nonsingular and needs no repair. Adding a constant δ to every structural score admits more structural columns for $\delta > 0$ and fewer for $\delta < 0$.

4.2 DECISION POLICY

A predictor of $B^\star$ is trained and judged only by closeness to $B^\star$; nothing in its objective sees what each iteration costs. The decision policy chooses, per LP, among a few well-separated actions that also change the setup and the work per iteration, and it is trained on the measured cost of every action. We study three nested action sets of 9, 11 and 20 actions.

Shifts ($\mathcal{A}_1$). Nine shifts $\delta \in \{0, \pm 0.5, \pm 1, \pm 2, \pm 3\}$. The policy over $\mathcal{A}_1$, with the harm gate below as a fallback to native HiGHS, is COSTSTART-SHIFT, which we froze, with its evaluation plan written down, before any test data were generated.

Algorithm ($\mathcal{A}_2$). The shifts plus native HiGHS, for LPs on which no learned start pays off, and the unshifted start continued with primal simplex, for starts that are nearly primal feasible. The policy over $\mathcal{A}_2$ is COSTSTART-ALG.

Pricing ($\mathcal{A}_3$). $\mathcal{A}_2$ plus the nine shifts with Devex pricing (Harris, 1973) forced from the first iteration. HiGHS' default edge weights use DSE and switch to Devex by themselves mainly for a near-optimal supplied basis or after at least $0.1(m+n)$ iterations of costly DSE solves. Devex drops DSE's extra solve τ and its exact weights, trading more iterations for cheaper ones, and which effect dominates varies across classes; of 13 alternative solver options we probed, it gave the largest gain (Appendix E). The policy over $\mathcal{A}_3$ is COSTSTART-FULL; COSTSTART-ALG and COSTSTART-FULL were designed on development data after COSTSTART-SHIFT was frozen.

Cost signal. The label of action a on LP x is $I_a(x)$, the CPU instructions HiGHS executes to install the start and solve, from a hardware counter. Repeated solves agree to a median of 5×10^{-5} in log instructions, so unlike wall time the label is reproducible and unaffected by machine load; the table for $\mathcal{A}_3$ costs about 16 cold solves per training LP.

Policy and loss. The policy $\pi_\theta(a \mid x) = \mathrm{softmax}(f_\theta(h(x), z(x)))$ is a two-layer MLP on h and 12 statistics z of the scores and the LP's size, trained on the exact expected log cost relative to the unshifted start,

$$\mathcal{L}(\theta) \;=\; \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi_\theta(a \mid x)\, \mathrm{clip}\big(\log I_a(x) - \log I_0(x),\, -3,\, 3\big), \tag{2}$$

whose exponential is, up to the clip, the geometric-mean instruction ratio, the training counterpart of the ratio we report (Appendix A.7). It has no sampling variance, and because the label is total cost, it prefers more iterations whenever each becomes cheap enough. We deploy the argmax of an ensemble; the policy runs in under 2 ms.

Why not train the predictor on the cost? Every attempt we made, including policy gradients through a stochastic crash, perturbed optimizers and end-to-end training, failed to beat continued supervision at matched budget per-LP cost gradients point in unrelated directions (cosine near 0 between half-batches, against 0.51 for the supervised loss). We therefore freeze the predictor.

4.3 GUARDRAILS

Novelty gate. Outside the training classes the policy confidently chooses harmful actions. If the input $[h, z]$ is farther from every training LP than the 0.99 quantile of in-distribution nearest-neighbor distances, the policy plays the default action (unshifted start, dual simplex, HiGHS' default edge weights).

Harm gate. A gradient-boosted regressor predicts, from cheap features of the LP and the decoded start and without class labels, whether the learned start will save at least 12% over native HiGHS; if not, native HiGHS runs. With COSTSTART-ALG and COSTSTART-FULL it is evaluated on every LP but is needed only where the novelty gate takes the decision away from the policy.

Optimality check. On one badly scaled class, whose optimal duals are of order 10^{-7}, a learned start can end at a basis that HiGHS accepts as optimal because each of its many wrong-signed reduced costs is below the tolerance 10^{-7}, with an objective up to 20% too high. If HiGHS' sum of dual infeasibilities, which includes such values, exceeds $10^{-6}(1 + |f|)$ for objective f, we continue from the returned basis with tolerances 10^{-9}; on development data this flagged all 12 wrong solves and none of 2,358 correct ones (Appendix F.1).

5 EXPERIMENTAL SETUP

Data. All LPs come from SyntheticLPs, a library of parametric generators for 45 operations-research problem classes, each with several variants and a primal and a dual form (see Appendix D); every LP is presolved once and stored, and set names give the typical number of variables. The predictor is trained on TRAIN-SMALL (114,874 LPs, 50 to 2,000 variables) and on 3,457 LPs at the 20k size; the policies and the harm gate are trained on TRAIN-20K (9,511 LPs, 15,000 to 25,000 variables), and DEV-20K and DEV-50K (40,000 to 60,000) are used for development. Sizes are those requested from the generator; after presolve, the median is 18,000 variables in TRAIN-20K and 88,000 in TEST-80K. After an audit found near-duplicates across our original splits, we froze COSTSTART-SHIFT and generated the test sets from new seeds, admitting an LP only if no training or development LP shares its content hash, generator call or seed (Table 13).

Test sets. TEST-20K has 1,356 LPs with 15,000 to 25,000 variables from 25 training classes; TEST-50K (382 LPs, 40,000 to 60,000 variables) and TEST-80K (81 LPs, 60,000 to 100,000) hold larger instances of the same classes. TEST-VARIANT is the subset of TEST-20K (296 LPs) from ten variants and six dual forms that a second, separately trained pipeline never sees. TEST-NEWCLASS (234 LPs) comes from six classes absent from training, and TEST-ADAPT from an earlier split of five of them (125 LPs for adaptation, 616 for evaluation). An LP was admitted if HiGHS' dual simplex needed at least 2,000 iterations or 1 s in a separate run at generation time, and LPs are grouped by that run's time into windows W1 (up to 10 s), W2 (10 to 120 s) and W3 (over 120 s), with time limits of 120, 600 and 3,600 s.

Protocol. Every method solves each LP in a fresh HiGHS instance on one pinned core; TEST-20K was timed twice and averaged. Charged time includes features, inference, the policy, decoding, installing the basis, the solve, gates, checks and fallbacks. The predictor runs on one GPU and is charged at its wall time (on CPU cores alone it would add about 30% to COSTSTART-SHIFT on solves under 10 s; Appendix I). Failures (time limits, errors, objectives off by more than 10^{-5} relative to a reference) count at twice the time limit. Iterations are reported relative to the all-slack dual simplex, the same algorithm as the warm starts, and time relative to native HiGHS. The main statistic is the class-balanced geometric mean of per-LP ratios (log ratios averaged within each class, then across classes), with 95% bootstrap intervals over classes and then generator instances.

Methods. Baselines are native HiGHS, the all-slack dual simplex, stock HiGHS (all default options, presolve on, applied to the stored presolved LP) and a classical crash (inverse-degree scores through our decoder). We reimplemented Fan et al. (2023) and Lu & Yan (2025) (0.2M and 1.2M parameters), trained them on our data at their original budget and at ours, chose their checkpoints by their own loss (ours by solve time on development data), and decode them with our crash, which suits them better than their own decoders here (Appendix K). COSTSTART-ALG and COSTSTART-FULL were designed after a pilot of TEST-20K and the TEST-NEWCLASS panel had run, and frozen before the full TEST-20K panel started (Appendix H).

6 RESULTS

6.1 UNSEEN LPS OF THE TRAINING CLASSES

Every learned start removes half or more of the iterations (Table 1), but the published methods end up within 10% of native HiGHS in time on TEST-20K. Relative to native HiGHS, our predictor alone takes $0.68\times$ the time, COSTSTART-SHIFT $0.61\times$ and COSTSTART-FULL $0.49\times$, faster on 24 of the 25 classes; on the larger LPs of TEST-50K and TEST-80K, COSTSTART-SHIFT still takes about half the time. The intervals of COSTSTART-SHIFT, and of the larger policies with the harm gate, exclude parity on all three test sets. We regard COSTSTART-FULL with both gates and the check as the deployed configuration: at the training size the gate costs 6%, but larger LPs need it (§6.3).

Most of our advantage over the published methods at the level of the start comes from the predictor; more training, another seed or their own decoders do not close the gap (Appendix K). On top of the predictor, COSTSTART-FULL lowers the time by a further 28%, which makes it 1.9 to 2.2 times faster than the published methods. COSTSTART-SHIFT lowers it by 10%, but that gain comes from its harm gate: with the gate present, the shift policy adds nothing on the test sets, because both act on the same LPs (Appendix D.1), and refit without the test LP's class the gate loses most of its value. A lookup

Table 1: Main results on unseen LPs of the training classes: every learned start removes half or more of the iterations, but only our systems save substantial time. Class-balanced geometric means (below 1 is better) of iterations relative to the all-slack dual simplex and of charged time relative to native HiGHS, with 95% intervals and classes won/lost. The class-blind gate is refit without the test LP's class; the lookup plays each class's cheapest training action. TEST-50K and TEST-80K predate the optimality check and the larger policies (see Table 2a). More rows: Appendix I.

	TEST-20K (1,356 LPs)			TEST-50K (382 LPs)			TEST-80K (81)
Method	iter.	time [95% CI]	W/L	iter.	time [95% CI]	W/L	time
all-slack dual simplex	1.00	1.07 [0.87, 1.30]	12/13	1.00	0.99 [0.76, 1.30]	12/13	0.96
stock HiGHS, presolve on	0.85	1.10 [1.06, 1.16]	3/22	–	–	–	–
classical crash	1.22	1.76 [1.19, 2.60]	9/16	–	–	–	–
Fan et al.	0.49	1.10 [0.89, 1.35]	15/10	0.50	1.02 [0.78, 1.37]	15/10	0.96
Lu & Yan	0.47	0.94 [0.70, 1.22]	14/11	0.50	0.85 [0.56, 1.23]	19/6	0.72
predictor	0.32	0.68 [0.50, 0.91]	20/5	0.33	0.61 [0.39, 0.89]	19/6	0.60
COSTSTART-SHIFT	0.30	0.61 [0.46, 0.76]	23/2	0.32	0.53 [0.34, 0.72]	23/2	0.52
with class-blind harm gate	0.32	0.68 [0.52, 0.85]	18/7	–	0.58 [0.40, 0.79]	18/7	0.58
COSTSTART-ALG	0.25	0.51 [0.38, 0.65]	23/2		see Table 2		
COSTSTART-FULL	0.28	0.49 [0.37, 0.63]	24/1		see Table 2		
with harm gate (deployed)	0.29	0.52 [0.39, 0.65]	23/2		see Table 2		
per-class lookup (uses class labels)	–	0.54 [0.41, 0.67]	–		–		

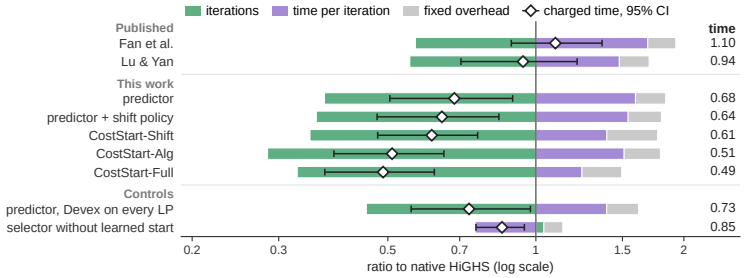

Figure 4: Every learned start trades fewer iterations for costlier ones. Charged time on TEST-20K relative to native HiGHS, split by Eq. (1) into iterations, time per iteration and fixed overhead (class-balanced geometric means). On the log axis, segments left of 1 lower the time, segments right of 1 raise it, and the diamond (95% interval) is their sum. "Predictor + shift policy" is COSTSTART-SHIFT without the harm gate.

that plays, for each class, the action with the lowest training cost is a strong control, although it needs class labels our methods never see: it beats COSTSTART-SHIFT, which only chooses between the start and native HiGHS (a two-action lookup ties it), and COSTSTART-FULL is 9% faster than it overall (not significant) and $0.76\times$ [0.64, 0.87] its time on solves of 10 to 120 s.

6.2 FEWER ITERATIONS AND CHEAPER ITERATIONS

Figure 4 applies the decomposition (1) to TEST-20K. Every learned start cuts the iterations of native HiGHS by at least 40% but makes each iteration 1.5 to 1.7 times as costly for the published methods and our predictor; the published methods remove too few iterations to compensate. COSTSTART-ALG gains mainly through fewer iterations and COSTSTART-FULL through cheaper ones: it takes 15% more iterations than COSTSTART-ALG, each 18% cheaper, a net difference that is not significant (Appendix E). Devex on every LP, or a selector trained with the same loss but without the learned start, is slower than the predictor alone, so the pricing choice must be made per LP and with the start.

Which cost to train on. Trained on iteration counts with the same recipe, the policy of COSTSTART-FULL removes 15% more pivots on held-out training LPs under 10 s than the instruction-trained one but needs $1.40\times$ [1.18, 1.73] its instructions (a tie on longer solves), because it sends LPs to primal simplex (36% against 8%) instead of Devex (12% against 51%), and its iterations cost 1.65 times as much. Shift policies show the same pattern in charged time (Appendix I.1), and on training LPs the shift closest to the known $B^\star$ gains 4%, a cost-trained policy 7%. Two more pipeline seeds reproduce COSTSTART-SHIFT on TEST-20K to within 1% (Appendix I.2).

Table 2: COSTSTART keeps most of its gain on larger LPs and held-out variants; new classes need adaptation. Charged time relative to native HiGHS (class-balanced). (a) A later pass over solves under 120 s with the optimality check inline (failures in parentheses); Table 1 also includes longer solves. (b) With 95% intervals; the last column is relative to the reference named in each block.

(a) Larger LPs	TEST-50K (309)	TEST-80K (63)
all-slack dual simplex	0.94	0.90
predictor	0.65 (1)	0.83 (1)
Devex on every LP	0.66	0.78 (1)
COSTSTART-SHIFT	0.58	0.67
COSTSTART-ALG	0.60 (2)	0.83 (1)
+ harm gate	0.55	0.66
COSTSTART-FULL	0.56 (1)	0.81 (1)
+ harm gate	0.54	0.65

(b) Unseen types	vs. native	vs. reference
TEST-VARIANT, 10 variants and 6 dual forms (ref.: deployed predictor)		
predictor trained without these	0.71 [0.33, 1.41]	1.33 [1.13, 1.64]
+ own shift policy, deployed gate	0.56 [0.30, 0.86]	1.05 [0.67, 1.50]
TEST-NEWCLASS, 6 unseen classes (ref.: predictor)		
predictor	1.42 [0.95, 2.30]	1.00
COSTSTART-SHIFT	1.21 [0.97, 1.62]	0.85 [0.62, 1.08]
classical crash	0.67 [0.30, 1.18]	–
TEST-ADAPT, 5 unseen classes (ref.: predictor, where the policy abstains)		
adapted, 25 labeled LPs per class	1.04 [0.73, 1.48]	0.91 [0.84, 0.98]
online bandit, no labels	0.93 [0.79, 1.09]	0.82 [0.64, 0.97]

6.3 GENERALIZATION AND DEPLOYMENT

Larger LPs. Training LPs were generated at up to 25,000 variables, yet the predictor keeps most of its advantage at up to 100,000 (Table 1). The decision policy transfers only in part: the novelty gate sends most larger LPs to the default action, where the policy cannot choose native HiGHS, so the harm gate becomes necessary. With both gates and the optimality check, which removes the wrong optima COSTSTART-SHIFT returned on TEST-50K in the first pass, COSTSTART-FULL and COSTSTART-ALG are the fastest methods and have no failures (Table 2a, which covers solves under 120 s, where savings are smaller).

New variants. On TEST-VARIANT, a predictor retrained without these ten variants and six dual forms is 33% slower than the deployed one; its own shift policy adds nothing there, and the deployed harm gate, which did see these types, recovers most of the loss. On development data, held-out dual forms keep most of the predictor's gain and held-out variants about a quarter (Appendix J).

New classes. On six classes that no model saw, every learned start of ours is slower than native HiGHS, while the classical crash is faster (Table 2b). The novelty gate flags every such LP, but its fallback, chosen by leave-one-class-out validation, is wrong for new classes. The supported scope is therefore new instances, sizes and variants of the training classes; for new classes we tested adaptation from the solver's own cost. On TEST-ADAPT, where the unadapted policy falls back to the predictor's start, 25 labeled LPs per class bring its time to $0.91\times$ [0.84, 0.98]. Without labels, a contextual bandit over shifts and solver strategies, learning from the cost of its own solves, ends at $0.82\times$ [0.64, 0.97] the predictor's time and $0.93\times$ that of native HiGHS over streams of 64 to 149 LPs per class, but not on the shorter streams of TEST-NEWCLASS (Appendix J.1).

The public benchmark of Lu & Yan. Under their protocol (one model per dataset; four datasets, 80 test LPs), we fine-tune our predictor per dataset and choose by cross-validation between a trained policy and the best single action, over actions including their top-m decoder (Appendix K.1). It takes $0.72\times$ [0.68, 0.75] (bootstrap over LPs) the time of HiGHS' dual simplex, against $1.03\times$ for our reimplementation of Lu & Yan and $0.92\times$ for Fan et al. (Lu & Yan report $0.73\times$ on their split); with 19 to 70 training LPs per dataset, the best single action captures most of the gain.

7 LIMITATIONS AND CONCLUSION

Limitations. The learned components do not transfer to classes absent from training, where adaptation needs labels or a stream of about a hundred LPs per class. COSTSTART-ALG and COSTSTART-FULL were designed after COSTSTART-SHIFT was frozen. All main results use one generator library and one solver, and test LPs were admitted by the dual simplex's solve time. The theorem concerns solve-vector work under one pivot rule, and a per-iteration cost linear in the rows bounds its ratio (Appendix L).

Conclusion. Learned bases save a smaller share of time than of iterations, because the iterations they remove are cheap and closeness to the optimum does not rank starts by cost. COSTSTART keeps supervised prediction for the basis, trains the decisions about its use on measured cost, and roughly halves the time of HiGHS on unseen LPs of the training classes, including larger ones.

REPRODUCIBILITY STATEMENT

Code to reproduce all experiments and the trained models are available at https://anonymous.4open.science/r/CostStart-9464/. All LPs are generated by an open-source generator library available at https://anonymous.4open.science/r/SyntheticLPs-674F/. The details of the datasets, hyperparamters, seeds, and more are described in Appendix H. The solver is HiGHS 1.13.1 with the options in Table 12, which also lists every setting of the predictor, the policies and the gates. The decoder is given as Algorithm 1 and the deployment procedure as Algorithm 2. Timing follows the protocol of §5 and Appendix H, and every time is reported beside a hardware-independent instruction count. The published baselines were re-implemented as described in Appendix K.

REFERENCES

Robert E. Bixby. Implementing the simplex method: The initial basis. *ORSA Journal on Computing*, 4(3):267–284, 1992. doi: 10.1287/ijoc.4.3.267.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022.

Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Zhenan Fan, Xinglu Wang, Oleksandr Yakovenko, Abdullah Ali Sivas, Owen Ren, Yong Zhang, and Zirui Zhou. Smart initial basis selection for linear programs. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 9650–9664. PMLR, 2023. URL https://proceedings.mlr.press/v202/fan23d.html.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, pp. 15554–15566, 2019.

Nicholas I. M. Gould and John K. Reid. New crash procedures for large systems of linear constraints. *Mathematical Programming*, 45(1–3):475–501, 1989. doi: 10.1007/BF01589115.

J. A. J. Hall and K. I. M. McKinnon. Hyper-sparsity in the revised simplex method and how to exploit it. *Computational Optimization and Applications*, 32(3):259–283, 2005. doi: 10.1007/s10589-005-4802-0.

Paula M. J. Harris. Pivot selection methods of the Devex LP code. *Mathematical Programming*, 5(1):1–28, 1973.

Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi: 10.1007/s12532-017-0130-5.

Achim Koberstein. *The Dual Simplex Method, Techniques for a Fast and Stable Implementation*. PhD thesis, Universität Paderborn, 2005. URL https://d-nb.info/978580478/34.

Anqi Lu and Junchi Yan. Learning initial basis selection for linear programming via duality-inspired tripartite graph representation and comprehensive supervision. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 40831–40847. PMLR, 2025. URL https://proceedings.mlr.press/v267/lu25q.html.

István Maros. *Computational Techniques of the Simplex Method*, volume 61 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, 2003. ISBN 978-1-4020-7332-8.

Taihei Oki and Shinsaku Sakaue. Faster discrete convex function minimization with predictions: The M-convex case, 2023. URL https://arxiv.org/abs/2306.05865.

Shinsaku Sakaue and Taihei Oki. Discrete-convex-analysis-based framework for warm-starting algorithms with predictions. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Dhruv Suri, Helgi Hilmarsson, and Shourya Bose. WARP: A benchmark for primal-dual warm-starting of interior-point solvers. *arXiv preprint arXiv:2605.05728*, 2026.

Babak Taheri and Daniel K. Molzahn. Not all warm starts help: Benchmarking primal-dual initializations for ACOPF algorithms, 2026. URL https://arxiv.org/abs/2606.08984.

A Proof of Theorem 1 and Corollary 1

We prove Theorem 1 and Corollary 1, state the cost model under which solve-vector work bounds total work, and report how HiGHS behaves on the chain LP. Throughout, $M = t^4$ is the number of path vertices $1, \dots, M$ as in the theorem, $R = M - 1$ is the number of equality rows (the main text's m), $q = t^2$ and $r = t^3$; all arithmetic is exact. The variables v_j are explicit surplus columns $-e_j$ that play the role of slacks; nothing relies on the solver's built-in slack (logical) columns.

Solve-vector work. A dual simplex iteration with leaving row p forms the BTRAN result $\rho_p = B^{-\top} e_p$, the FTRAN result $\alpha = B^{-1} a_{\text{in}}$ of the entering column a_{in} and, under DSE pricing, the second FTRAN result $\tau = B^{-1}\rho_p$. The solve-vector work K of a path is the total number of nonzeros of these three vectors over its pivots. The vectors are determined by the basis and the pivot, not by the LU factors, so K is intrinsic to the sequence of bases, and it is a lower bound on the work of any implementation that forms them. It omits PRICE (the pivotal row $\rho_p^\top A_N$), the choice of the leaving row (CHUZR), the choice of the entering column with the bound-flipping ratio test (CHUZC and BFRT), factor updates and reinversion. On the chain LP every row has at most three nonzeros, and the arithmetic model below bounds these omitted steps by $O(K) + O(\log M)$ per pivot.

A.1 The chain LP as a minimum-cost-flow problem

Index nodes $0, \dots, R$ and use the convention $+1$ at an arc's head and -1 at its tail, deleting the row of node 0. Then u_1 is the arc $0 \to 1$, u_j ($2 \le j \le R$) the arc $j-1 \to j$, u_M the arc $R \to 0$, and v_j the arc $j \to 0$, with unit cost on every u arc and zero cost on every v arc. The LP is therefore an uncapacitated minimum-cost transshipment problem on a path with return arcs to a hub, with R rows, $2M - 1$ columns and $3R$ nonzeros, and its constraint matrix is a reduced node-arc incidence matrix, hence totally unimodular. The separation of Theorem 1 is thus $\Theta(\sqrt{R})$ in the number of rows. Two cautions apply. Total unimodularity alone does not make every selection of columns a basis or free of fill; the selections used here are rooted spanning trees, and in the stated slot order their basis matrices are upper bidiagonal, which gives both properties. The right-hand side is rational, so basic solutions need not be integral; multiplying b by $100M$ gives integer supplies with identical DSE choices and solve-vector supports. The construction is an existence result within this class and makes no claim about typical network-flow instances.

A.2 Basis family, dual feasibility and the unique optimum

Set $e_0 = e_M = 0$ in $\mathbb{R}^R$. The columns of u_j and v_j are $e_j - e_{j-1}$ and $-e_j$. For $S \subseteq \{1, \dots, R\}$, put u_j in slot j if $j \in S$ and v_j otherwise; u_M stays nonbasic. The basis matrix is $B(S) = U(S)\Sigma(S)$ with $U(S) = I - \sum_{j \in S, j \ge 2} e_{j-1} e_j^\top$ and $\Sigma_{jj} = +1$ if $j \in S$, -1 otherwise. It is upper bidiagonal with unit-magnitude diagonal and at most $2R - 1$ nonzeros, so $L = I, U = B(S)$ is an exact no-fill factorization and $|\det B(S)| = 1$.

Selected edges join consecutive vertices into components. If $[h, v]$ is a component, back-substitution gives $u_j = \sum_{s=j}^{v-1} b_s > 0$ for $j \in S$ in $[h, v]$ and $v_j = -\sum_{s=j}^{v-1} b_s < 0$ for a missing boundary j whose right component is $[j+1, v]$. Hence exactly the missing-boundary basics are primal infeasible.

Let λ solve $B(S)^\top \lambda = c_B$ with $\lambda_0 = 0$. A basic surplus sets $\lambda_j = 0$; a basic structural gives $\lambda_j = 1 + \lambda_{j-1}$, so λ_j is the length of the selected run ending at j. A missing nonbasic u_j has reduced cost $1 + \lambda_{j-1} = a$, the length of its left component; a nonbasic v_j at a selected edge has reduced cost $\lambda_j > 0$; u_M has reduced cost $1 + \lambda_R > 0$. Every basis in the family is strictly dual feasible for every positive right-hand side. The constraints imply $u_j \ge u_{j+1} + b_j$ and $u_M \ge 0$; since every u_i has positive cost the unique optimum is $u_j^\star = \sum_{s \ge j} b_s$, $u_M^\star = 0$, $v^\star = 0$, with $f^\star = \sum_j j b_j$ and the unique nondegenerate optimal basis $S^\star = \{1, \dots, R\}$. Every missing edge requires a separate basis exchange.

A.3 Legal pivots and exact solve-vector counts

Let missing edge j separate a left component of length a and a right component $[j+1, v]$ of length b. Each inverse block of $U(S)$ is upper triangular with all upper entries equal to one, and Σ only changes row signs; component $[h, v]$ corresponds to inverse indices $[\max(1, h-1), v-1]$. For the

missing row j,

$$B(S)^{-\top} e_j = -\sum_{s=j}^{v-1} e_s, \qquad \gamma_j = b, \qquad \delta_j = \sum_{s=j}^{v-1} b_s.$$

Pricing this BTRAN result against the nonbasic columns gives coefficient -1 at u_j, $+1$ at the next missing u_v (or u_M), nonnegative coefficients at surplus columns inside the block and zero elsewhere; the leaving $v_j < 0$ rises to its bound, so only negative coefficients are eligible and u_j is the unique entrant; its reduced cost is $a > 0$, the dual step is a, and the new basis is exactly $B(S \cup \{j\})$. Every insertion order is a legal improving dual path with no ratio-test ties. Column $e_j - e_{j-1}$ has one source in each adjacent inverse block, reaching row j on the right and $a - \chi$ rows on the left, where $\chi = \mathbf{1}\{h = 1\}$; the supports are disjoint. Two inverse rows have nonzero inner product exactly when they lie in the same block, since row i of a block is $\sigma_i(0, \ldots, 0, 1, \ldots, 1)$ and the products cannot cancel. Therefore

$$\operatorname{nnz}(B^{-1} a_{u_j}) = a + 1 - \chi, \qquad \operatorname{nnz}(B^{-\top} e_j) = b, \qquad \operatorname{nnz}(B^{-1} B^{-\top} e_j) = b,$$

and a pivot contributes $a + 2b + 1 - \chi$ to K. Its DSE score is $\delta_j / \sqrt{b}$, which with a unit right-hand side equals $\sqrt{b}$; the merge order is therefore forced by the rule, not chosen.

A.4 One right-hand side forces all paths strictly

For even $q \geq 4$, $M = q^2$, $\varepsilon = 1/100$ and seeds $E = \{q - 1, 2q - 1, \ldots, M - 1\}$ (all odd), define for odd i and even j

$$z_i = \begin{cases} 3 - i/M & i \in E \\ 0 & i \notin E \end{cases}, \qquad b_i = 1 + \varepsilon z_i, \qquad b_j = 1 - \varepsilon z_{j+1} + \varepsilon (M - j)/M.$$

Every entry lies in $[1 - 3\varepsilon, 1 + 3\varepsilon]$ and $b_j + b_{j+1} = 2 + \varepsilon(M - j)/M$ for even j.

Cold path. From $S_0 = \emptyset$ (the all-slack basis), exact DSE forms blocks of q consecutive vertices from left to right: within block k it inserts $kq - 1, kq - 2, \ldots, (k - 1)q + 1$ and then, for $k \geq 2$, inserts $(k - 1)q$ to attach the block. Suppose the previous blocks are complete; every remaining right component is a singleton with score b_j; seed entries exceed $1 + 2\varepsilon$, nonseeds are at most $1 + \varepsilon$, and seed values decrease with index, so $kq - 1$ uniquely wins. It creates a right component of size two, and during the cascade exactly one missing boundary has a nonsingleton right component, with score at least $(1 - 3\varepsilon)\sqrt{2} > 1 + 3\varepsilon$, the maximum of any singleton; so it wins uniquely until the cascade reaches the prefix, including at the attachment step. Induction gives the whole path and its strictness.

Warm B. Omit only $F = \{M - 2r, M - 2r + 2, \ldots, M - 2\}$: components $(M - 2r, 2, \ldots, 2)$. Every missing boundary is even with score $(2 + \varepsilon(M - j)/M)/\sqrt{2}$, strictly decreasing in j; inserting the leftmost joins the prefix to one pair and changes no other score, so all missing edges are inserted left to right with unique winners.

Warm A. Select $S_A = \{1, \ldots, M' - 1\}$ with $M' = M - r = (q - t)q$: components $(M', 1, \ldots, 1)$. This is the cold basis after its first $q - t$ blocks are complete and attached, so the same right-hand side and rule execute the last t cold blocks with all their strict winners: each block grows from its seed by $q - 1$ internal pivots and then attaches, $tq = r$ pivots in all. (This compares exact algebraic DSE states; it does not assume that a solver restarted from a saved basis preserves any numerical history.)

A.5 Counts and the corollary

Every cold block contributes internal symmetric work $2 + \cdots + q = q(q + 1)/2 - 1$, and attachments contribute the same total, so $W_0 = q^3 + q^2 - 2q$ and, with $\chi = 1$ exactly q times, $K_0 = \frac{3}{2} W_0 + (M - 1) - q = (3q^3 + 5q^2 - 8q - 2)/2$. Warm B's merged lengths are $M - 2r + 2, \ldots, M$, so $N_B = r$ and $K_B = r(M - r + 3)$. For warm A each internal block pivot has $a = 1$, $\chi = 0$ and $b = 1, \ldots, q - 1$, and the attachment of block k has left length $M' + (k - 1)q$, right length q and $\chi = 1$, giving $N_A = r$ and $K_A = 2t^5 - t^4/2 + 5t^3/2 - 2t$. Both overlaps equal $1 - r/(M - 1)$, and substituting $M = t^4$, $q = t^2$, $r = t^3$ gives the limits in the theorem. Both warm starts select u_1 and every later inserted column has index at least two, so after k pivots each basis has $2R - r - 1 + k$ nonzeros: the sparsity profiles agree. The valid dual bound $f(S) = \sum_j b_j \lambda_j$ gives both warm starts a relative gap of at most $4(1 + 3\varepsilon)r/[(1 - 3\varepsilon)(M - 1)] = O(1/t)$ against a cold gap of one.

Where warm B's work goes. Each pivot of warm B attaches a pair to the correct prefix, whose length grows from $M-2r$ to $M-2$. Its BTRAN result and DSE FTRAN result have $b=2$ nonzeros each, while the FTRAN result of the entering column spans the whole prefix ($a+1-\chi$ nonzeros; in network terms, the tree path closed by the entering arc). Almost all of K_B is therefore the FTRAN of the entering column, which every dual simplex iteration forms whatever its pricing rule: the separation is a property of the path, and DSE only fixes the order of the merges. The bases along the two warm paths have equal numbers of nonzeros throughout, so what differs is the sparsity of B^{-1} along the path. Warm B also costs $K_B/K_0 \sim 2t/3 = \Theta(M^{1/4})$ times the cold start, although it needs only $r/(M-1) \approx 1/t$ of the cold pivots.

Robustness. For a fixed basis row R_i the score $s_i(b) = (-R_i b)_+/\|R_i\|_2$ is 1-Lipschitz in b; cold winner gaps are at least ε/q, cascade gaps at least $(1-3\varepsilon)\sqrt{2} - (1+3\varepsilon)$, warm-B gaps at least $\sqrt{2}\varepsilon/M$, and all basic sign margins at least $1-3\varepsilon$. The eligible entrant and the dual ratio test depend only on A and c, not on b. Hence every perturbation with $\|h\|_2 < \varepsilon/(\sqrt{2}M)$ preserves all three paths and all counts. The radius shrinks with M and no uniform radius is claimed; at $t=16$ it is about 10^{-7}, the order of HiGHS' default feasibility tolerances. This completes Theorem 1.

Corollary 1. The warm-B proof holds for any $1 \le r' < M/2$. With $r' = r-1$ the start has overlap $1-(r-1)/(M-1)$ and $r-1$ pivots, so it is the unique overlap maximizer and pivot minimizer of the pair, and its work is $K'_B = (r-1)(M-r+4) \ge t^7/2$ against $K_A \le 3t^5$, a ratio of at least $t^2/6 > 1$ for all $t \ge 4$. □

Arithmetic model. The chain matrix has at most three entries per row. With support-sensitive triangular solves, row-wise pricing over the rows in the support of the BTRAN result, updates proportional to the charged supports, and factor maintenance that changes one diagonal sign and at most one superdiagonal entry per pivot, forming the three vectors and updating costs $\Theta(K)$. Initial values, reduced costs and exact row norms follow from component recurrences in $O(M)$; a max-heap of infeasible boundaries gives exact full row selection in $O(\log M)$ per pivot. The total is $\Theta(M+K) + O(N \log M)$, and $K_A = \Theta(M^{5/4})$, $K_0 = \Theta(M^{3/2})$, $K_B = \Theta(M^{7/4})$ dominate. The unbounded ratio is a statement about this model. An implementation with a per-iteration term linear in the number of rows (a full CHUZR scan, column-wise PRICE or dense solves), or with refactorization after a fixed number of updates, bounds the ratio of total work by a constant, although the constant can be large. Computing every exact initial DSE weight by a separate BTRAN instead of by the recurrences materializes $\Theta(M^2)$ entries on both warm starts, which masks the separation (below). An independent exact checker reproduces 20 complete strict DSE trajectories across the four starts (46,934 pivots through 20,735 rows) and four dense exact paths with all inverse matrices, reduced costs, solve-vector supports and factor profiles.

A.6 Initialization and the HiGHS realization

The theorem counts continuation work after exact initial weights are in place, and it assumes the linear-time initialization that these bidiagonal bases allow. If every exact initial DSE weight is instead computed by a separate BTRAN, the initial output counts are $J_A = \binom{M-r}{2} + r$ and $J_B = \binom{M-2r}{2} + 3r$ against $J_0 = M-1$ for the cold start. Both warm counts grow like $M^2/2$ and dominate the continuation, so $(J_B+K_B)/(J_A+K_A) \to 1$: in this cost model, overlap and pivot count miss the setup as well as the remaining path.

HiGHS computes exact initial DSE weights, one BTRAN per row, for any nonslack start that is not already dual feasible and near optimal; for such a near-optimal start it uses Devex instead. This holds in its default and DSE modes. In its default mode it can also switch from DSE to Devex during a solve in which the extra DSE solve stays much denser than the other solves.

Table 3 gives the three starts at $t=16$ under three settings. With presolve off, Devex forced from the first iteration and scaling off, the solve-vector supports that HiGHS reports match the exact DSE paths, and warm A and warm B, which share overlap 0.94 and 4,096 pivots, take 0.16 and 4.2 s, against 1.2 s from the cold start: warm B needs 1/16 of the cold pivots and is still 3.5 times slower than the cold start. In the runs with scaling off, HiGHS' continuation instructions per pivot on the cold and warm-A paths fit $\approx 14\text{k} + 730\,t^2$, a fixed cost of about 14k instructions per pivot plus a term proportional to K/N, and on warm B $\approx 140\,M$, so the solver follows the model's scaling. The measured ratio of 27 is below the model's $K_B/K_A = 121$ at $t=16$ because the solver's cost is not

Table 3: The chain LP of Theorem 1 at $t = 16$ (65,535 rows) in HiGHS 1.13.1: wall time in seconds of installing the start and solving (one thread, presolve off, dual simplex; one repetition at this size). Warm A and warm B share overlap 0.94 and 4,096 pivots; the cold start takes 65,535. The first two settings also turn off cost perturbation and HiGHS' less-infeasible DSE row choice; the defaults keep automatic edge weights, scaling and that row choice.

Setting	cold	warm A	warm B	warm B / warm A
Devex forced, scaling off	1.21	0.156	4.21	27.0
DSE requested, scaling off	1.48	34.0	34.6	1.02
HiGHS defaults	26.4	36.1	34.9	0.97

proportional to K alone: the fixed per-pivot cost weighs most on warm A's short pivots, and the times include the setup. With DSE requested, the exact initialization makes both warm solves about 23 times slower than cold. Under the full defaults the paths change as well as the initialization: both warm continuations cost $\Theta(M)$ per pivot and their ratio stays near 1.8, and the warm totals are within 4% of each other at 1.3 to 1.4 times cold, whose own solve takes 26 s under these settings. On our real LPs, whose crash bases are triangular and sparse, the exact initial weights cost a median 0.4% of the solve (Appendix E).

A.7 Metric alignment of the policy's loss

Let Q sample classes uniformly and then LPs from each class, and let $L_\delta = \log(C_\delta / C_{\text{ref}})$ be the log cost ratio of action δ against a common positive reference. For a policy whose draw is conditionally independent of solver randomness, $\mathbb{E}_Q \sum_\delta \pi_\theta(\delta \mid x) L_\delta$ equals the expected log cost ratio of the sampled deployment, and its exponential is the class-balanced geometric-mean ratio; at fixed x the convex combination is minimized by putting all mass on a minimum-cost action, and a common policy-independent baseline (here $\log C_0$) preserves risk differences. Eq. (2) is this identity with instructions as C, the unshifted start as reference, a clip on the per-action loss, and uniform rather than class-balanced weights over the training LPs (the training set is roughly class-balanced by construction). The argmax deployment differs from the sampled policy; a one-line bound gives $\mu_{h(x)}(x) - \mu_\star(x) \le |\mathcal{A}| \sum_\delta \pi(\delta \mid x)(\mu_\delta(x) - \mu_\star(x))$ for the argmax $h(x)$, a population statement only.

B Within-LP study and landscape measurements

This appendix gives the design and the full statistics of the within-LP study of §3.2 (Table 4), and the measurements behind the cost per iteration of learned starts and the roughness of the cost landscape.

Within-LP panel. The panel has 381 LPs from the W1 part of Dev-20k (25 classes; 179 have a primal or dual twin in the predictor's training data, and results without them are released with the code). Each LP is solved from up to 53 starts: 42 from the deployed decoder (33 shifts of the structural scores in $[-4, 4]$ at degree exponent $p = 0.5$, and the nine-shift grid at $p = 1.0$), ten high-overlap starts (sequential-softmax crash samples on $B^\star$ membership at temperatures 0.02 to 0.3), and the all-slack start; the median LP has 52 distinct starts, 17,329 in all. Every start of an LP was solved on one pinned core of an exclusive node with the deployed dual-simplex settings and a 120 s limit, with two timing repetitions in seeded random order; iteration counts were identical across repetitions and the median repeat difference in wall time was 1.1%. Overlap is measured against the nearest of the known optimal bases of the LP (the reference $B^\star$ and the distinct optimal final bases reached by the 53 solves, a median of 45 per LP; 78% of LPs have more than one). Five of the 17,329 solves failed and are kept at twice the time limit (PAR2). Rank statistics are class-balanced with 2,000-replicate bootstrap intervals over classes and then LPs. The perfect-proxy regrets in §3.2 take the best-time member among proxy ties. Table 4 lists all statistics.

Per-pivot cost. Figure 5 shows the gap between iterations and time per LP on the 400-LP pilot of Test-20k: on most LPs the predictor saves a far larger share of pivots than of time. On two earlier development panels, 874 LPs at the 20k size and 477 LPs at 40k to 60k variables (an earlier version of Dev-50k that still contained 23 duplicates of training LPs; Appendix H), the variance of $\log(T_{\text{predictor}} / T_{\text{all-slack dual}})$ decomposes by covariance share into iterations (70% and 78%), cost per iteration (29% and 9%) and start overhead (1% and 13%). Pivots from the learned start cost 1.50

Table 4: Within-LP statistics on the 381-LP panel of DEV-20K (25 classes, 17,329 distinct starts). Rank and variance statistics are computed within each LP and pooled; intervals are class-balanced bootstraps over classes and then LPs.

Statistic	Quantity	Value [95% CI]
Kendall τ with time	overlap	0.31 [0.22, 0.41]
	pivot count	0.65 [0.59, 0.71]
	instruction count	0.91 [0.89, 0.93]
Kendall τ, other pairs	overlap and pivot count	0.45 [0.36, 0.55]
	overlap and instructions	0.30 [0.20, 0.40]
	pivot count and instructions	0.63 [0.56, 0.71]
Variance of log time explained (R^2)	overlap alone	0.145
	$\log(1 + \text{pivots})$	0.706
	$\log(1 + \text{pivots})$ and overlap	0.707
	log instructions	0.985
Matched pairs (overlap within 0.005, pivots within 2%; 26,908 pairs)	share with time ratio > 1.25	0.092 [0.043, 0.152]
	share with time ratio > 2	0.019 [0.000, 0.055]
	quantiles 50/75/90/99%	1.03 / 1.08 / 1.17 / 1.72
	largest ratio	11.6
Proxy-dominance reversals (higher overlap and fewer pivots)	dominant start $> 5\%$ costlier	0.122 [0.082, 0.170]
	dominant start $> 25\%$ costlier	0.054 [0.029, 0.084]
Cost of proxy-optimal start (relative to the fastest of 42 decoder starts)	highest overlap	1.275 [1.18, 1.39]
	fewest pivots	1.089 [1.054, 1.130]
	best under both (exists on 31% of LPs)	1.087 [1.043, 1.149]

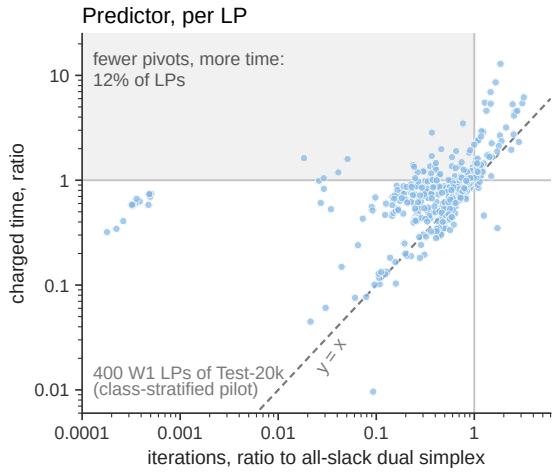

Figure 5: Per-LP iteration and charged-time ratios of the predictor against the all-slack dual simplex on a class-stratified pilot of 400 W1 LPs of TEST-20K (two timing repetitions averaged). Points in the upper left take fewer pivots and more time.

times as much as pivots from the all-slack start (geometric mean) on both panels, and 1.34 times on the 283-LP selection split of DEV-20K. The learned basis has 0.80 to 0.84 times the nonzeros of $B^\star$, and basis density does not track the ratio of the cost per iteration (Spearman $+0.07$). In instrumented solves, the cumulative solve-vector work (Appendix A), estimated from HiGHS' kernel statistics as calls times mean result density and dominated by the DSE FTRAN and the BTRAN, explains paired

instruction differences beyond iterations, overlap and initial fill (within-LP leave-one-class-out R^2 from 0.51 to 0.88), while no static property of the start predicts it (leave-one-class-out $R^2 \leq 0$).

Per-iteration traces. To see directly which iterations a learned start removes, we recorded the instructions of every iteration through HiGHS' simplex callback (the traced solves reproduce the plain solves' iterations and objectives exactly, and the callback's overhead, typically 0.3%, is subtracted) on 20 LPs of DEV-20K, one or two per class, for native HiGHS, the all-slack dual simplex, the predictor's start with the dual simplex and COSTSTART-FULL. For a reference path with N_R iterations and a learned path with $N_L < N_R$, we compare the reference's work per iteration over its first $N_R - N_L$ iterations with its work per iteration over its last N_L, and the learned path's work per iteration with the latter; setup is excluded. Against the all-slack dual simplex, the first iterations cost $0.44\times$ [0.28, 0.66] as much as the last ones for the predictor's start (below 1 on 15 of 17 LPs) and $0.45\times$ [0.26, 0.72] for COSTSTART-FULL (14 of 16), and the learned paths' iterations cost $0.89\times$ [0.71, 1.17] and $0.81\times$ [0.62, 1.06] as much as the reference's last ones; against native HiGHS the figures are 0.42 and 0.38, and 0.92 and 0.72. The learned starts skip the cheap early part of the path and pay about the price of its late part for each of their own iterations, which is why their mean cost per iteration is higher although their total work is lower. Installing a learned basis (`setBasis`, factorization and initial weights) costs about five times native HiGHS' setup, 11% of the learned solves' instructions against 1.2% for native HiGHS. Figure 6 shows eight of the traces, including three classes where the learned starts lose. The test does not apply where a learned start takes more iterations, and a few LPs go the other way.

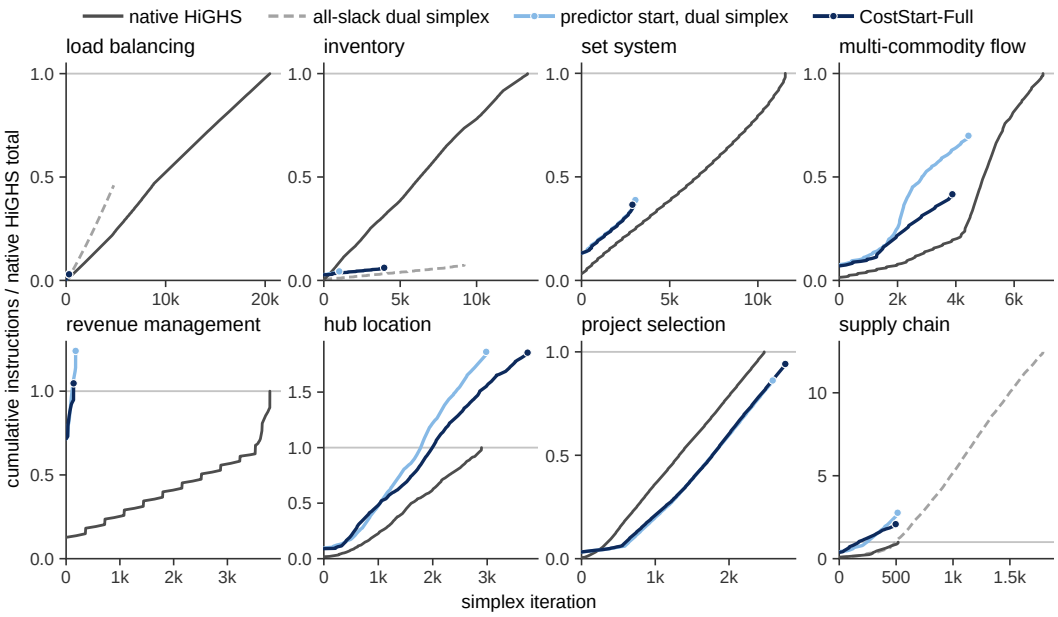

Figure 6: Cumulative instructions against iteration, normalized by native HiGHS' total, on eight of the 20 traced LPs of DEV-20K (the LPs at evenly spaced ranks of native HiGHS' iteration count). The value at iteration 0 is the setup; each curve ends at the solve's total, so a learned curve's end point is its total relative to native HiGHS. The all-slack dual simplex is drawn only where its path differs from native HiGHS'.

Roughness and basin. This panel has 312 LPs at the 20k size that the predictor never saw (24 classes), with 33 candidate starts each. On it, repeat solves are bitwise identical in iterations and objective and agree to 5×10^{-6} in log instructions at the median, so all variance among candidates is real. Swap statistics are measured on 60 of the 312 LPs and direct installations on 200.

- One random valid swap from the predictor's start changes log instructions by a median of 0.042 (90th percentile 0.28); 24% of swaps are better and 24% worse by more than 5%. $|\Delta|$ grows only slowly with the number of swaps, while the mean drifts upward.
- The best of eight stochastic-crash samples takes $0.86\times$ the instructions of the deterministic start, but the mean sample is no better than it. A cross-fitted selector on cheap features of the start captures 34% of the gap to the best of the 33 candidates within the training classes and about 10% outside them.

- An exact optimal basis installed directly (HiGHS repairs singular sets on install; charged) solves in $0.051\times$ [0.036, 0.075] the predictor's instructions, with 86% of LPs solving in zero iterations; such a basis is primal and dual feasible, so HiGHS also skips the initial DSE weights and the cost perturbation. The same basis through the crash costs $0.595\times$, because the crash drops a median of 221 of its basic columns.
- Random swaps of 0.1%, 1%, 5% and 20% of the optimal basis cost 0.30, 0.67, 1.47 and $2.87\times$.
- Model-generated complete bases sit about 6% (about 900 swaps) from an optimum and cost 1.03 to $1.09\times$ installed directly.

C ITERATIONS AND TIME IN THE PUBLISHED RESULTS

The gap between iteration savings and time savings is visible in the published papers' own tables. Table 5 lists, for every dataset and solver setting that Fan et al. (2023) and Lu & Yan (2025) report, the ratio of the learned start's iterations and wall time to those of the solver's default start, computed from their tables. In 18 of the 21 settings the time ratio is larger than the iteration ratio, often by a wide margin (OptVerse STOCH: $0.42\times$ the iterations and $0.96\times$ the time); the three exceptions are marked. These ratios come from different data, solvers, decoders and charging rules than ours, and we do not compare them with our numbers; they only show that the difference is not specific to our setting.

Table 5: Iteration and time ratios (learned start / default start) computed from the published tables: Fan et al. (2023), Tables 1 (OptVerse) and 2 (HiGHS), dual simplex; Lu & Yan (2025), Tables 3 to 7 (HiGHS dual simplex, their tripartite model). Geometric means over the datasets of each column. † time falls more than iterations.

Fan et al. (2023): iterations / time			Lu & Yan (2025): iterations / time		
Dataset	OptVerse	HiGHS	Dataset	unpresolved	presolved
LIBSVM	0.61 / 0.66	0.58 / 0.62	Mirp1	0.42 / 0.52	0.32 / 0.42
MIRP	0.64 / 0.70	0.61 / 0.81	Mirp2	0.93 / 1.05	0.86 / 0.83†
STOCH	0.42 / 0.96	0.48 / 0.35†	Anonymous	0.59 / 0.72	0.74 / 1.19
GEN	0.23 / 0.38	0.04 / 0.15	Load balance	0.46 / 0.71	0.63 / 0.70
SC-1	0.10 / 0.29	0.25 / 0.52			
SC-2	0.14 / 0.25	0.18 / 0.33	5 large MIRP LPs	0.93 / 0.86†	
Geometric mean	0.28 / 0.48	0.25 / 0.41	Geometric mean	0.57 / 0.73	0.60 / 0.73

D PREDICTOR, DECODER AND SHIFT POLICY

This appendix details the predictor and its training (§4.1), the decoder (Algorithm 1), and the shift policy with its training, ablations (Table 6) and controls (Table 7). Appendix D.1 explains where the shift policy's gain arises. Table 12 collects all settings.

Graph and features. The LP is a graph with one node per structural column, per slack column and per constraint, and an edge for every nonzero of A, in the style of Gasse et al. (2019). Node and edge features are normalized objective coefficients, bounds, right-hand sides, degrees and matrix entries. The target is membership in the optimal basis that HiGHS returns from the all-slack start, which is one optimal basis among possibly several.

Predictor training. Stage 1 trains on the 114,874 LPs of TRAIN-SMALL (37 classes, median 830 columns and 612 rows after presolve) in three chained runs of 2, 4 and 6 epochs (43k steps, 8.8 H100-hours) with TF32 matrix multiplication. The loss per LP is the basis-membership cross-entropy over all $n + m$ columns of $[A\ I]$, divided by m, plus 0.2 times the cross-entropy of an auxiliary dual-tightness output and 0.2 times that of the nonbasic-bound outputs. Stage 2 trains on 3,457 LPs of TRAIN-20K (26 classes, median 18.6k columns) for 6 passes at a constant learning rate, in fp32 with deterministic kernels. A checkpoint is saved every 37 steps and selected on the 283-LP selection split of DEV-20K by the paired wall time of `setBasis` and the solve against the model after stage 1 (best at step 333, ratio 0.847). Ablations:

- Overlap of the crash with $B^\star$ rises from 0.645 (an early 4-layer model) to 0.787, and precision at $|B^\star|$ to 0.76; the crash on $B^\star$'s own membership reaches 0.93.

- Depth: 4 vs. 8 layers at matched data give overlap 0.730 vs. 0.742. Twelve layers add 0.004 at 1.2 to 1.3 times the forward cost and take $1.043\times$ [0.95, 1.14] the charged time of 8 layers on DEV-20K (one timing repetition). GATv2, TransformerConv and GINE backbones agree within 0.005 at depth 4.
- Length of stage 1: 2, 6, 12, 18 and 24 epochs give overlap 0.761, 0.780, 0.787, 0.791 and 0.793; charged time saturates at 12 epochs (18 vs. 12: $1.018\times$ [0.97, 1.08]).
- Retraining stage 2 on the deduplicated 9,511 LPs of TRAIN-20K at the same number of LP visits gives overlap 0.787 and precision 0.763, identical to the deployed checkpoint.

Deployment. The forward pass runs one LP at a time (microbatching perturbs scores by 10^{-6} and flips crash decisions) on a fused GATv2 kernel with TF32 matrix multiplication: 8-layer inference falls from 19.5 to 6.4 ms median. TF32 changes 87% of starts, so every policy, gate and panel uses the deployed path.

Decoder. Algorithm 1 gives the triangular crash. Consider the bipartite graph between structural columns and rows in which an edge (j, i) exists when column j has an entry in row i of at least 10^{-8} times the column's largest entry; its gain is $g_{ji} = (s_j + \delta - s_{n+i})/\deg(j)^p$ with $p = 0.5$, where s_{n+i} is the score of row i's slack and $\deg(j)$ the number of eligible entries of column j. Selecting column j in row i removes every remaining edge of column j and every edge whose row lies in the support of column j, so the selected columns form a triangular submatrix with nonzero diagonal, and the basis is nonsingular without repair. A shift δ raises column j's gains by $\delta/\deg(j)^p$, which tends to admit more structural columns for $\delta > 0$ and fewer for $\delta < 0$, and tilts the choice toward low-degree columns. A finer grid of shifts, and a second per-LP decoder dimension (the exponent p alongside δ), raise the oracle but not the learned policy (Table 6).

Algorithm 1 Triangular crash with a structural shift

Require: scores $s \in \mathbb{R}^{n+m}$ (structural columns $1..n$, slacks $n+1..n+m$), shift δ, exponent p
$E \leftarrow \{(j,i) : |a_{ij}| \geq 10^{-8} \max_k |a_{kj}|\}$; $\deg(j) \leftarrow |\{i : (j,i) \in E\}|$
$g_{ji} \leftarrow (s_j + \delta - s_{n+i})/\deg(j)^p$ for $(j,i) \in E$; $\mathcal{B} \leftarrow \emptyset$
while some $(j,i) \in E$ has $g_{ji} > 0$ **do**
  take $(j,i) = \arg\max g_{ji}$ (ties: smaller slack score, then smaller index)
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{(j,i)\}$ ▷ column j becomes basic in row i
  remove from E every edge of column j and every edge whose row lies in $\mathrm{supp}(a_j)$
end while
every row i without a structural column keeps its slack $n+i$ basic
set nonbasic bound statuses from the bound types and the signs of the scores
return basis $\mathcal{B}$ with statuses

Decoder tie rules and statuses. Within a column, candidate rows are ordered by the score of their slack, ascending, with ties broken by row index; heap ties go to the smaller column index. Nonbasic statuses: lower-only bound $\rightarrow$ lower, upper-only $\rightarrow$ upper, free $\rightarrow$ zero, fixed $\rightarrow$ lower, boxed $\rightarrow$ upper if and only if the logit of the bound-status output is positive. The reference Python and the native implementation produce identical selections. Statuses are not a confound for any result: forcing a start's statuses to those of another method leaves instruction counts bitwise unchanged on a development panel.

Shift policy: training. The cost of an action is the number of user-space instructions, read from `perf_event` around `setBasis` and the solve and summed; on the cost table, repeated solves agree to a median of 5×10^{-5} and at most 6×10^{-4} in log instructions. A solve is a failure if it is not optimal, if its basis was changed on install, or if its objective misses the reference by more than 10^{-6} relative; a failure costs $\log(2 \cdot 120\,\mathrm{s} \cdot 5 \times 10^9)$. The policy is an MLP $84 \rightarrow 128 \rightarrow 128 \rightarrow 9$ (29k parameters) with SiLU and dropout 0.1, whose last layer is initialized so that the untrained policy plays $\delta = 0$; the two larger policies have 11 and 20 outputs and ten-seed ensembles (Appendix E). The policy stage costs 1.7 ms per LP (3.9 ms with the novelty gate). The training data are the 7,699 W1 LPs of TRAIN-20K times 9 actions, drawn from a 131,706-solve table over 14,634 LPs that took 56 minutes on 64 cores (about 60 core-hours; 71 solves were failures); COSTSTART-ALG adds the costs of native HiGHS and primal simplex and the W2 LPs, 9,349 LPs in all. Training uses AdamW (learning rate 10^{-3}, weight decay 10^{-4}), batches of 256 LPs and 100 epochs; the epoch is chosen on the 283-LP selection split of DEV-20K (a held-out fifth of the training LPs gives the same numbers), with three seeds and the argmax of the mean log-probability. The loss clip at 3 keeps $20\times$ blow-ups

visible; clipping at 1 hid them, and that policy played catastrophic shifts on six bin-packing LPs of a development panel.

Shift policy: ablations. Table 6 lists the ablations. No change of the recipe moves the result outside seed noise, a finer or two-dimensional action grid raises the oracle but not the policy, and the policy does not transfer to held-out classes without the novelty gate (on load balancing it plays ± 3 on every LP). Adding proxies of the formulation and of recall to the inputs helps within the training classes by two points and does not transfer, and a margin rule on the ensemble is flat from 0 to 4.

Table 6: Ablations of the shift policy: instructions relative to the unshifted predictor (class-balanced geometric means), on the 283-LP selection split of DEV-20K for the first two blocks. Single seeds unless noted. The action-set rows compare with the nine shifts on the same LPs. Transfer rows hold out each class, variant or dual form of the training LPs in turn (three seeds, epoch chosen on a held-out fifth of the training LPs); the worst held-out classes are knapsack (2.10), telecom (2.17) and load balancing (2.01).

Change	Ratio
Recipe	
main recipe, 5 seeds (mean $\pm$ sd)	0.896 ± 0.013
ensemble of the 5 seeds	0.900 [0.839, 0.961]; pivots 0.920; summed instructions 0.905
hidden width 64 / 256 / 512 / 1024	0.900 / 0.900 / 0.904 / 0.925
learning rate 3×10^{-4} / 3×10^{-3}	0.896 / 0.898
loss clip 1 / 6 / none	0.903 / 0.891 / 0.888
dropout 0 / 0.3	0.893 / 0.902
training LPs 1k / 3k / 7k (per seed)	0.919 / 0.913 / 0.898
ensemble of ten seeds	0.879
Action set	
shifts in $[-1, 1]$ (5 actions)	0.954 (oracle 0.833)
shifts in $[-4, 4]$ in steps of 0.25 (33 actions)	0.917 vs. 0.912 for the nine shifts
nine shifts $\times$ degree exponent $p \in \{0.5, 1\}$	0.883 vs. 0.889 (oracles 0.718 vs. 0.767)
Transfer	
leave one class out	1.144 to 1.171
leave one variant out (39 variants)	1.027 [0.91, 1.19]; with novelty gate 0.996 (99% abstention)
leave one form (primal or dual) out	1.154; with novelty gate 1.000

Table 7: Controls for the shift policy on the 283-LP selection split of DEV-20K (22 classes): instruction ratios to the unshifted predictor. The oracle picks the best of the nine shifts per LP.

Rule	Ratio
best fixed shift chosen on training data	1.000 ($\delta = 0$)
every other fixed shift	1.005 to 1.090
best fixed retune of the decoder (shift and degree exponent)	0.967
best constant shift per class, chosen in hindsight	0.926
learned policy, score statistics only (no embedding)	0.975
learned policy, embedding only	0.897
learned policy as deployed (3-seed ensemble; single seeds 0.896 ± 0.013)	0.898
learned policy, 10-seed ensemble	0.879
oracle over the nine shifts	0.767
learned policy, leave-one-class-out	1.144 to 1.171
with the novelty gate	1.000

What the learning contributes. Table 7 shows that the shift policy's gain is a learned per-LP decision and not a retuning of the decoder. Every fixed shift is worse than none, the best decoder retune gains 3%, the best constant per class chosen in hindsight gains 7%, and the learned per-LP policy gains 10% (12% with ten seeds), 41% of the oracle's log gain. The predictor's embedding carries the signal. On the predictor side, training length and the stage-2 fine-tune reduce solve time and greater depth does not: on DEV-20K the model trained only on TRAIN-SMALL is at parity with native HiGHS, the fine-tune takes it to $0.82\times$, and a 12-layer network at matched budget is 4% slower than the 8-layer one.

D.1 MECHANISM: A LEARNED CONTROL OF STRUCTURAL RECALL

A uniform shift helps although §3 found the cost landscape rough, because the landscape is rough at the scale of single swaps but smooth along one coarse direction, the number of structural columns the crash admits. The predictor's losses concentrate on LPs where it admits too few of the optimal structural columns. Recall, the fraction of $B^\star$'s structural columns that a start contains, is the start property whose effect transfers best across classes (leave-one-class-out R^2 of 0.3), and it interacts with degeneracy: degenerate LPs with recall below 0.3 run at 1.3 to 2.3 times the all-slack dual simplex, while degenerate LPs with adequate recall behave like nondegenerate ones ($0.7\times$). Alternative optima are not the explanation (recall against the nearest of six optima found rises by only 0.02), and neither is the crash's stopping rule: the predictor scores the right kind of columns highly, but too few of them. The shift policy's gain is concentrated on these LPs (Figure 7b). Where the predictor's recall is below 0.3, the policy runs in $0.81\times$ its time with $0.71\times$ its iterations, against $0.98\times$ where recall exceeds 0.6. The shift policy is thus a learned per-LP control of how many structural columns to commit to, chosen against what that commitment costs the solver.

The harm gate targets the same LPs, which is why it subsumes most of the shift policy's gain in COSTSTART-SHIFT: fit to the cost of the unshifted start, it hands many of them to native HiGHS before the shift is used. On the frozen test sets, with the gate present, adding the shift policy changes the result by less than 4% (TEST-20K: no change; TEST-50K and TEST-80K: 2% slower; an 80-LP W2 pilot of TEST-20K: 3.5% faster). Without the gate, the shift policy takes $0.94\times$ the predictor's time on TEST-20K but moves it by at most one point on TEST-50K and TEST-80K, because the novelty gate sends most of those LPs to the unshifted start.

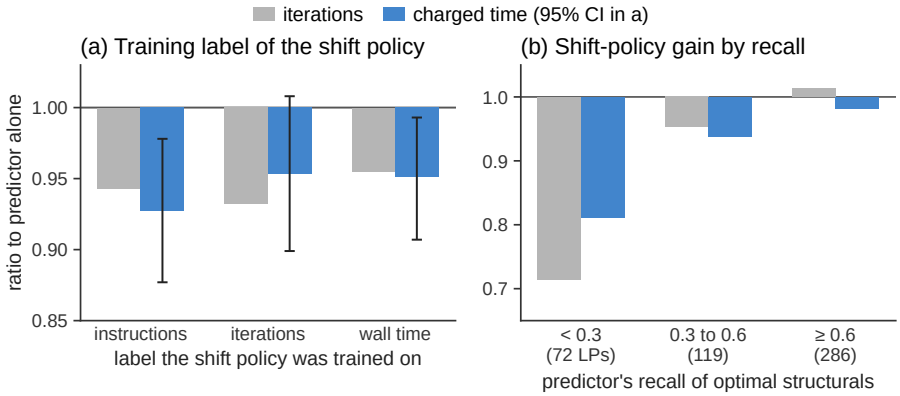

Figure 7: (a) Shift policies that differ only in their training label, on DEV-20K (charged time, three repetitions; Appendix I.1): the policy trained on iterations removes more pivots and saves less time. (b) The shift policy's gain over the predictor as a function of the predictor's recall of the optimal structural columns, on 477 development LPs at 40k to 60k variables (an earlier version of DEV-50K that still contained 23 duplicates of training LPs), with a shift policy trained before the audit. Panel (b) only locates where the gain arises; no reported comparison uses it.

E DECISION POLICIES WITH ALGORITHM AND PRICING ACTIONS

This appendix covers the design of the action sets $\mathcal{A}_2$ and $\mathcal{A}_3$ (Table 8), the training of COSTSTART-ALG and COSTSTART-FULL and their development readouts (Table 9), the probe of solver options behind the pricing action (Table 10), and the decomposition of Figure 4 (Table 11).

Design of the action sets. Native HiGHS and primal simplex were added when a held-out-class panel showed that on some classes no learned start beats HiGHS' own choice of algorithm. Per LP, the oracle over $\mathcal{A}_2$ reaches $0.67\times$ the unshifted start's instructions on the 275 LPs of the probe below, against $0.76\times$ over the shifts alone; adding dual simplex from the all-slack basis, which nearly duplicates native HiGHS, or the classical crash made the learned policy worse on held-out training LPs. The oracle over $\mathcal{A}_3$ reaches $0.54\times$, and its gain over $\mathcal{A}_2$ lies almost entirely in the work per iteration (iterations stay at $0.72\times$ while the work per iteration falls from $0.92\times$ to $0.74\times$). The decoder offers no such lever: every crash start is exactly triangular, and static properties of the start

explain about 2% of the within-LP variance of work per iteration beyond the iteration count, so the policy must reach the cost per iteration through a solver option. Every design choice was made on a fixed held-out fifth of the training LPs (Table 8).

Table 8: Design of the action sets on a held-out fifth of the training LPs: instructions relative to the predictor (class-balanced geometric means). For COSTSTART-ALG, W1 is reported on two sub-splits. Configuration only: a selector among HiGHS' own starts under both pricing rules, without the learned start.

Action set	W1	W2
Design of COSTSTART-ALG		
shifts only	0.906 / 0.935	0.851
shifts and native HiGHS	0.810 / 0.781	0.792
shifts, native HiGHS and primal simplex ($\mathcal{A}_2$, chosen)	0.783 / 0.754	0.605
$\mathcal{A}_2$ plus all-slack dual simplex and classical crash	0.796 / 0.773	0.648
Design of COSTSTART-FULL		
start and algorithm ($\mathcal{A}_2$)	0.785	0.645
plus the nine shifts under Devex ($\mathcal{A}_3$, chosen)	0.679	0.666
plus native HiGHS under Devex	0.690	0.850
shifts under both rules, without the strategy actions	0.732	1.017
configuration only	1.166	1.895

Training of COSTSTART-ALG and COSTSTART-FULL. The training rows of COSTSTART-ALG are the nine shifts of the shift policy's table plus the two strategy actions (native HiGHS, and the unshifted start with primal simplex), each relative to the same collection's unshifted dual-simplex solve, clipped at ± 3 with failures at $+3$, on 9,349 LPs of TRAIN-20K from W1 and W2. The table of COSTSTART-FULL adds, per training LP, the nine shifts under Devex and native HiGHS under both pricing rules (the latter used only by the configuration-only control), bridged to the existing rows through each collection's own unshifted solve under default pricing; 7,699 W1 and 1,501 W2 LPs had been collected when the policy was frozen, 9,197 with every action. Both policies use the shift policy's MLP, inputs and loss, with ten-seed ensembles, 150 epochs, learning rate 3×10^{-3}, dropout 0.1, batches of 256 LPs and window weight 1. Instruction counts agree across CPU models. The 20-action table took 184k solves and 373 core-hours of loaded single-thread time, 15.7 times the instructions of one all-slack dual-simplex solve per LP ($20.1\times$ in W1, $14.5\times$ in W2); the eleven actions of $\mathcal{A}_2$ account for $8.5\times$ and the Devex shifts for $7.2\times$.

Design readouts. For COSTSTART-ALG, sixty training epochs undertrained the policy (0.870 / 0.883 / 0.719 in the layout of Table 8); hidden width 256 and 300 epochs were within seed spread (three disjoint ten-seed ensembles differ by about 0.02). Leave-one-class-out without the novelty gate gives 1.196; with the gate it gives 1.000 when the gate falls back to the unshifted start, 1.452 to native HiGHS and 2.266 to primal simplex, so the fallback stays the unshifted start. At the freeze of COSTSTART-FULL, the held-out fifth held 1,531 W1 LPs of 25 classes and 316 W2 LPs of 11: COSTSTART-FULL took $0.669\times$ [0.589, 0.751] and $0.668\times$ [0.526, 0.826] the predictor's instructions and COSTSTART-ALG $0.822\times$ and $0.615\times$; paired, COSTSTART-FULL took $0.813\times$ [0.733, 0.897] the instructions of COSTSTART-ALG in W1, with iterations $1.00\times$ and work per iteration $0.81\times$, and $1.086\times$ [0.946, 1.359] in W2.

Table 9: Development readouts of the two larger policies: charged time relative to the predictor (class-balanced, 95% intervals where available). The DEV-20K panel has 207 W1 LPs of 23 classes and 45 W2 LPs of 8; DEV-50K has 290 W1 LPs of 23 classes and 164 W2 LPs of 16. No training LP is larger than 25,000 variables.

	DEV-20K panel		DEV-50K	
Method	W1	W2	W1	W2
COSTSTART-FULL	0.761 [0.618, 0.912]	0.524 [0.409, 0.666]	0.891 [0.813, 0.971]	0.857 [0.719, 0.993]
+ check and harm gate	–	–	0.751	0.682
COSTSTART-ALG	0.801	0.462	0.941 [0.88, 1.00]	0.856 [0.73, 0.97]
+ check and harm gate	–	–	0.796	0.679
predictor, Devex on every LP	0.980	1.189	0.865 [0.730, 1.019]	1.372 [0.943, 2.326]
configuration only	1.164	0.904	–	–

Table 9 gives the charged development readouts. On the DEV-20K panel, COSTSTART-FULL's gain in W1 splits into iterations $0.94\times$, time per iteration $0.77\times$ [0.67, 0.88] and fixed overhead $1.06\times$.

COSTSTART-ALG was read on the selection split during design, and the W1 part of this panel shares 98 LPs with it, so the selection split and that W1 part are development readouts only; the clean evaluations are W2 of the panel, DEV-50K and the test sets. On DEV-50K, with the harm gate applied to both, COSTSTART-ALG takes $0.98\times$ and $0.97\times$ the gated predictor's time on W1 and W2. On W2 of DEV-50K, COSTSTART-FULL takes $1.000\times$ [0.955, 1.051] the time of COSTSTART-ALG and $0.604\times$ [0.360, 0.872] that of the predictor with Devex on every LP. On W1 it takes $0.923\times$ [0.853, 0.996] the instructions of COSTSTART-ALG and ties Devex on every LP ($1.031\times$ [0.852, 1.244] in time), because the novelty gate sends 152 of the 290 LPs to its fallback under default pricing. A variant whose fallback uses Devex helped on the W1 part that motivated it ($0.889\times$ the frozen COSTSTART-FULL's instructions) but not out of sample on W2 ($1.076\times$ [0.954, 1.224] in time), so the frozen COSTSTART-FULL is the one we report.

Harm gate with the policies. When combined with COSTSTART-ALG or COSTSTART-FULL, the harm gate is evaluated on every LP, on the start the policy deployed (for the primal and native actions, the unshifted start); if it predicts too small a saving, native HiGHS runs instead, and the policy's start, the gate features and the gate evaluation are charged. The gate was trained on the unshifted start with dual simplex and default pricing, so it judges primal and Devex choices with a model fit to another action; on TEST-20K most of its overrides reverse primal-simplex choices (137 of 286 for COSTSTART-ALG, 117 of 156 for COSTSTART-FULL). COSTSTART-ALG and COSTSTART-FULL can choose native HiGHS themselves. Adding the harm gate slows them at the training size (COSTSTART-ALG $0.54\times$ instead of $0.51\times$ native, COSTSTART-FULL $0.52\times$ instead of $0.49\times$ on TEST-20K), because their own native action covers most LPs the gate would hand over while the gate's features and evaluation are charged. At larger sizes the gate is needed, because the novelty gate sends most LPs to the unshifted start, where the policy never gets to choose native HiGHS.

Pricing on the test and development panels. On TEST-20K, COSTSTART-FULL takes $0.72\times$ [0.63, 0.81] the predictor's time and $0.80\times$ [0.70, 0.90] that of COSTSTART-SHIFT. Against COSTSTART-ALG it is at $0.96\times$ [0.88, 1.05], with 15% more iterations that each cost 18% less. By window (Table 15), COSTSTART-FULL takes 0.55, 0.36 and $0.33\times$ native HiGHS on W1, W2 and W3, against 0.60, 0.33 and $0.22\times$ for COSTSTART-ALG, so its gain is concentrated on short solves. Development data showed the same pattern, on held-out training LPs ($0.81\times$ the instructions of COSTSTART-ALG, iterations unchanged) and on the DEV-20K panel ($0.95\times$ [0.83, 1.08] the time of COSTSTART-ALG, $0.81\times$ its time per iteration). The predictor's start with Devex on every LP takes $1.07\times$ the predictor's time on TEST-20K, and a selector trained with the same loss to choose among HiGHS' own starts and the two pricing rules takes $1.26\times$; COSTSTART-FULL takes $0.67\times$ and $0.57\times$ their times. Table 17 lists the actions the three systems take on TEST-20K.

Probe of solver options. On 275 LPs of the selection split, the predictor's start was solved under 14 option settings, the default included (Table 10). Devex was the only one that clearly helped. Devex relative to default pricing ranges by class from 0.51 (knapsack, a pure per-iteration effect) to 2.39 (project selection). Wall-time ratios measured under 39 concurrent workers track the instruction ratios with a per-LP correlation of 0.987. Setup under default pricing costs $1.12\times$ the Devex setup; the difference, the cost of the exact initial DSE weights, is a median 0.4% of the solve (90th percentile 3.6%). At this size Devex's gain is therefore cheaper iterations rather than avoided weight initialization, unlike on the chain LP of Appendix A. Per-LP oracles relative to the predictor: shifts 0.764; shifts under both rules 0.589; $\mathcal{A}_2$ 0.667; $\mathcal{A}_3$ 0.538, a paired $0.807\times$ [0.736, 0.879] beyond $\mathcal{A}_2$ with iterations unchanged (0.726 to 0.724) and work per iteration from 0.919 to 0.743.

Headroom inside the decoder and what determines work per iteration. On the within-LP panel of Appendix B:

- Oracle headroom. The instruction oracle over the nine shifts is $0.947\times$ [0.919, 0.970] the iteration oracle, with $1.04\times$ the iterations and $0.91\times$ the work per iteration, and over the 42 decoder starts it is $0.909\times$ (1.10 and 0.83). Within an LP, log iterations and log work per iteration correlate with a median of -0.54, so starts with fewer iterations tend to have costlier ones.
- Triangularity and setup. Every one of the 17,329 starts is exactly triangular (no nucleus after singleton peeling, so no fill beyond the basis), and setup (installing and factorizing the start and initializing the pricing weights) is a median 3.1% of the solve's instructions. Within an LP, log work per iteration has $R^2 = 0.433$ against log iterations alone and 0.457 with the structural count, basis nonzeros, the nonzeros of a COLAMD LU factorization and the measured factorization cost added.

Table 10: Probe of HiGHS options on the predictor's start (275 LPs of the selection split of DEV-20K): instructions, iterations and work per iteration relative to HiGHS' default options (class-balanced geometric means), and failures. HiGHS' default dual edge weights use DSE with automatic switches to Devex.

Option	instructions	iterations	work per iteration	failures
Devex pricing	0.910	1.067	0.853	0
Dantzig pricing	1.846	2.073	0.891	7
DSE without the switch to Devex	1.057	0.982	1.076	0
price strategy 1 / 2	1.010 / 0.994	1.000	1.010 / 0.994	0
cost perturbation off / $\times 5$	1.160 / 1.046	1.121 / 1.035	1.035 / 1.011	1 / 0
factor pivot threshold 0.01 / 0.5	1.007 / 1.014	1.005 / 1.011	1.003 / 1.003	0
less-infeasible DSE check off	0.992	0.981	1.011	0
scaling off	1.084	1.098	0.988	1
update limit 100	1.149	1.011	1.136	0
Devex and cost perturbation off	1.115	1.263	0.883	1

- Noise floor. Calling `setBasis` with the same basic set in three selection orders gives bytewise identical statuses. Setup instructions change by a median 0.3% under random row and column relabelings of the LP, and whole-solve instructions by a median 2.1% through tie-breaking; the better of two relabelings of one start is $0.971\times$ the other, a noise floor for any oracle gain.

Offline, on the 283-LP selection split, the shift policy's gain over the predictor splits into iterations $0.907\times$ and work per iteration $0.974\times$ [0.916, 1.036], that of COSTSTART-ALG into $0.808\times$ and $0.999\times$, and the gain of the instruction-trained shift policy over the iteration-trained one into iterations $1.010\times$ and work per iteration $0.932\times$ [0.872, 0.991]; in the charged label ablation (Appendix I.1) the last split is $1.012\times$ and $0.957\times$ [0.925, 0.987].

Decomposition on TEST-20K. Figure 4 uses the per-LP rows of the TEST-20K panel with the check inline, restricted to LPs where both the method and native HiGHS are valid and take at least one iteration (all 1,356 for every method shown except the configuration-only control, 1,353); intervals are two-level bootstraps over classes and LPs. Table 11 gives four paired contrasts split the same way by Eq. (1). The analysis script is released with the code.

Table 11: Paired contrasts on TEST-20K, split by Eq. (1) into iterations, time per iteration and fixed overhead (class-balanced geometric means of the first method relative to the second, 95% intervals where computed).

Contrast	time	iterations	time per iteration	fixed
COSTSTART-FULL vs. COSTSTART-ALG	0.959 [0.884, 1.046]	1.149 [1.051, 1.282]	0.821 [0.730, 0.909]	1.017
COSTSTART-ALG vs. COSTSTART-SHIFT	0.831 [0.744, 0.915]	0.820	1.084	0.934
COSTSTART-FULL vs. COSTSTART-SHIFT	0.796 [0.696, 0.903]	0.942	0.890 [0.815, 0.961]	0.950
predictor vs. Lu & Yan	0.725 [0.637, 0.823]	0.671	1.078 [0.946, 1.212]	1.002

F GUARDRAILS

This appendix details the three guardrails of §4.3: the novelty gate and the harm gate, which decide before the solve, and the optimality check (Appendix F.1), which acts after it. Table 12 lists their settings.

Novelty gate. The policy is learned within the training classes; outside them it chooses harmful actions (leave-one-class-out, 1.14 to 1.17 times the predictor's time for the shift policy and 1.20 for COSTSTART-ALG), and thresholding its confidence does not help, because the ensemble assigns high probability to harmful actions. The nearest-neighbor test on the standardized $[h, z]$ vector abstains on 98 to 100% of held-out-class LPs and on 2% of in-class ones, brings the held-out-class ratio to 1.000 of the predictor, and costs 2 ms. Abstaining to native HiGHS instead of the unshifted start would be worse on the training classes held out one at a time ($1.45\times$); §6.3 shows that this choice does not carry over to new classes.

Harm gate. A gradient-boosted regressor predicts $\log(I_{\text{learned}}/I_{\text{native}})$ from 73 deployment-time features: the LP's size and sparsity statistics, statistics of the predicted scores, and the composition of the decoded start. It uses no class labels. When the predicted saving is below 12%, native HiGHS runs

instead. The model and threshold were chosen by leave-one-class-out cross-validation on the training classes. To measure how much of the gate's value comes from recognizing the training classes, we also evaluate a class-blind variant: for each test LP, a gate refit with that LP's class removed from its training data.

Gate details.

- Novelty gate: standardized 84-dimensional $[h, z]$ against the 7,699 training rows, threshold 2.466, the 0.99 quantile of nearest-neighbor distances from a random fifth of the training rows to the rest. Every quantile in $[0.5, 0.99]$ gives leave-one-class-out 1.000, and 0.99 was chosen for the smallest in-class abstention (2%).
- Harm gate: models compared by leave-one-class-out cross-validation on 3,466 training rows (ridge, cost-weighted logistic regression, and boosted trees with squared or Huber loss); boosted trees with Huber loss were chosen (150 trees of depth 3, learning rate 0.05, leaves of at least 25 rows). The threshold is chosen from 201 quantiles of the out-of-fold predictions to minimize the class-balanced time-weighted loss: keep the learned start if and only if $\hat{y} \leq -0.128$. Its tree evaluation costs 3.3 ms per LP and is charged.
- On DEV-20K the harm gate turns bin packing from $1.75\times$ native HiGHS to $1.05\times$ and hub location from $1.98\times$ to $0.99\times$. Fit without the LP's class, it loses most of its gain (Table 1), so its skill within the training classes is partly recognition of those classes, which is why the novelty gate is reported beside it.
- A budgeted restart (abort the learned solve after $4m$ iterations and run native HiGHS) removes one stall on the development panel and is available as a tail guard.

F.1 OPTIMALITY CHECK

The failure. On one badly scaled class, load balancing in its primal form, a learned start that is not dual feasible can end at a basis that HiGHS accepts as optimal but whose objective is up to 20% too high. These LPs minimize one makespan variable subject to 47k inequality rows with coefficients up to 10^4; the optimum is about 0.003 and every optimal dual is of order 10^{-7}. HiGHS returns a basis that is optimal within its absolute dual feasibility tolerance of 10^{-7}: about 8,800 reduced costs have the wrong sign, each below the tolerance, and their sum is about 4×10^{-4}. On this class the tolerance is thus of the size of the optimal duals themselves. The likely mechanism is that HiGHS starts the dual simplex in phase 2 when the dual infeasibilities of the start are small, absorbs them by shifting costs, and after removing the shifts finds no wrong-signed reduced cost above tolerance, so it triggers no cleanup. Native starts, which are dual feasible on this class, never failed this way, so the check is charged to the learned start.

The check. HiGHS' sum of dual infeasibilities, unlike its count, includes values below tolerance, which is why it detects these failures. If the sum exceeds $10^{-6}(1 + |f|)$ for objective value f, the solve continues from the returned basis with the primal and dual feasibility tolerances tightened to 10^{-9}; if that does not reach optimality, native HiGHS solves the LP afresh. The threshold compares a sum of reduced costs with an objective value, so it is an empirical rule and not a certificate; it rests on a separation of more than three orders of magnitude on development data. Scaling the objective of this class would be a solver-side remedy; the check needs no class-specific setting.

Development evaluation. On the 477 development LPs at 40k to 60k variables (an earlier version of DEV-50K that still contained 23 duplicates of training LPs), HiGHS reported 2,370 solves as optimal. Twelve of them were wrong, every one a learned start on a primal-form load-balancing LP, as were the wrong-objective terminations on TEST-50K. Candidate checks on these 2,370 solves:

- HiGHS' sum of dual infeasibilities with threshold $10^{-6}(1 + |f|)$ flags 12 of the 12 wrong solves and 0 of the 2,358 correct ones. The statistic is at least 1.9×10^{-4} on every wrong solve and at most 5.8×10^{-8} on every correct one, so any threshold from 10^{-7} to 10^{-5} works. It reads one solver statistic (8 μs).
- A zero-iteration re-check at 10^{-9} tolerances flags the 12 wrong solves and 97 correct ones, and continuing from those false positives breaks four correct solves.
- An explicit KKT residual takes 73 ms and flags 137 correct solves.

Continuation at 10^{-9} from the returned basis recovered the reference optimum on all 12 wrong solves in 1.7 to 67 s, in every case less than a native solve (77 to 144 s); the fallback to a native solve never

triggered. On W2 of DEV-50K the check reduces the predictor's failures from 6 to 1 (a time limit) and its summed time ratio from 1.385 to 0.863, while the class-balanced geometric mean moves from 0.619 to 0.582. The rule was frozen on development data after the TEST-50K failures had been inspected to name their class. The TEST-50K and TEST-80K columns of Table 1 are reported without the check (the failures of COSTSTART-SHIFT there are the load-balancing wrong optima plus one solver error on TEST-80K); the TEST-20K panel and the later pass of Table 24 apply it inline.

G DEPLOYMENT PROCEDURE AND METHOD DETAILS

Algorithm 2 gives the per-LP procedure of COSTSTART at deployment. Every step, including the ones whose result is discarded, is included in the charged time. COSTSTART-SHIFT uses the action set $\mathcal{A}_1$ and always evaluates the harm gate; COSTSTART-ALG and COSTSTART-FULL use $\mathcal{A}_2$ and $\mathcal{A}_3$ and are reported with and without the harm gate. Table 12 collects the settings of every learned component and of the solver.

Algorithm 2 COSTSTART at deployment, for one LP

Require: LP $x = (A, b, c, \ell, u)$, stored after presolve; action set $\mathcal{A}$; predictor, policy ensemble, novelty and harm gates

$s, h \leftarrow$ predictor(x) ▷ column scores and LP embedding, one forward pass
$z \leftarrow$ statistics of s and of the size of x
if nearest-neighbor distance of $[h, z]$ to the training LPs $>$ threshold **then** ▷ novelty gate
    $a \leftarrow$ (shift 0, dual simplex, HiGHS' default pricing)
else
    $a \leftarrow \arg\max_{a' \in \mathcal{A}} \frac{1}{E} \sum_{e=1}^{E} \log \pi_{\theta_e}(a' \mid h, z)$ ▷ ensemble of E policies
end if
if a is native HiGHS **then**
    solve x with native HiGHS; **return**
end if
$B_0 \leftarrow$ triangular crash on $s + \delta_a \mathbf{1}_{\text{struct}}$ (Algorithm 1)
if harm gate is used **and** predicted saving of B_0 over native HiGHS $< 12\%$ **then**
    solve x with native HiGHS; **return**
end if
`setBasis`(B_0); run HiGHS with the algorithm and pricing rule of a
if sum of dual infeasibilities $> 10^{-6}(1 + |f|)$ **then** ▷ optimality check
    continue from the returned basis with primal and dual feasibility tolerances 10^{-9}
    if not optimal, solve x afresh with native HiGHS
end if

Table 12: Settings of the learned components and of the solver. Selection was always on development data or a held-out fifth of the training LPs, never on test data.

Component	Setting
Predictor	8-layer GATv2 on the variable-constraint graph, 1.8M parameters, 72-dimensional pooled embedding; fused kernel with TF32, one LP per forward pass (6.4 ms median at 20k variables on an H100)
stage 1	114,874 LPs of TRAIN-SMALL, 12 epochs (43k steps, 32 LPs per step, edge budget 400k), AdamW at 10^{-3} without weight decay, warm-up and cosine decay, gradient clip 5; loss: basis-membership cross-entropy divided by m plus $0.2\times$ auxiliary dual-tightness and bound-status terms
stage 2	3,457 LPs of TRAIN-20K, 6 passes, 48 LPs per step, AdamW at 3×10^{-4}, clip 1; checkpoint every 37 steps, chosen by paired solve time on the 283-LP selection split of DEV-20K
Decoder	entry threshold 10^{-8} of the column maximum, degree exponent $p = 0.5$, shifts $\{0, \pm 0.5, \pm 1, \pm 2, \pm 3\}$
Policies	MLP $84 \to 128 \to 128 \to \lvert\mathcal{A}\rvert$, SiLU, dropout 0.1, AdamW (weight decay 10^{-4}), batch 256 LPs, loss clip ± 3, failures at $+3$, last layer initialized to the unshifted start $\lvert\mathcal{A}_1\rvert = 9$; 7,699 training LPs; learning rate 10^{-3}, 100 epochs, 3 seeds
COSTSTART-SHIFT	
COSTSTART-ALG,	$\lvert\mathcal{A}_2\rvert = 11$, $\lvert\mathcal{A}_3\rvert = 20$; 9,349 and 9,197 training LPs; learning rate 3×10^{-3}, 150
COSTSTART-FULL	epochs, 10 seeds
Novelty gate	standardized 84-dimensional $[h, z]$; threshold 2.466, the 0.99 quantile of nearest-neighbor distances from a random fifth of training LPs to the rest
Harm gate	gradient-boosted regression trees on 73 features, Huber loss, 150 trees of depth 3, learning rate 0.05, leaves of at least 25 rows; keep the learned start if the predicted log ratio to native HiGHS is at most -0.128
Optimality check	sum of dual infeasibilities $> 10^{-6}(1 + \lvert f\rvert)$; continuation from the returned basis at primal and dual feasibility tolerances 10^{-9}, then native HiGHS if needed
Solver	HiGHS 1.13.1, one thread, presolve off, all other options at their defaults (primal and dual feasibility tolerances 10^{-7}) unless listed here. Learned starts: dual simplex (`simplex_strategy` 1) with HiGHS' default dual edge weights, or Devex forced (`simplex_dual_edge_weight_strategy` 1) for the Devex actions; primal simplex for the primal action. Native HiGHS: `simplex_strategy` 0 (choose)

H DATA, TEST SETS AND PROTOCOL

This appendix lists the datasets (Table 13), and the timing protocol that were used.

Table 13: Datasets. Sizes are requested numbers of variables. Every set is disjoint from every other by content hash, generator call and seed, and every test set except TEST-ADAPT (an earlier held-out-class split whose classes were never used in training) was generated after COSTSTART-SHIFT was frozen.

Set	Variables	LPs	Classes	Use
TRAIN-SMALL	50–2,000	114,874	37	predictor, stage 1
TRAIN-20K	15k–25k	9,511	26 (25 with test LPs)	predictor stage 2 (3,457 LPs); policies and gates
DEV-20K	15k–25k	450	25	selection and ablations; half twinned with training
selection split	15k–25k	283 of DEV-20K	22	checkpoints, epochs and designs
DEV-50K	40k–60k	454	23	optimality check; policies at larger sizes
TEST-20K	15k–25k	1,356	25	main result; pilots of 400 and 80 LPs run first
TEST-50K	40k–60k	382	25	larger instances
TEST-80K	60k–100k	81	24	larger still
TEST-VARIANT	15k–25k	296 of TEST-20K	10 variants, 6 dual forms	held-out variants and dual forms
TEST-NEWCLASS	15k–25k	234	6 unseen	transfer to new classes
TEST-ADAPT	15k–25k	616 + 125	5 unseen	few-shot adaptation, online bandit

Tiers and test sets. LPs are generated in tiers named by the requested number of variables: small (50 to 2,000), 20k (15,000 to 25,000) and 50k/80k (40,000 to 100,000). One fresh set at the 20k size is split by rule into TEST-20K (the 25 training classes with test instances) and TEST-NEWCLASS (six classes no model saw). Two fresh sets at larger sizes give TEST-50K and TEST-80K; a pre-declared rule excludes three variants whose instances do not grow with the requested size. TEST-ADAPT comes from an earlier 800-LP held-out-class panel, carved by rule into 125 adaptation and 616 evaluation LPs. DEV-20K is used for selection and ablations; about half of it has a primal or dual twin in the predictor's training data, so its levels against native HiGHS are optimistic, and we use it for paired comparisons among learned methods. Its 283-LP selection split (the W1 LPs of the original validation split, 22 classes) selected checkpoints, epochs and designs.

Timing protocol. Every method solves the same stored LP in a fresh HiGHS instance on one pinned core after a discarded warm-up solve, in a seeded random order per LP. Panels at the 20k size were run twice (TEST-NEWCLASS three times, the 80-LP W2 pilot once) and averaged within LP; panels at larger sizes ran once on exclusive nodes. Instruction counts are recorded beside every time. A solve fails if it hits the time limit, returns an error, or ends with an objective more than 10^{-5} relative from a reference solve; failures stay in the data at twice the time limit, as in PAR2. The class-balanced geometric mean averages the per-LP log ratios within each class and then across classes; intervals come from 2,000 two-level bootstrap draws (classes, then generator instances within class, so that the primal and dual forms of one instance are resampled together). Because a geometric mean can hide a few very slow LPs, we also report summed time, classes won and lost, and failures.

Generators and admission. SyntheticLPs.jl provides 46 problem classes with 130 variants in all, each in a primal and a dual form; the 20k tier covers 26 classes and 67 variants that produce LPs of that size (project selection has no test instances). Presolve runs once offline with HiGHS; the stored LP is what every method solves. Timed sets admit an LP only if the all-slack dual simplex takes at least 2,000 iterations or 1 s on it, and the same solve's time assigns the windows W1 to W3.

Machines. Panels ran on three kinds of machines: a 112-core Xeon 8480+ server for the earlier development panels, a 128-core Xeon 8592+ node with four H100s, and exclusive Xeon 8168, 8268 and 6248R nodes of a CPU partition for the solves at 40k to 100k variables, with starts and gate features prepared on the GPU node and charged from there. Ratios are compared only within a panel, where every method of an LP ran on one machine; the CPU model is recorded per row.

I FULL RESULTS, LABEL ABLATION, SEEDS AND DEPLOYMENT COST

This appendix extends the results of §6.1 and §6.2. Table 14 is the full version of Table 1. On TEST-20K, Table 15 breaks the results down by window, Table 16 by class, Table 17 lists the actions taken and Table 18 the charged overhead; Table 19 and Figure 8 give TEST-50K by class. Appendices I.1

and I.2 give the training-label ablation and the seed study, and Appendix I.3 the cost of training and of CPU-only inference.

Table 14: Full version of Table 1, with the ablation rows of COSTSTART-SHIFT and the supervised control (the predictor trained for one more pass over the 20k training data). Iterations relative to the all-slack dual simplex, charged time relative to native HiGHS; same conventions as Table 1.

	TEST-20K (1,356 LPs)			TEST-50K (382 LPs)			TEST-80K
Method	iter.	time [95% CI]	W/L	iter.	time [95% CI]	W/L	time (81)
all-slack dual simplex	1.00	1.07 [0.87, 1.30]	12/13	1.00	0.99 [0.76, 1.30]	12/13	0.96
stock HiGHS, presolve on	0.85	1.10 [1.06, 1.16]	3/22	–	–	–	–
classical crash	1.22	1.76 [1.19, 2.60]	9/16	–	–	–	–
Fan et al. (original budget)	0.50	1.10 [0.89, 1.38]	15/10	0.52	1.00 [0.78, 1.28]	15/10	0.96
Fan et al. (full budget)	0.49	1.10 [0.89, 1.35]	15/10	0.51	1.02 [0.78, 1.37]	15/10	0.96
Lu & Yan (original budget)	0.47	0.97 [0.72, 1.29]	14/11	0.46	0.83 [0.52, 1.18]	18/7	0.72
Lu & Yan (full budget)	0.47	0.94 [0.70, 1.22]	14/11	0.50	0.85 [0.56, 1.23]	19/6	0.72
predictor	0.32	0.68 [0.50, 0.91]	20/5	0.33	0.61 [0.39, 0.89]	19/6	0.60
supervised control	0.32	0.69 [0.49, 0.93]	20/5	–	–	–	–
predictor + shift policy	0.31	0.64 [0.48, 0.86]	21/4	0.34	0.60 [0.37, 0.89]	19/6	0.60
predictor + harm gate	0.30	0.61 [0.46, 0.76]	23/2	0.31	0.52 [0.35, 0.70]	23/2	0.51
COSTSTART-SHIFT (shift policy + harm gate)	0.30	0.61 [0.46, 0.76]	23/2	0.32	0.53 [0.34, 0.72]	23/2	0.52
with class-blind harm gate	0.32	0.68 [0.52, 0.85]	18/7	–	0.58 [0.40, 0.79]	18/7	0.58
COSTSTART-ALG	0.25	0.51 [0.38, 0.65]	23/2		Table 24		
with harm gate	0.26	0.54 [0.41, 0.69]	23/2		Table 24		
COSTSTART-FULL	0.28	0.49 [0.37, 0.63]	24/1		Table 24		
with harm gate	0.29	0.52 [0.39, 0.65]	23/2		Table 24		
Controls for the pricing action							
predictor, Devex on every LP	0.39	0.73 [0.56, 0.97]	18/7		–		
selector without learned start	0.88	0.86 [0.75, 0.96]	16/9		–		

Robustness of the main result. On TEST-20K, leaving any one class out moves COSTSTART-FULL within 0.46 to $0.54\times$ native HiGHS (median class 0.54), COSTSTART-ALG within 0.49 to 0.56, COSTSTART-SHIFT within 0.59 to 0.68 and the predictor within 0.64 to 0.76, against 0.90 to 1.04 for Lu & Yan and 1.04 to 1.14 for Fan et al. The most influential class for every method is set systems, on which every learned start is far faster than native HiGHS; the upper ends of these ranges are the values without it. Failures are few: 3 of 382 LPs for COSTSTART-SHIFT on TEST-50K, all wrong-objective terminations on badly scaled load-balancing LPs, against 4 time limits for native HiGHS. The optimality check, frozen on development data after these panels ran, removes all three in the later pass (Table 24); in the TEST-20K panel, where it runs inline, none of our learned methods fails. The two classes of TEST-50K that COSTSTART-SHIFT loses (Figure 8), facility location and telecom network design ($1.2\times$), are ones where HiGHS' primal simplex from a primal-feasible all-slack basis is cheap per pivot and the harm gate does not reliably predict this.

Windows and summed time on TEST-50K.

- COSTSTART-SHIFT against native HiGHS by window: W1 (164 LPs) 0.621 [0.47, 0.80], W2 (145) 0.635 [0.50, 0.80], W3 (73) 0.407. Summed time: 0.762 pooled, and 0.37, 1.03 and 0.74 by window.
- Leaving out any one class moves the pooled figure of COSTSTART-SHIFT within 0.513 to 0.612 (predictor with harm gate: 0.503 to 0.600); the median class is at 0.629. COSTSTART-SHIFT is faster than native HiGHS on 215 of 382 LPs, the predictor alone on 248.
- Failures: native HiGHS 4 (time limits), all-slack dual simplex 0, COSTSTART-SHIFT 3 (wrong objectives), predictor 5, Fan et al. 6, Lu & Yan 4 to 5 (one unbounded status). On TEST-80K COSTSTART-SHIFT fails twice (one wrong objective, one solver error).

I.1 TRAINING LABEL OF THE DECISION POLICY

The label ablation of §6.2 (Figure 7a, Table 20) trains three shift policies that differ only in the label, each relative to shift 0: log instructions (I; a failure counts as $\log(2\cdot 120\,\mathrm{s}\cdot 5\times 10^9)$), $\log(1+\text{iterations})$ (N; failure $\log(1+2\times 10^6)$), and log measured seconds under 64 concurrent workers (T; failure $\log 240$). Everything else follows the shift policy's recipe (nine actions, hidden width 128, dropout 0.1, AdamW at 10^{-3}, 100 epochs, batch 256, clip 3, seeds 0 to 2, epoch chosen on a random fifth of the training LPs). The labels have standard deviations 0.350, 0.329 and 0.351 and within-LP Spearman correlations of 0.98 (I and T), 0.85 (I and N) and 0.83 (N and T). The charged panel has the 381 W1 LPs of DEV-20K; starts are prepared once and are identical for the three policies, and the

Table 15: Charged time on TEST-20K by window of native HiGHS' dual simplex time. Time: class-balanced geometric-mean ratio to native HiGHS with 95% bootstrap interval (2,000 draws over classes, then generator instances) and classes won/lost; iterations: class-balanced geometric-mean ratio to the all-slack dual simplex of one plus the iteration count; summed: total charged time over that of native HiGHS; failures count at twice the window's time limit (120, 600 and 3,600 s). Numbers are averaged within LP over two timing repetitions.

Method	time [95% CI]	W/L	iterations	summed	failures
W1 ($\leq$ 10 s): 1,052 LPs, 24 classes					
native HiGHS (reference)	1.00	–	0.86	1.00	0
all-slack dual	1.04 [0.85, 1.29]	10/14	1.00	0.91	0
stock HiGHS, presolve on	1.13 [1.09, 1.18]	1/23	0.86	1.03	0
classical crash	1.84 [1.28, 2.72]	9/15	1.27	2.65	1
Fan et al.	1.09 [0.88, 1.37]	14/10	0.46	1.20	0
Lu & Yan	1.06 [0.88, 1.29]	13/11	0.48	1.19	0
predictor	0.74 [0.59, 0.93]	19/5	0.31	0.76	0
COSTSTART-SHIFT	0.67 [0.56, 0.80]	22/2	0.29	0.57	0
COSTSTART-ALG	0.60 [0.50, 0.72]	22/2	0.25	0.52	0
COSTSTART-FULL	0.55 [0.46, 0.66]	23/1	0.29	0.58	0
W2 (10–120 s): 267 LPs, 12 classes					
native HiGHS (reference)	1.00	–	0.77	1.00	0
all-slack dual	1.38 [1.02, 1.94]	5/7	1.00	1.08	0
stock HiGHS, presolve on	1.00 [0.93, 1.05]	3/9	0.76	0.98	0
classical crash	1.62 [0.91, 3.13]	4/8	1.18	1.55	0
Fan et al.	1.21 [0.94, 1.61]	6/6	0.80	1.04	0
Lu & Yan	0.70 [0.28, 1.35]	7/5	0.63	0.82	0
predictor	0.53 [0.20, 1.01]	10/2	0.48	0.72	0
COSTSTART-SHIFT	0.46 [0.18, 0.80]	8/4	0.44	0.61	0
COSTSTART-ALG	0.33 [0.15, 0.61]	11/1	0.32	0.45	0
COSTSTART-FULL	0.36 [0.15, 0.64]	11/1	0.35	0.51	0
W3 ($>$ 120 s): 37 LPs, 5 classes					
native HiGHS (reference)	1.00	–	0.71	1.00	0
all-slack dual	1.85 [0.94, 6.23]	3/2	1.00	1.58	0
stock HiGHS, presolve on	0.80 [0.49, 1.14]	3/2	0.54	0.95	0
classical crash	0.78 [0.30, 4.16]	4/1	0.49	0.71	0
Fan et al.	1.09 [0.38, 4.48]	4/1	0.64	0.99	0
Lu & Yan	0.79 [0.26, 3.96]	4/1	0.53	0.93	0
predictor	0.80 [0.25, 4.31]	4/1	0.47	0.98	0
COSTSTART-SHIFT	0.82 [0.33, 2.70]	3/2	0.53	0.81	0
COSTSTART-ALG	0.22 [0.12, 0.42]	5/0	0.18	0.23	0
COSTSTART-FULL	0.33 [0.12, 1.29]	4/1	0.21	0.30	0
all windows: 1,356 LPs, 25 classes					
native HiGHS (reference)	1.00	–	0.85	1.00	0
all-slack dual	1.07 [0.87, 1.30]	12/13	1.00	1.26	0
stock HiGHS, presolve on	1.10 [1.06, 1.16]	3/22	0.85	0.98	0
classical crash	1.76 [1.19, 2.60]	9/16	1.22	1.34	1
Fan et al.	1.10 [0.89, 1.35]	15/10	0.49	1.04	0
Lu & Yan	0.94 [0.70, 1.22]	14/11	0.47	0.90	0
predictor	0.68 [0.50, 0.91]	20/5	0.32	0.83	0
COSTSTART-SHIFT	0.61 [0.46, 0.76]	23/2	0.30	0.68	0
COSTSTART-ALG	0.51 [0.38, 0.65]	23/2	0.25	0.37	0
COSTSTART-FULL	0.49 [0.37, 0.63]	24/1	0.28	0.44	0

Table 16: Per-class charged time on TEST-20K relative to native HiGHS: geometric mean over the class's LPs of the per-LP ratio, all three windows, failures at twice the time limit, lowest entry per row in bold. The last rows give the class-balanced geometric mean over the 25 classes, the classes won/lost against native HiGHS, and the failures. Every learned arm runs the optimality check inline; numbers are averaged within LP over two timing repetitions.

		baselines				COSTSTART		
Class	LPs	all-slack	Fan et al.	Lu & Yan	predictor	Shift	Alg	Full
bin packing	50	**0.46**	0.96	1.00	0.83	0.54	0.50	0.74
crop planning	11	1.05	0.74	0.80	**0.68**	0.92	0.70	0.69
energy	104	0.74	0.79	0.81	0.58	0.64	0.53	**0.50**
facility location	37	2.92	2.07	1.63	1.39	1.19	1.02	**0.91**
graph optimization	156	1.26	1.09	1.12	0.76	0.69	**0.36**	0.63
hub location	195	2.54	2.85	2.60	2.07	0.96	0.91	**0.77**
inventory	77	0.48	0.39	1.07	0.38	0.42	0.38	**0.32**
inverse optimization	143	1.00	0.73	0.56	0.38	0.36	0.28	**0.27**
knapsack	39	1.00	1.51	0.99	0.69	0.69	0.67	**0.50**
land use	13	1.02	1.66	1.56	0.80	0.91	0.80	**0.59**
load balancing	52	0.98	1.42	1.85	0.60	**0.37**	0.38	0.42
maritime inventory routing	26	1.00	0.97	0.87	0.78	**0.75**	**0.75**	0.86
multi-commodity flow	26	1.00	0.93	1.00	0.81	0.99	0.58	**0.39**
nurse scheduling	26	0.99	1.30	1.06	0.81	**0.71**	**0.71**	0.73
operating room scheduling	65	1.00	0.84	0.80	0.70	0.76	**0.61**	**0.61**
revenue management	39	1.01	0.83	0.89	0.76	0.77	0.64	**0.54**
scheduling	6	0.82	0.90	0.70	0.36	0.33	**0.30**	0.37
set systems	26	1.00	0.80	0.08	**0.05**	0.06	**0.05**	**0.05**
stochastic programming	26	1.01	0.81	0.71	0.52	0.69	0.53	**0.37**
supply chain	46	2.37	1.72	1.79	1.05	0.92	0.86	**0.85**
telecom network design	25	1.85	3.79	2.58	3.76	**1.42**	1.68	1.69
transportation	13	0.57	0.53	0.42	0.52	0.52	0.31	**0.29**
traveling salesman	116	2.59	2.77	2.24	1.68	0.95	0.92	**0.86**
unit commitment	13	0.99	0.83	0.56	0.54	0.55	**0.20**	**0.20**
vehicle routing	26	0.57	0.72	0.67	0.43	0.38	0.44	**0.34**
class-balanced mean	1,356	1.07	1.10	0.94	0.68	0.61	0.51	**0.49**
classes won/lost		12/13	15/10	14/11	20/5	23/2	23/2	24/1
failures		0	0	0	0	0	0	0

Table 17: Actions taken on TEST-20K, as shares of LPs (%). Top: action groups by pricing rule and shift δ; for COSTSTART-SHIFT the native column is the harm gate's fallback, and for all three systems $\delta = 0$ under DSE includes the LPs that the novelty gate returns to the unshifted start. Bottom: the shift chosen by the shift policy of COSTSTART-SHIFT before the harm gate, and the gate's fallback rate. Numbers are averaged within LP over two timing repetitions; actions and gate decisions are the same in every repetition.

					DSE pricing			Devex pricing		
System	window	LPs	native HiGHS	primal	$\delta < 0$	$\delta = 0$	$\delta > 0$	$\delta < 0$	$\delta = 0$	$\delta > 0$
COSTSTART-FULL	W1	1,052	8.1	7.2	0.0	12.9	16.7	41.8	0.0	13.2
	W2	267	16.1	25.8	0.0	7.9	27.3	20.2	0.0	2.6
	W3	37	0.0	29.7	0.0	0.0	24.3	29.7	0.0	16.2
	all	1,356	9.4	11.5	0.0	11.6	19.0	37.2	0.0	11.2
COSTSTART-ALG	W1	1,052	13.9	12.6	8.6	32.1	32.8	–	–	–
	W2	267	2.6	49.1	12.7	6.7	28.8	–	–	–
	W3	37	0.0	59.5	0.0	0.0	40.5	–	–	–
	all	1,356	11.3	21.1	9.1	26.3	32.2	–	–	–
COSTSTART-SHIFT	W1	1,052	27.8	–	18.1	16.3	37.8	–	–	–
	W2	267	44.2	–	11.6	27.7	16.5	–	–	–
	W3	37	13.5	–	0.0	83.8	2.7	–	–	–
	all	1,356	30.6	–	16.3	20.4	32.7	–	–	–

		shift δ chosen by the shift policy									harm gate
window	LPs	-3	-2	-1	-0.5	0	$+0.5$	$+1$	$+2$	$+3$	to native
W1	1,052	20.2	0.0	0.6	3.7	27.9	1.5	0.0	0.0	46.1	27.8
W2	267	12.0	0.0	0.0	0.0	49.8	0.0	0.0	0.0	38.2	44.2
W3	37	0.0	0.0	0.0	0.0	97.3	0.0	0.0	0.0	2.7	13.5
all	1,356	18.0	0.0	0.4	2.9	34.1	1.2	0.0	0.0	43.4	30.6

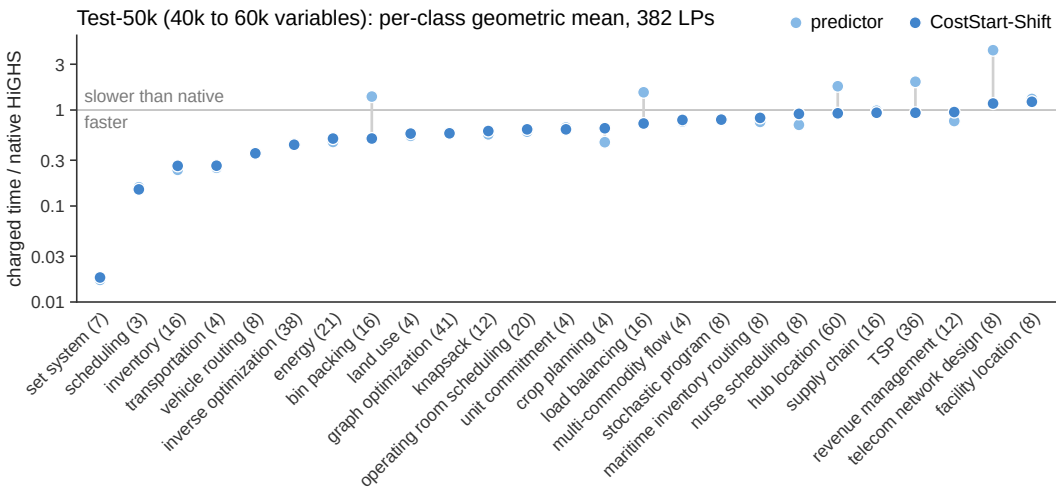

Figure 8: Per-class charged time on TEST-50K (ratio to native HiGHS, geometric mean over the class's LPs, all three windows; LP counts in parentheses). The predictor loses six classes; the shift policy and harm gate of COSTSTART-SHIFT turn four of them into wins, and the two remaining losses are classes where HiGHS' primal simplex from a primal-feasible all-slack basis is hard to beat. Table 19 gives the same panel with more methods.

Table 18: Charged overhead per LP on TEST-20K: median wall time in milliseconds and median share of the LP's total charged time (%), by window. Decode includes the policy (with its novelty gate) and the crash; the harm gate (features and model) is charged on every LP, including those it hands to native HiGHS, which then need no setBasis. Total is the sum of the five components. Numbers are averaged within LP over two timing repetitions.

	W1		W2		W3		all	
Component	ms	%	ms	%	ms	%	ms	%
predictor								
features	9.1	1.28	9.0	0.05	10.4	0.01	9.1	0.89
inference	6.7	0.99	8.9	0.05	11.1	0.01	7.0	0.69
decode	8.6	1.31	10.6	0.06	19.0	0.01	9.0	0.90
setBasis	5.5	0.70	9.6	0.05	8.8	0.01	6.3	0.49
harm gate	–	–	–	–	–	–	–	–
total overhead	30.1	4.38	39.4	0.21	53.6	0.04	32.5	3.19
COSTSTART-SHIFT								
features	8.8	1.50	9.0	0.07	10.4	0.01	8.9	1.07
inference	6.7	1.30	8.9	0.07	11.1	0.01	7.1	0.88
decode	19.7	3.84	23.1	0.19	39.9	0.02	20.5	2.61
setBasis	2.3	0.46	4.0	0.01	8.7	0.01	2.8	0.27
harm gate	10.0	1.91	11.0	0.09	16.5	0.01	10.2	1.23
total overhead	49.8	9.38	56.9	0.44	94.2	0.07	52.0	6.34
COSTSTART-ALG								
features	8.7	1.67	9.0	0.11	10.3	0.05	8.9	1.16
inference	6.7	1.45	9.0	0.11	11.0	0.03	7.1	0.94
decode	11.8	2.43	13.1	0.16	21.3	0.05	12.1	1.63
setBasis	3.9	0.66	8.7	0.09	8.9	0.06	5.2	0.50
harm gate	–	–	–	–	–	–	–	–
total overhead	31.9	6.44	41.8	0.49	59.2	0.19	34.0	4.60
COSTSTART-FULL								
features	8.7	1.98	9.0	0.07	10.2	0.02	8.9	1.31
inference	6.7	1.55	9.0	0.07	11.1	0.02	7.1	1.03
decode	11.6	2.83	13.0	0.11	21.3	0.04	11.9	1.72
setBasis	4.6	0.87	7.7	0.04	9.5	0.02	5.5	0.66
harm gate	–	–	–	–	–	–	–	–
total overhead	32.4	7.59	38.7	0.30	57.8	0.10	34.2	4.82

solves run on one exclusive node with three repetitions in seeded order, failures at 240 s (PAR2), and no failures.

Split by Eq. (1), the gain of the instruction policy over the iteration policy lies entirely in the time per iteration (the split is in Appendix E): on the offline label tables, the iteration policy's choices sit 4%

Table 19: Per-class charged time on TEST-50K relative to native HiGHS: geometric mean over the class's LPs of the per-LP ratio, all three windows, failures at twice the time limit, lowest entry per row in bold. This panel ran before the optimality check existed, and Fan et al. comes from a later paired pass on the same LPs with its own native HiGHS solves. The last rows give the class-balanced geometric mean, the classes won/lost against native HiGHS, and the failures.

		baselines				
Class	LPs	all-slack	Fan et al.	Lu & Yan	predictor	COSTSTART-SHIFT
bin packing	16	**0.34**	0.76	0.77	1.38	0.50
crop planning	4	1.04	0.58	0.63	**0.46**	0.65
energy	21	0.56	0.54	0.64	**0.47**	0.50
facility location	8	2.21	2.27	1.69	1.30	**1.22**
graph optimization	41	1.15	0.95	0.92	0.58	**0.57**
hub location	60	2.52	2.65	2.41	1.77	**0.92**
inventory	16	0.38	0.27	0.71	**0.24**	0.26
inverse optimization	38	1.00	0.88	0.71	**0.44**	**0.44**
knapsack	12	0.99	0.95	0.99	**0.56**	0.60
land use	4	1.01	1.14	0.99	**0.54**	0.57
load balancing	16	1.17	3.24	2.80	1.53	**0.72**
maritime inventory routing	8	1.00	0.91	0.93	**0.75**	0.83
multi-commodity flow	4	1.04	1.07	0.91	**0.76**	0.79
nurse scheduling	8	0.78	1.13	0.82	**0.70**	0.91
operating room scheduling	20	1.01	0.73	0.69	**0.59**	0.63
revenue management	12	0.98	0.79	0.85	**0.77**	0.95
scheduling	3	0.45	0.49	0.41	0.16	**0.15**
set systems	7	1.00	0.75	0.03	**0.02**	**0.02**
stochastic programming	8	1.03	1.09	0.96	**0.78**	0.79
supply chain	16	2.85	2.06	1.80	0.98	**0.94**
telecom network design	8	2.42	4.03	4.94	4.20	**1.17**
transportation	4	0.31	0.39	**0.25**	**0.25**	0.26
traveling salesman	36	3.36	3.22	2.82	1.97	**0.94**
unit commitment	4	0.99	0.89	0.69	0.66	**0.63**
vehicle routing	8	0.47	0.69	0.56	0.36	**0.35**
class-balanced mean	382	0.99	1.02	0.85	0.61	**0.53**
classes won/lost		12/13	15/10	19/6	19/6	23/2
failures		0	7	5	5	3

Table 20: Training-label ablation on the 381 W1 LPs of DEV-20K: three shift policies that differ only in the label (I: instructions, N: iterations, T: wall time under load). Charged time, iterations and summed time relative to the predictor (class-balanced, three timing repetitions), then paired contrasts between the policies.

Label	time [95% CI]	iterations	summed time
instructions (I)	0.927 [0.877, 0.978]	0.943	0.918
iterations (N)	0.953 [0.899, 1.008]	0.932	0.947
wall time (T)	0.951 [0.907, 0.993]	0.955	0.963
I relative to N	0.973 [0.934, 1.010]	by repetition 0.9735, 0.9735, 0.9719	
on the 202 LPs without a twin in training	0.939 [0.886, 0.984]		
I relative to T	0.974 [0.951, 0.996]		

above their LP's mean work per iteration and the instruction policy's 3% below it. The offline tables show the same without timing noise:

- On the selection split of DEV-20K, the shift with the fewest iterations costs $1.06\times$ [1.03, 1.09] the instruction-optimal one, and on 38% of LPs choosing by iterations costs at least 1% more than the instruction-optimal shift.
- On the same split, the cheapest action in instructions among the iteration minimizers costs $1.058\times$ [1.029, 1.094] the instruction oracle over the nine shifts and $1.096\times$ [1.051, 1.159] over the 42 decoder starts; on the 7,699 training LPs, $1.048\times$ [1.028, 1.072]. At least 1% instruction regret is unavoidable for any iteration-optimal choice on 33 to 55% of LPs.

Label ablation for the full policy. The shift actions leave little room for the label to matter (the offline gap above is about 6%). The 20 actions of COSTSTART-FULL include actions that change the time per iteration, so we also trained the COSTSTART-FULL policy on log iteration counts relative to the unshifted start (clipped at ± 3, failures at $+3$), with its exact recipe (ten seeds, 150 epochs, the same held-out fifth of the training LPs). An overlap label was not possible, since the cost tables record no per-action overlap. Table 21 scores each policy's chosen actions in instructions relative to the predictor's unshifted start. Trained on iterations, the policy removes 15 to 18% more iterations but routes 5 to 15% of LPs to Devex instead of 29 to 60%, and primal simplex instead; its iterations then cost 1.2 to 1.7 times as much, for 40% more instructions on short solves and a tie on longer ones. This is the 20-action counterpart of the shift-policy result, and the per-iteration term is where the instruction label earns its gain.

Table 21: Training label of the COSTSTART-FULL policy, offline on held-out training LPs: instructions of the chosen actions relative to the predictor's unshifted start, split into iterations and work per iteration, and the share of LPs routed to Devex and to primal simplex. The selection split was read during the policy's design.

LPs	Label	instructions [95% CI]	iterations	per iteration	Devex	primal
held-out, W1 (1,531)	instructions	0.669 [0.589, 0.751]	0.892	0.749	51%	8%
	iterations	0.937 [0.817, 1.103]	0.759	1.234	12%	36%
held-out, W2 (316)	instructions	0.668 [0.526, 0.826]	0.729	0.916	29%	20%
	iterations	0.679 [0.489, 0.998]	0.601	1.129	5%	51%
selection split of DEV-20K (275)	instructions	0.682 [0.570, 0.797]	0.868	0.786	60%	9%
	iterations	0.960 [0.798, 1.170]	0.723	1.327	15%	46%

Iteration label relative to instruction label, paired (instructions; iterations; per iteration): held-out W1 1.402 [1.183, 1.727]; 0.851 [0.769, 0.933]; 1.647 [1.364, 2.040]. Held-out W2 1.016 [0.767, 1.335]; 0.824 [0.684, 0.950]; 1.233 [1.002, 1.649]. Selection split 1.408 [1.152, 1.771]; 0.834 [0.742, 0.917]; 1.689 [1.352, 2.152].

I.2 SEEDS

Predictor seeds. A second seed of stage 2 matches the first within 0.01 in overlap, precision and paired time at matched steps (0.873 vs. 0.881 against the model after stage 1, at step 296). A second seed of stage 1 at 2 epochs reaches overlap 0.7575 and precision 0.7248, against 0.7608 and 0.7194. A replicate of stage 2 (from the same stage-1 initialization, on the deduplicated training set with a new seed), with its own action costs and shift policy, on DEV-20K: its predictor takes $1.065\times$ [1.01, 1.13] the deployed predictor's instructions, its shift policy $0.890\times$ [0.83, 0.95] its own predictor's, and the replicate system $1.014\times$ the deployed system in instructions and $1.007\times$ in time.

Full-pipeline seeds. Stage 1, stage 2 (on the deduplicated training set), the nine-shift cost table and the gated shift policy were retrained with seeds 101, 202 and 303. On the 283-LP selection split of DEV-20K, which also selected their checkpoints and epochs, the three predictors reach overlap 0.785 to 0.793 and 0.822 to $0.840\times$ native HiGHS in charged time (geometric mean 0.830, standard deviation of the log ratio 0.011). With their own gated shift policies they take 0.733, 0.770 and $0.761\times$ (mean 0.755, standard deviation 0.026), with iterations $0.513\times$ and instructions $0.702\times$, against $0.728\times$ for the deployed predictor with its policy. Offline, the three shift policies take 0.849, 0.903 and $0.904\times$ their own predictor's instructions, and a policy fitted to another seed's predictor gains little ($0.977\times$), so policies are specific to their predictor. Single policy seeds give 0.896 ± 0.013 (5 seeds), and they are portable across hardware: six single policies re-costed on a different CPU model agree to 0.001.

On TEST-20K (Table 22; all windows, both repetitions averaged, check inline), the pipelines of seeds 202 and 303 ran as paired passes anchored on native HiGHS, the deployed predictor and COSTSTART-SHIFT; seed 101 is part of the main panel. The seed spread of the full pipeline on the test set (± 0.003) is two orders of magnitude below the width of the class and LP intervals.

Table 22: Full-pipeline seeds on TEST-20K: charged time relative to native HiGHS (class-balanced, two repetitions averaged, check inline), and the shift policy's time relative to its own predictor. The deployed predictor, trained before the audit on the original split, takes 0.683, and COSTSTART-SHIFT 0.614.

	seed 101	seed 202	seed 303
predictor	0.714	0.711	0.708
+ own shift policy	0.642	0.647	0.649
+ own shift policy, check and harm gate	0.621	0.616	0.619
shift policy relative to its predictor	0.90	0.91 [0.85, 0.96]	0.92 [0.87, 0.96]

I.3 DEPLOYMENT COST

Inference on the CPU. All panels run the predictor's forward pass on a GPU and charge its time. On CPU cores alone, an optimized forward pass (fused message passing, bf16 matrix units, compiled kernels) takes a median 93 ms per LP of the 20k tier on 16 cores, against 8 ms on the GPU, with starts whose solve cost is within 1 to 2% of the GPU starts. Charged, this makes COSTSTART-SHIFT 29 to 30% slower on solves under 10 s (on the 400-LP W1 pilot of TEST-20K, from $0.70\times$ to $0.90\times$ native HiGHS), 3 to 5% slower on longer solves, and 14% slower on TEST-50K pooled; a CPU deployment would need a forward pass of about 30 ms to keep the loss on short solves under 5%.

Training cost. Predictor: 8.8 H100-hours for stage 1 plus 2.5 hours for stage 2, including validation solves. Action costs for the shift policy: the 131,706-solve table took about 60 core-hours, of which the 7,699 training LPs use 69k solves; for the twenty actions of COSTSTART-FULL, 184k solves and 373 core-hours (Appendix E). Labels of the harm gate: 7.7 core-hours. Published methods: Fan et al. 1.5 GPU-hours for stage 1 at 0.43 times our stage-1 samples; Lu & Yan 2.5 GPU-hours plus 86 CPU-hours of late-path labels at 0.33 times; both second stages at our stage-2 budget, and Lu & Yan additionally at 12 stage-1 epochs. Test generation: an estimated 58 CPU-hours for the 20k test sets and about 100 for the 50k and 80k sets.

J GENERALIZATION: LARGER LPS, HELD-OUT VARIANTS AND NEW CLASSES

This appendix gives the full versions of Table 2 (Table 24 for part a, Table 23 for part b) and the details behind §6.3: larger LPs, held-out variants and dual forms, unseen classes, and adaptation to new classes (Appendix J.1).

Larger LPs. The novelty gate sends most LPs of the larger sets to the default action (208 of 309 on TEST-50K, 61 of 63 on TEST-80K), where the policy never gets to choose native HiGHS, which is why the harm gate carries most of the gain there. On DEV-50K, without any training data at that size, COSTSTART-FULL and COSTSTART-ALG are within a few percent of each other, and the predictor with Devex on every LP loses on W2 (Table 9). On the test sets (Table 24), COSTSTART-FULL with the harm gate takes $0.98\times$ the time of COSTSTART-ALG with the harm gate on TEST-50K and $0.99\times$ on TEST-80K, while Devex on every LP gives no gain over the predictor. The later pass

Table 23: Full version of Table 2b. Charged-time ratios, class-balanced, with 95% intervals. TEST-ADAPT: the shift policy adapted with n labeled LPs per new class, against the unadapted policy, whose novelty gate falls back to the predictor's start on these classes. TEST-VARIANT is a subset of the TEST-20K panel.

Method	vs. native HiGHS	vs. reference
TEST-NEWCLASS: 234 LPs, 6 classes absent from training (reference: predictor)		
predictor	1.42 [0.95, 2.30]	–
COSTSTART-SHIFT, harm gate off (policy abstains on 234/234)	1.42 [0.96, 2.32]	1.00
COSTSTART-SHIFT	1.21 [0.97, 1.62]	0.85 [0.62, 1.08]
Fan et al. (full budget)	1.37 [0.96, 2.09]	–
Lu & Yan (full budget)	0.94 [0.49, 1.70]	–
classical crash	0.67 [0.30, 1.18]	–
TEST-ADAPT: 616 LPs, 5 classes absent from training (reference: unadapted policy = predictor's start)		
unadapted policy	1.14 [0.82, 1.68]	1.00
COSTSTART-SHIFT (with harm gate)	1.05 [0.94, 1.18]	0.93
adapted with $n = 10$ labeled LPs per class	1.05 [0.74, 1.49]	0.93 [0.85, 1.00]
adapted with $n = 25$ labeled LPs per class	1.04 [0.73, 1.48]	0.91 [0.84, 0.98]
online bandit over strategies, no labels	0.93 [0.79, 1.09]	0.82 [0.64, 0.97]
TEST-VARIANT: 296 LPs, 10 held-out variants and 6 held-out dual forms (reference: deployed predictor)		
predictor trained without these variants and forms	0.71 [0.33, 1.41]	1.33 [1.13, 1.64]
with its own shift policy, the deployed harm gate and check	0.56 [0.30, 0.86]	1.05 [0.67, 1.50]
deployed COSTSTART-FULL (trained on these variants and forms)	0.38 [0.19, 0.63]	0.71 [0.49, 0.91]

Table 24: Full version of Table 2a: a later pass over the W1 and W2 LPs of TEST-50K (309 of 382 LPs) and TEST-80K (63 of 81). Charged time relative to native HiGHS, class-balanced, with 95% intervals, failures in parentheses, and summed time. Every learned method runs the optimality check inline; gate evaluation is charged. The pass omits W3, where the first pass found the largest gains. An earlier pass without COSTSTART-FULL gave $0.57\times$ and $0.69\times$ for COSTSTART-ALG with the harm gate and $0.60\times$ and $0.72\times$ for COSTSTART-SHIFT.

	TEST-50K (309)		TEST-80K (63)	
Method	time [95% CI]	summed	time [95% CI]	summed
all-slack dual simplex	0.94 [0.72, 1.22]	0.87	0.90 [0.63, 1.28]	1.13
native HiGHS with Devex	1.06 [0.89, 1.34] (6)	1.63	1.07 [0.93, 1.26]	1.28
predictor	0.65 [0.48, 0.87] (1)	0.72	0.83 [0.54, 1.27] (1)	1.70
predictor, Devex on every LP	0.66 [0.46, 0.96]	0.89	0.78 [0.48, 1.30] (1)	2.29
COSTSTART-SHIFT	0.58 [0.46, 0.72]	0.50	0.67 [0.48, 0.88]	0.84
with class-blind harm gate	0.67 [0.52, 0.83]	0.74	0.75 [0.55, 1.01]	0.95
COSTSTART-ALG	0.60 [0.45, 0.78] (2)	0.61	0.83 [0.53, 1.28] (1)	1.71
+ harm gate	0.55 [0.43, 0.69]	0.47	0.66 [0.48, 0.86]	0.83
+ class-blind harm gate	0.63 [0.49, 0.79]	0.69	0.74 [0.53, 1.01]	0.95
COSTSTART-FULL	0.56 [0.43, 0.74] (1)	0.60	0.81 [0.53, 1.25] (1)	1.71
+ harm gate	0.54 [0.42, 0.69]	0.49	0.65 [0.47, 0.85]	0.84
+ class-blind harm gate	0.60 [0.48, 0.76] (1)	0.70	0.73 [0.53, 0.99]	0.96

covers only W1 and W2 and so omits the longest solves, on which the first pass found the largest gains (COSTSTART-SHIFT at $0.407\times$ native HiGHS on W3 of TEST-50K; Appendix I); its ratios are therefore higher than those of Table 1.

Held-out variants and dual forms on development data. Between a new instance of a training class and a new class lies a third case: a new variant of a training class, or the dual form of one. On a development panel (357 LPs of 10 held-out variants and 6 held-out dual forms, with 330 control LPs of matching variants and forms; instruction counts, fully charged) we compare a full pipeline trained without these variants and forms to the deployed one by a difference in differences: the ratio of the two pipelines' costs on the held-out LPs divided by the same ratio on the controls. It is 1.23 [0.99, 1.55] over all 16, 1.06 [0.86, 1.29] for the dual forms, which retain about 93% of the predictor's gain over native HiGHS, and 1.34 [1.02, 1.93] for the variants, which retain about 28%. The pipeline trained without the variants still beats native HiGHS on 7 of the 10 and fails badly on three (a load-balancing variant at $5.1\times$ the deployed pipeline's cost); the novelty gate abstains on 98.7% of the held-out LPs, so the policy abstains there. On TEST-VARIANT (Table 23), the predictor trained without the held-out variants and forms takes $1.33\times$ the deployed predictor's time, and on the other 1,060 LPs of

TEST-20K $1.04\times$, so the loss is specific to the held-out variants and forms. Its own shift policy adds nothing there ($0.72\times$ native HiGHS without the harm gate, against $0.71\times$ for its predictor alone); the gain to $0.56\times$ comes from the deployed harm gate, which was fit on TRAIN-20K and so did see these variants and forms.

Unseen classes. On the six classes of TEST-NEWCLASS, every learned start of ours is slower than native HiGHS (the predictor by $1.4\times$; job-shop scheduling $2.7\times$, diet problems $3\times$), while the classical crash is faster ($0.67\times$). Fan et al. at full budget is at $1.37\times$; Lu & Yan is near parity only because on the diet problems its scores nearly reduce to the crash's ordering. The novelty gate flags every LP but falls back to the predictor's unshifted start, which leave-one-class-out validation on the training classes had preferred over native HiGHS (1.00 against $1.45\times$); on new classes the order reverses ($1.42\times$ against 1.00). The harm gate recovers part of the loss but leaves COSTSTART-SHIFT at $1.21\times$ native HiGHS, because it too was fit on the training classes. Held-out classes were the pre-specified transfer test; after this negative result we added the few-shot study and the held-out-variant pipeline, and we report all three.

Per-class results on TEST-NEWCLASS. The predictor against native HiGHS on W1 and W2: blending 0.87, container loading 0.88, process planning 1.05, resilient network design 1.22, job-shop scheduling 2.74, diet problem 3.43. W3 holds only diet problems (35 LPs), where the predictor takes $3.0\times$ native HiGHS, the classical crash $0.09\times$, Lu & Yan's original-budget start $0.10\times$ with iteration counts identical to the crash's (its scores reduce to the crash ordering there), and its full-budget start $0.20\times$. The harm gate falls back on 45% of TEST-NEWCLASS LPs (51% on W1 and W2); it recovers job-shop scheduling ($2.74 \to 1.27$) but hurts container loading ($0.88 \to 1.07$) and worsens summed time on W1 and W2 ($1.01 \to 1.10$). On the 800-LP held-out-class panel from which TEST-ADAPT was carved, the picture is the same (predictor 1.169, with harm gate 1.064).

J.1 ADAPTATION TO NEW CLASSES

We measured two ways to adapt to a new class from the solver's own cost: labeling a few LPs of the class, and an online bandit that needs no labels. Figure 9 summarizes the labeled route.

Few-shot adaptation with labels. With n labeled LPs of a new class (every action solved once on each), the shift policy is retrained on the training data plus the n new LPs, replicated to 10% of the rows. In replay on the 7,699 training LPs, holding out one class at a time over three stream orders (Table 25), ten labeled LPs turn the harmful unadapted policy into a gain, and 50 bring it near the in-class reference of 0.929. Excluding LPs whose generator-instance cluster appears among the labeled LPs changes these numbers by less than 0.01. Charged for acquisition, a stream pays 1.02 at $n = 10$ and 1.28 at $n = 50$, so the option pays off only for streams of thousands of LPs per class.

Table 25: Few-shot replay on the training LPs, one class held out at a time (three stream orders): instructions relative to shift 0 on the rest of the held-out class's stream after adapting with n labeled LPs. At $n = 200$ only the 13 classes with at least 220 LPs qualify. The in-class reference is 0.929.

n	0	10	25	50	100	200
13 classes with at least 220 LPs	1.098	0.965	0.952	0.946	0.934	0.924
all classes with enough LPs	1.076	0.944	0.932	0.929	0.912	0.924

Charged result on TEST-ADAPT. With 25 labeled LPs per class, the adapted policy with its novelty gate (whose reference set now includes the labeled LPs) takes $0.913\times$ [0.842, 0.980] the time of the unadapted policy on the rest of the class; the unadapted policy's gate abstains on these classes, so this reference is the predictor's start (second repetition 0.913, third on a quiet node 0.914), with instructions $0.920\times$ and iterations $0.998\times$, and is worse on none of the five classes (container loading 0.83, job-shop scheduling 0.84, process planning 0.93, blending 0.99, resilient network design 1.00), which meets the criterion. With $n = 10$ it takes $0.925\times$ [0.847, 0.995]; without the novelty gate, the adapted policy takes $0.905\times$; against COSTSTART-SHIFT, which also has the harm gate, it takes $0.994\times$. Against native HiGHS it takes $1.038\times$ [0.733, 1.482], so it recovers most of the policy's in-distribution gain but does not win. Labeling costs 10.2 times the plain solves of the labeled LPs, so adaptation breaks even at about 4,900 LPs per class ($n = 25$) or 2,700 ($n = 10$).

Labels that do not work.

- Hindsight pseudo-labels from the solved LP (distance of each action's start to the final basis, structural recall, count gaps, bound flips) rank actions poorly: the median per-LP Spearman correlation with the true costs is 0.00 to 0.26. The best of them, distance to $B^\star$, picks actions worth $0.96\times$ shift 0 against the cost oracle's $0.79\times$; this is the overlap-oracle selector of §6.2. Few-shot adaptation on such labels gives 1.01 to 1.02.
- A continual online variant on such labels gives 1.31, charged.

Online bandit without labels. The bandit keeps one learner per class stream with 13 arms: the nine shifts with dual simplex, native HiGHS with and without its choice of algorithm, the unshifted start with primal simplex, and the classical crash. Its context is 16 principal components of the penultimate features of a shift policy trained on TRAIN-20K. A Bayesian linear model of each arm's log cost relative to the unshifted start, with a prior centered on that start and a prior covariance equal to the between-class covariance of the 25 training classes, is updated from the observed instruction count of the arm played. Arms are chosen by Thompson sampling after 20 warm-up LPs at the unshifted start, under a conservative budget that plays a nondefault arm only while an upper confidence bound on the summed relative cost of all such plays stays below 5% of the stream so far; every stage is charged. Results:

- Development replay on the training LPs (instructions relative to shift 0): a noncontextual per-class bandit is limited by information (full-information follow-the-leader 0.961; realistic feedback 1.002); a contextual bandit on the frozen shift policy's features reaches 0.975 [0.942, 0.999] over full streams at zero acquisition cost, with gains arriving after about 200 LPs. The replay puts the break-even at roughly 125 LPs per class.
- TEST-ADAPT (the joint bandit over shifts and strategy arms, frozen on DEV-20K, run on the 616 LPs as real charged streams in five orders): $0.885\times$ [0.770, 1.013] the frozen system's time ($0.864\times$ in instructions), $0.820\times$ [0.637, 0.968] the predictor's and $0.928\times$ [0.793, 1.091] that of native HiGHS, over streams of 64 to 149 LPs per class; $1.03\times$ over the first 50 LPs of a stream and $0.85\times$ over the last 100; worst class process planning ($1.11\times$). A bandit over shifts alone gives $1.03\times$, so the held-out loss comes from the choice of strategy rather than of shift.
- TEST-NEWCLASS (supplementary; streams of 26 to 91 LPs per class): negative. The joint bandit ends at $0.966\times$ the predictor's time and $1.129\times$ that of the frozen system, because the streams end before it reaches native HiGHS' large wins on job-shop scheduling and diet problems (about $0.37\times$). For the diet problem, two of the three stream orders are included.

Pre-registration and deviations.

- The bandit's criterion on TEST-ADAPT was at most the frozen system's time, an upper interval bound below 1.05, and no class above 1.15; it is met.
- One deviation, absolute instead of control-variate feedback, was logged with development evidence before the test run.
- The strategy arms were motivated by the TEST-NEWCLASS result, so TEST-NEWCLASS is not a clean test for the bandit.

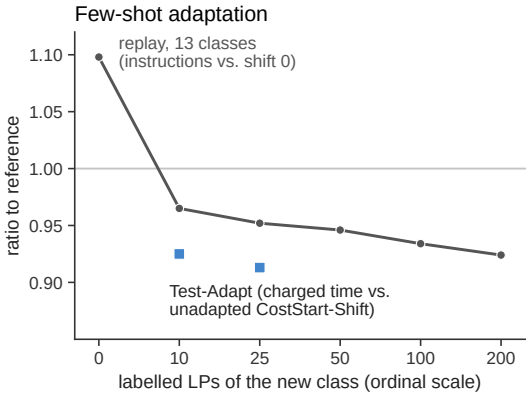

Figure 9: Few-shot adaptation to unseen classes; the n axis is ordinal. Line: replay on the training LPs with one class held out at a time, instructions relative to the unshifted start after adapting with n labeled LPs of the held-out class (the 13 classes with long enough streams; Table 25). Squares: the charged panel on TEST-ADAPT, time relative to the unadapted policy (the predictor's start).

K PUBLISHED METHODS: REIMPLEMENTATION AND THE PUBLIC BENCHMARK

This appendix describes our reimplementations of the two published methods, checks that our decoder and training choices do not handicap them on our data (Table 26), and reports the public benchmark of Lu & Yan (2025) under its own protocol (Appendix K.1).

Reimplementation.

- Fan et al. (2023): the released GCN with the released balanced cross-entropy loss, 0.2M parameters, Adam at 10^{-3} with weight decay 10^{-4}, learning rate $\times 0.1$ at one and two thirds of the budget, one LP per step. The paper's factorization-based basis repair is replaced by our crash on the basic log-odds, so that every learned method shares one decoder.
- Lu & Yan (2025): tripartite graph, 1.2M parameters, the basic-variable loss, the multi-level loss with $\theta = 0.9$ and ten levels from HiGHS replays under iteration limits from the all-slack start, and label preprocessing (the paper's separate feasibility term is not implemented). Inputs are clipped at 10 with global normalization and a gradient-norm clip of 1, which the unclipped bound features (up to 7×10^6) require; LPs without late-path labels (15%) use the terminal level only.
- Both: stage 1 on our 114,874 LPs of TRAIN-SMALL (Fan et al. 5.1 epochs, Lu & Yan 4.0, capped by label cost; a further Fan et al. continuation to 7.3 epochs gained nothing on its own loss), stage 2 on exactly the predictor's stage-2 data and budget (the full-budget Lu & Yan model on the deduplicated set), and the checkpoint chosen by the method's own loss on the selection split of DEV-20K. The full budget is 12 stage-1 epochs.

Differences from the papers that a reader should weigh: one model over all classes rather than one per dataset; presolved, stored LPs rather than raw files; our decoder rather than theirs. On their own benchmarks the papers report iteration ratios of roughly 0.3 to 0.6 against HiGHS' default (0.2 to 0.9 across individual datasets); our reimplementations reach 0.46 to 0.52 against the all-slack dual simplex on TEST-20K and TEST-50K (0.44 to 0.57 with TEST-80K).

Decoder fairness and seed variance on our data. Table 26 compares decoders and training seeds on DEV-20K (fully charged). Decoding the published models' scores with the papers' top-m decoders instead of our crash makes them slower relative to the predictor, except for Fan et al. on W2, where the difference is not significant; with top-m repair, Lu & Yan takes $2.41\times$ the predictor's time on W1. Models trained across 26 classes select many dependent structural columns, HiGHS alters the installed basis on about 90% of such starts, and the probe installs of the repair variant cost 0.13 to 1.2 s; the triangular crash avoids both. A second training seed of each reimplementation, at full stage-1 budget and decoded by the crash, changes its charged time by 0.94 to $1.08\times$ (paired; every interval includes 1), at most about 8%, far below the 1.2 to $2.9\times$ gap to the predictor; every interval against the predictor excludes 1 in the predictor's favor. Fan et al.'s failures are the known load-balancing wrong optima (4 to 5 per seed on W1, 1 on W2).

Table 26: Decoders and training seeds of the reimplemented published methods on DEV-20K: charged time relative to our predictor (class-balanced; W1: 392 LPs, W2: 69 LPs). Top-m installs the m highest-scoring columns and lets HiGHS repair the set. n.s.: not significantly different from the crash.

Method	Decoder or seed	W1	W2
Decoder (same model)			
Lu & Yan	our crash	1.41	1.21
	top-m	1.82	1.72
Fan et al.	our crash	1.50	2.84
	top-m	1.69	2.27 (n.s.)
Training seed (full budget, our crash)			
Lu & Yan	first / second seed	1.40 / 1.43	1.20 / 1.30
Fan et al.	first / second seed	1.46 / 1.38	2.85 / 2.90

The same holds on the W1 and W2 LPs of the frozen larger test sets (check inline; charged time relative to native HiGHS on TEST-50K/ TEST-80K):

- Lu & Yan at full budget: 1.02 / 1.03 with the crash, against 1.39 / 1.46 with the top-m decoder;

- Fan et al.: 0.99 / 1.00 against 1.12 / 1.19, with three top-m solves returning a solver error after installation and counted as failures;
- our predictor with the top-m decoder: 0.73 / 0.83, indifferent to the decoder ($1.04\times$ [0.88, 1.26] relative to the crash).

The crash-decoded published methods therefore do not understate the baselines in our setting, and the predictor's margin holds under every decoder. On the per-dataset benchmark below the opposite holds, which is why top-m is the right decoder there.

Results on TEST-20K. At full budget, Fan et al. removes 51% of the iterations on TEST-20K and is 10% slower than native HiGHS; Lu & Yan removes 53% and is 6% faster, and a second training seed of each gives $1.04\times$ and $0.96\times$. Their starts go through the same crash as ours with the same 40 to 60 ms of overhead, so the difference lies in the scores; with the papers' own top-m decoders they are slower still on our data (1.16 to $1.19\times$ on TEST-20K). Our predictor takes $0.73\times$ [0.64, 0.82] the time of Lu & Yan, paired, with $0.67\times$ its iterations and $1.08\times$ its time per iteration (Table 11).

K.1 THE PUBLIC BENCHMARK OF LU & YAN

Protocol. The benchmark has four public datasets (Mirp1, Mirp2, Anonymous, Load balancing). We follow the protocol of Lu & Yan (2025): one model per dataset, a 7:3 split (ours, since theirs is unpublished; 9 to 30 test LPs per dataset, 80 in total), presolved LPs, and fully charged time against the HiGHS dual simplex with presolve off. Our predictor, pretrained on our synthetic data, is fine-tuned per dataset; the published models are trained from scratch per dataset as in the papers. The top-m repair decoder fixes the number of basic variables at m by rank and repairs singular selections as in Fan et al. (2023); top-m without repair installs the top-m set through `setBasis` and lets HiGHS repair it.

Per-dataset decision policy. The action set has 39 actions: our crash with the nine shifts and the top-m decoder with seven shifts, each under default pricing or Devex; top-m repair under either rule; the unshifted crash and top-m starts with primal simplex; and native HiGHS with three settings. Every training LP is solved once under every action, with instruction counts as labels. For each dataset, 5-fold cross-validation over the training LPs selects among the best single action, a policy trained from scratch with the loss of Eq. (2), and a policy initialized from COSTSTART-FULL and fine-tuned; the winner is refit on all training LPs. No novelty gate is used, since every test LP comes from the dataset's own generator. With 19 to 70 training LPs per dataset, the selection keeps top-m repair on Mirp1 and a constant action (top-m with primal simplex) on Load balancing, and trained policies on the other two.

Results. Table 27 gives the per-dataset results. The decision policy takes $0.58\times$ the iterations and $0.72\times$ [0.68, 0.75] the time of the HiGHS dual simplex, against $1.03\times$ for our reimplementation of Lu & Yan and $0.92\times$ for Fan et al. Paired, it takes $0.69\times$ [0.64, 0.75] the time of Lu & Yan (Mirp1 0.48, Anonymous 0.64, Load balancing 0.74, Mirp2 1.03) and $0.78\times$ that of Fan et al. Against our fine-tuned predictor with top-m repair it takes $0.84\times$ the time at the same iterations ($1.02\times$), so the gain comes from cheaper decoding and cheaper iterations. The best single action per dataset alone reaches $0.73\times$, so on datasets this small most of the gain comes from choosing among decoders, algorithms and pricing rules by measured cost, not from a per-LP choice; initializing the policy from COSTSTART-FULL, which was trained on our data, hurts ($0.86\times$).

Raw (unpresolved) files. On the unpresolved versions of the four datasets (same protocol, 79 test LPs), the decision policy takes $0.45\times$ the iterations and $0.585\times$ [0.53, 0.65] the time of the HiGHS dual simplex, against 0.58 / 0.82 for our reimplementation of Lu & Yan, 0.58 / 0.75 for Fan et al. and 0.45 / 0.65 for our fine-tuned predictor with top-m repair; paired, it takes $0.71\times$ [0.63, 0.81] the time of Lu & Yan and $0.78\times$ that of Fan et al. Lu & Yan report 0.57 / 0.73 on raw files. On these files, however, HiGHS with its own presolve takes $0.29\times$ the time of the dual simplex without presolve, faster than every warm start, ours included; learned starts are useful here only when the LP is solved without presolve or presolve has already been applied, as on our data.

Earlier panel and zero-shot model. An earlier panel on the same split, without the decision policy, gives the following. Our predictor without fine-tuning (trained on our 25 classes) reaches $0.91\times$ the iterations and $1.07\times$ the time with our crash and does not benefit from the top-m decoders (1.11 to $1.15\times$ time). Mirp2 is at parity for every method on time, and it is the one dataset where the

Table 27: Lu & Yan's public benchmark under their protocol (one model per dataset, 7:3 split, presolved LPs): geometric-mean ratios over test LPs against the HiGHS dual simplex with presolve off (iterations / fully charged time), with the geometric mean over the four datasets and its 95% interval. Test LPs per dataset: Mirp1 9, Mirp2 18, Anonymous 23, Load balancing 30; native solve times 9.6, 30, 1.2 and 3.0 s. All rows come from one paired panel except the two reported ones.

Method (decoder)	Mirp1	Mirp2	Anonymous	Load bal.	Geomean [95%]
Fan et al. reimplemented (top-m)	0.58 / 0.72	1.01 / 1.14	0.92 / 1.62	0.61 / 0.69	0.76 / 0.98 [0.93, 1.04]
Fan et al. reimplemented (top-m repair)	0.55 / 0.69	0.78 / 0.92	0.87 / 1.69	0.56 / 0.68	0.68 / 0.92 [0.86, 0.99]
Lu & Yan reimplemented (top-m)	0.70 / 1.10	0.92 / 1.07	0.71 / 1.37	0.60 / 0.68	0.72 / 1.02 [0.95, 1.10]
Lu & Yan reimplemented (top-m repair)	0.71 / 1.12	0.80 / 0.99	0.70 / 1.48	0.58 / 0.70	0.69 / 1.03 [0.96, 1.12]
our predictor, fine-tuned (top-m)	0.43 / 0.64	1.00 / 1.07	0.55 / 1.05	0.59 / 0.67	0.61 / 0.83 [0.78, 0.88]
our predictor, fine-tuned (top-m repair)	0.35 / 0.57	0.99 / 1.06	0.56 / 1.27	0.55 / 0.69	0.57 / 0.85 [0.80, 0.91]
best single action per dataset	0.39 / 0.54	0.96 / 1.00	0.55 / 1.04	0.51 / 0.51	0.57 / 0.73 [0.67, 0.80]
CostStart decision policy, per dataset	0.35 / 0.53	0.98 / 1.02	0.65 / 0.94	0.51 / 0.51	**0.58 / 0.72** [0.68, 0.75]
Lu & Yan, as reported (their decoder)			–		0.60 / 0.73
Fan et al., as reported (their decoder)			–		0.74 / 0.89

published top-m repair beats our predictor on iterations ($1.24\times$ against it). Paired at each method's best decoder, our fine-tuned predictor takes:

- on the presolved files, 0.83 [0.78, 0.88] the iterations and 0.81 [0.76, 0.87] the time of Lu & Yan, and 0.84 / 0.89 those of Fan et al.;
- on the raw files, 0.78 / 0.79 those of Lu & Yan and 0.79 / 0.87 those of Fan et al. There our predictor reaches $0.45\times$ the iterations and $0.65\times$ the time of the dual simplex, against the 0.57 / 0.73 that Lu & Yan report, and HiGHS with presolve on takes $0.29\times$ the time of the dual simplex without presolve, faster than every warm start.

Reproduction. Our reimplementation reproduces the reported results of Lu & Yan on the load-balancing and anonymous datasets and those of Fan et al. on Mirp1 (raw iterations 0.536 against the paper's 0.539), but not Lu & Yan's Mirp1 figure. The paper reports $0.32\times$ the iterations; our reimplementation reaches 0.62 to $0.70\times$ across five retrains that vary the input clipping and the preprocessing term (seed noise ± 0.04), and only 0.63 to $0.68\times$ on its own training LPs, while our own per-dataset predictor reaches $0.35\times$ there. An audit ruled out:

- the HiGHS version (1.7.2, 1.9.0 and 1.13.1 give identical iteration counts);
- the algorithm choice and presolve;
- the data sizes, which match the paper's table to within one;
- the labels and the split.

What remains is something in the unreleased model or training beyond the paper's text, or an unreported evaluation detail. We report both figures. Full tables are released with the code.

Reading of the public benchmark. Three caveats apply: the split is ours, since theirs is unpublished, with 9 to 30 test LPs per dataset; our predictor is pretrained on our synthetic data and then fine-tuned per dataset, whereas theirs is trained from scratch per dataset; and zero-shot, without fine-tuning, no learned start beats the default there. On their smallest dataset, whose LPs solve in about a second, our time is dominated by decoding and basis installation. On these datasets the decoder matters more than the model: every per-dataset model gains from the top-m repair, and our triangular crash is the wrong decoder there (the same fine-tuned predictor reaches only $0.83\times$ the iterations and $0.97\times$ the time with it). The shift policy adds at most 0.04 there, so the gain is the predictor's.

L Extended discussion: scope, limitations and deployment

When the system is meant to be used. The setting in which learned warm starts pay off is a stream of new LPs from known problem classes that must be solved by simplex, for example because a basic solution is needed at the root of branch-and-bound, for cut generation or for sensitivity analysis, or because runs must be reproducible. In that setting the classes are known in advance, no earlier basis of the same LP is available, the training cost is paid once, and the relevant question is how much solve time is saved per LP. Our results support three practical recommendations. First, train on the classes that will be solved and include held-out dual forms or variants only where the class itself is known; the gains transfer to larger instances of the same classes and to their dual forms, and

partly to new variants. Second, deploy with the guardrails: the novelty gate keeps the policy from extrapolating, the harm gate is needed wherever the novelty gate takes decisions away from the policy (at sizes beyond the training range), and the optimality check is necessary on badly scaled classes, where it removed every wrong optimum we observed. Third, when a new class appears, expect no gain without in-class feedback; either label a few LPs of the class (about ten times their plain solve cost each) or let an online bandit adapt from its own solves over the first hundred or so LPs.

Transfer to new problem classes. None of the learned components transfers to problem classes absent from training, and neither did any selector or policy input we tried. On such classes the gated system takes $1.2\times$ the time of native HiGHS; the novelty gate recognizes the LPs as unfamiliar, but its fallback was chosen on training classes held out one at a time, which proved a poor stand-in for new classes. Adaptation recovers part of the loss, but needs labels or a stream of roughly 100 LPs or more per class.

The pricing action. The advantage of COSTSTART-FULL over COSTSTART-ALG is not significant in time on any test or development panel. What is consistent is how they reach their times: COSTSTART-FULL accepts more iterations that each cost less, and its gain concentrates on short solves. Only one alternative pricing rule was offered: Devex forced from the first iteration, against HiGHS' default, which starts with DSE and switches to Devex on its own in some cases. It was the only one of 14 solver settings in our probe that clearly helped.

Scope of the theorem. Theorem 1 is an existence result for solve-vector work under one pivot rule on a network-flow LP. The unbounded ratio holds in a cost model without per-iteration terms linear in the number of rows; an implementation with such a term, or with refactorization after a fixed number of updates, bounds the ratio of total work by a constant. HiGHS shows the separation in full with presolve and scaling off and Devex forced. Under its defaults, the exact initialization of the DSE weights, which a nonslack start pays unless it is already near optimal, masks most of it, and the defaults also change the paths. On our real LPs that initialization is a median 0.4% of the solve. The theorem motivates the design; the empirical case rests on the measurements of §3.2.

Data, solver and hardware. All main results use synthetic instances from one generator library and one solver. On the public benchmark of Lu & Yan (2025), our decision policy beats the reimplemented baselines under their protocol, but it needs per-dataset training there, and on the benchmark's unpresolved files HiGHS' own presolve is faster than every warm start, and with 19 to 70 training LPs per dataset most of its gain comes from choosing a single action per dataset. Timing depends on hardware; we report paired ratios within panels and instruction counts beside every time. Our timings charge a GPU forward pass; on CPU cores alone the forward pass is about ten times slower, which costs 29 to 30% on solves under 10 s (Appendix I.3).

Cost of the labels. The cost table requires solving every training LP under every action, about 16 times the cost of solving each LP once for the twenty actions of COSTSTART-FULL. This is a one-time cost per set of classes, but it limits how many actions the policy can consider.

M SYNTHETIC LP INSTANCE GENERATION

Empirical work on linear programming is constrained by the supply of instances. The public collections (Netlib, MIPLIB and their descendants) are small, fixed, and curated for purposes that need not align with a given study. Netlib contains roughly one hundred usable linear programs: enough to populate a benchmark table, but not enough to fit a model, to measure how a method's behaviour varies with problem size, or to hold out a test set genuinely disjoint from the instances used during development.

The common alternative is a random-matrix generator that samples A, b and c from some distribution. Such instances are unlimited but structurally uninformative: a dense Gaussian A exhibits none of the sparsity, block structure or network incidence pattern characteristic of practical linear programs, and conclusions drawn from it transfer poorly. The gap, then, is between a hundred realistic instances and an unlimited supply of unrealistic ones.

We use SYNTHETICLPS.JL[1] to close it. Rather than sampling matrices, the library samples *problems*: a transportation plan, a refinery blend, a radiotherapy fluence map, a crew roster. Each is a genuine

[1] https://anonymous.4open.science/r/SyntheticLPs-674F/README.md

Table 28: Orthogonal axes of the generated instance space. The final three are post-`build_model` transforms applied centrally, so they act uniformly across all variants rather than being reimplemented per generator.

Axis	Effect on the emitted model	Cardinality
`variant`	the formulation itself	129
`target_variables`	dimensions re-derived, not padded	wide range
`seed`	fresh data at fixed size and structure	unbounded
`feasibility_status`	feasible / infeasible / unknown	3
`relax_integer`	a MIP, or its LP relaxation	2
`bounds_to_constraints`	bounds kept, or emitted as explicit rows	2
`dualize`	the primal, or its LP dual	2

formulation carrying the constraint structure its domain implies, parameterised so that its data—and hence its dimensions—can be redrawn for every instance. The matrix structure is inherited from the model; only the numbers are sampled. This section describes the design (§M.1), characterises the resulting instance space (§M.2–§M.3), and reports measurements of the properties we rely on (§M.4–§M.5).

All measurements below were obtained with HiGHS 1.15.1 through JuMP, with presolve disabled for every solve-based statistic, over 2364 generated instances across three campaigns.

M.1 Design

The library is organised in two levels. A *category* is a problem domain, such as transportation, radiotherapy or hub location, and a *variant* is a concrete formulation within it. Variants are distinct models rather than parameter settings of a common model: within the transportation category, the capacitated and fixed-charge variants differ in their rows, not merely in their coefficients. The version used here provides 45 categories comprising 129 variants, of which 71 contain integer or binary variables and 58 are pure-continuous linear programs.

Every variant implements a two-stage contract. A constructor

$$\texttt{Variant}(\textit{target_variables},\ \textit{status},\ \textit{seed})$$

performs all random sampling and stores the resulting data in a plain struct; a second function `build_model` maps that struct to a JuMP model and contains no calls to a random number generator. Sampling draws from a constructor-local `MersenneTwister` seeded by the *seed* argument.

Three consequences follow, and each is relied upon later. First, an instance is determined entirely by the tuple $(\textit{variant}, \textit{target_variables}, \textit{status}, \textit{seed})$. Second, generation neither reads nor advances the caller's global random state, so instances may be produced concurrently on many threads without coordination and without perturbing any other stochastic component of a pipeline. Third, the sampled data remains inspectable as a typed object before any model is built: a transportation instance exposes supplies, demands and costs rather than an anonymous sparse matrix, so downstream code may condition on problem semantics that an MPS file does not retain.

M.2 The instance space

The 129 variants constitute a catalogue, not an instance count. Each is crossed with several orthogonal axes, summarised in Table 28. Because the axes compose, the reachable set of distinct models is effectively unbounded.

These are not cosmetic relabellings. Changing the seed changes the data; we verify in §M.4 that a different seed yields a byte-different model for all 129 variants. Changing `target_variables` re-derives the underlying dimensions—the numbers of sources and destinations, periods and machines, beams and voxels—rather than padding a fixed skeleton. Dualisation transposes the entire structure. Both the size distribution and the dualisation probability may themselves be sampled, which makes formulation shape a controllable covariate of a generated dataset rather than a fixed property of it.

Table 29: Size fidelity and generation cost (558 builds). The median ratio of achieved to requested variable count is exactly 1.000 at every scale. Median build time is flat at $\approx$52 ms from 200 variables upward; the 50-variable column is omitted because it absorbs per-variant JIT compilation on first call. Variant counts fall at the largest targets by design (see text).

Requested	Variants	Median ratio	Within $\pm 10\%$	Within $\pm 25\%$	Median build (s)
50	129	1.000	91 %	99 %	—
200	129	1.000	96 %	100 %	0.052
1000	129	1.000	97 %	98 %	0.052
5000	93	1.000	96 %	97 %	0.052
20 000	78	1.000	90 %	91 %	0.053

Table 30: Determinism and RNG isolation, verified over all 129 variants by comparing SHA-256 digests of serialised MPS output.

Property	Variants passing
Identical inputs $\Rightarrow$ byte-identical model	129 / 129
Different seed $\Rightarrow$ different model	129 / 129
Unaffected by the caller's global RNG state	129 / 129
Does not advance the caller's global RNG	129 / 129

M.3 Scale and generation cost

The `target_variables` argument is a request, and the generator selects internal dimensions to meet it. Table 29 reports how faithfully it does so across 2.6 orders of magnitude, building one instance per variant at each target.

For most of the catalogue the requested count is met exactly, because the generator solves for dimensions rather than approximating them. Build cost is essentially independent of size—work is proportional to the number of nonzeros, and sparse models remain sparse as they grow—giving roughly 19 instances per second on a single thread, or about 1.6 million per core-day. Combined with the RNG isolation of §M.1, this parallelises without coordination.

One class of exception should be stated. A small number of formulations are dense by mathematical necessity—the regression family, whose design matrix is dense, and both portfolio variants—so their nonzero count grows as $O(mn)$ rather than $O(n)$. These reach 10 s to 13 s per build at a few thousand variables and become impractical beyond that; they are why 78 rather than 129 variants were measured at 20 000. This is a property of dense linear programs rather than a defect of the generator, but it means that arbitrary scaling holds for the sparse majority of the catalogue and not for every member of it.

M.4 Reproducibility and feasibility control

Two properties make the generated instances usable for controlled experiments rather than merely for volume.

Determinism. We serialised each generated model to MPS and compared SHA-256 digests across all 129 variants. Every variant satisfies all four properties in Table 30. The latter two follow from the constructor-local RNG and are what make parallel generation safe. A practical consequence is that a dataset is fully described by its seed and configuration, and can therefore be distributed as a few lines of code rather than as gigabytes of MPS files.

Feasibility as a contract. Each generator accepts `feasible`, `infeasible` or `unknown` and honours the request by construction, adjusting capacities, demands or budgets so that the requested status holds without invoking a solver. We tested this exhaustively over 129 variants $\times$ 3 seeds $\times$ both definite statuses, solving each with presolve disabled: all 387 `feasible` requests terminated OPTIMAL and all 387 `infeasible` requests terminated INFEASIBLE, giving 774/774 contracts honoured with no construction failures. Many generators additionally attach an auditable artifact to the problem struct—a complete primal *witness* for feasible requests, or a structural infeasibility

Table 31: Structural comparison against Netlib (medians). Sparsity statistics agree closely; numerical range and degeneracy remain lower than in real instances.

Property (median)	Netlib	Synthetic	Assessment
Matrix density	0.0079	0.0132	close
Nonzeros per column	4.23	3.71	close
Nonzeros per row	8.45	5.14	close
Column-degree variability (CV)	0.55	0.43	close
Primal degeneracy	0.196	0.094	about half
Simplex iterations per row	1.24	0.55	less work
Coefficient spread, $\log_{10}\lvert A_{ij}\rvert$	4.12	1.85	narrower
Variables absent from the objective	0.382	0.000	gap

Table 32: Feature-space breadth, reported as the 5th–95th percentile interval over each corpus.

Feature	Synthetic	Netlib
Matrix density	0.006–1.000	0.000–0.092
Nonzeros per column	1.96–55.98	2.18–14.24
Column-degree variability (CV)	0.00–2.40	0.11–1.23
Coefficient spread, $\log_{10}\lvert A_{ij}\rvert$	0.00–5.75	0.00–8.65
Fraction of entries equal to ± 1	0.000–1.000	0.094–1.000
Primal degeneracy	0.000–0.800	0.000–0.841

certificate checkable without a solver. Where a generator's feasibility logic is heuristic, supplying an optimizer enables verification: the model is solved on a copy and rebuilt with the next seed on mismatch, walking seeds deterministically so that a given (seed, optimizer) pair always resolves to the same instance.

The `unknown` status deliberately provides no guarantee; roughly a quarter of instances generated under it are infeasible, and we therefore use the explicit statuses wherever feasibility matters.

A controlled infeasible set of arbitrary size is difficult to assemble from public collections—Netlib offers 29 infeasible instances in total—and is available here at the same cost as a feasible one.

M.5 Structural characterisation

Structured generation is worthwhile only if the output resembles real linear programs. We compared the corpus against the Netlib collection using an identical pipeline—the same extraction of A, b and c, the same solver, presolve disabled—over 283 synthetic instances at 400 variables and 109 Netlib instances. Table 31 reports medians.

Sparsity—the property that naive generators reproduce worst—falls in the right range, with density, column length, row length and column-degree variability all within a factor of 1.7 of Netlib. This is the direct return on generating from formulations rather than from distributions: the incidence structure is inherited from the model and is therefore correct without being fitted.

Two gaps are worth stating explicitly. Synthetic coefficients span 1.85 orders of magnitude against Netlib's 4.12, so the corpus under-exercises numerical robustness; and in most variants every variable carries a nonzero objective coefficient, whereas real models accumulate auxiliary and balance columns absent from the objective (Netlib median: 38% of columns). Both stem from the same source—parameters drawn independently from smooth continuous distributions—and both are addressable in the sampling layer without modifying any formulation.

Against this, breadth is itself a useful property, and on several axes the corpus reaches further than Netlib in at least one direction (Table 32). It ranges from pure network matrices, in which every entry is ± 1 and the constraint matrix is totally unimodular, to fully dense statistical linear programs, and from completely non-degenerate instances to ones that are 80% degenerate. A study requiring a particular regime can select variants that occupy it rather than waiting for one to occur.

M.6 LIMITATIONS

For completeness we record the limitations that bear on how these instances should be interpreted. Numerical range is narrower than reality (1.85 against 4.12 decades), so results turning on numerical conditioning should not treat this corpus as representative. Primal degeneracy is roughly half Netlib's: it is present where the combinatorics force it—assignment, covering and routing polytopes—and largely absent in continuous allocation models, where real data would supply it through repeated and rounded values. Dense variants do not scale past a few thousand variables (§M.3). Multi-period data is drawn with temporally independent noise, so inter-temporal coupling constraints—ramping, storage, inventory balance—bind less tightly than they would under the persistent deviations real demand exhibits. Finally, Netlib is an imperfect yardstick: its instances are larger than those measured here (median 1376 against 400 columns), so part of the iteration-count difference reflects size rather than realism.

M.7 FULL GENERATOR CATALOGUE

All 45 categories and 129 variants registered in the version used here.

Category	#	Variants
airline_crew	1	standard
assignment	2	standard, workload_balance
bin_packing	2	heterogeneous, standard
blending	3	equipment_batches, multi_product, standard
container_loading	2	standard, two_dimensional_bin_packing
crop_planning	1	standard
cutting_stock	4	due_dates, integer_patterns, setup_cost, standard
diet_problem	3	food_groups, nutrient_bounds, standard
energy	7	dc_opf, optimal_transmission_switching, ramping, reserves, standard, storage, transmission
facility_location	3	p_median, standard, two_echelon
feed_blending	1	standard
graph_optimization	6	generalized_independent_set, independent_set, map_labeling, quasi_clique, vertex_coloring, vertex_cover
hub_location	8	budgeted_backbone, capacitated, compact_single_allocation, hub_covering, hub_network, multiple_allocation, p_hub_median, r_allocation
inventory	4	lot_sizing, multi_echelon, multi_item, standard
inverse_optimization	8	classical_normalized, linf, market_clearing, noisy_observations, restricted_optimal_value, shortest_path, shortest_path_layered, standard
job_shop_scheduling	1	standard
knapsack	4	bounded, mixed_integer_set, multidimensional, standard
land_use	1	standard
load_balancing	2	discrete_placement, standard
maritime_inventory_routing	1	standard
multi_commodity_flow	3	binary_capacity, integer_flow, standard
network_flow	2	generalized_flow, standard
neural_network_verification	1	relu_big_m
nurse_scheduling	1	standard
operating_room_scheduling	6	benchmark_loading, case_sequencing, elective_assignment, master_surgical_schedule, robust_elective, weekly_planning
portfolio	2	cvar, tracking_error
process_planning	5	campaign, capacity_expansion, hydrogen_network, mode_switching, refinery
product_mix	1	standard
production_planning	1	standard
project_selection	1	standard
radiotherapy	5	beam_angle_selection, mean_tail_dose, minmax_deviation, robust_fluence, weighted_deviation
regression	4	basis_pursuit, chebyshev, lad, quantile
resilient_network_design	1	standard
resource_allocation	1	standard

Category	#	Variants
revenue_management	2	standard, stochastic_overbooking
scheduling	1	standard
set_system	4	combinatorial_auction, set_cover, set_packing, set_partitioning
stochastic_program	1	standard
supply_chain	5	carbon, multi_product, network_planning, single_source, standard
telecom_network_design	1	standard
transportation	6	balanced, capacitated, emission_constrained, fixed_charge, standard, transshipment
tsp	8	assignment_relaxation, asymmetric, flow, multiple_salespersons, precedence, prize_collecting, standard, time_windows
unit_commitment	1	standard
vehicle_routing	1	cvrp
workforce_shift_scheduling	1	covering