DESCENT SHRINKS THE ADVANTAGE OF LEARNED MOVE SELECTION IN LOCAL SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Iterated local search (ILS) runs a best-improvement descent after each perturbation, which turns even random perturbations into an effective search. We find that this descent helps random move selection far more than learned local-search policies for combinatorial optimization, so that in five of the six methods we study the learned policy loses most or all of its advantage over random selection once descent is added. We call the number of descent steps run after each applied move the *commit depth* (cd). In a controlled job-shop study, a random-forest scorer beats random selection by 220 makespan units at $\mathrm{cd}{=}1$ but trails it by 126 with descent to a local optimum, because a scorer trained on the outcome after one descent step prefers moves that change the solution least. In six published methods, with each selector at its best commit depth, the advantage of the learned selector shrinks by a factor of 34 to 69 in ECO-DQN and 42 in NeuOpt, and disappears or reverses in DACT, L2S, and a learned SAT heuristic that we trained. Where the published policy is still ahead of random selection with descent, fine-tuning it with descent in the training loop helps: the fine-tuned ECO-DQN and NeuOpt policies beat four baselines at equal wall-clock time on the instance sizes used for fine-tuning, and in ECO-DQN they close 45 to 87% of the published policy's remaining gap to the best known cut by selecting flips from which descent finds larger improvements. Where the published policy is not ahead (DACT, MIStar, L2S), the fine-tuned policy does not beat random selection with descent. Whether the published policy is ahead can be checked with evaluation runs alone; of the fine-tuning outcomes we recorded as predictions before the runs, two held and a third held in part. We recommend reporting learned-versus-random comparisons with and without descent, with the same number of evaluated candidates per step.

1 INTRODUCTION

Learned improvement heuristics select moves in a local search: a trained model scores candidate moves, or a policy samples one, and the search loop applies the chosen move (Chen & Tian, 2019; Hottung & Tierney, 2020; Wu et al., 2022; Zhang et al., 2024). Their added value over the loop itself is best measured against random move selection in the same loop, but the loop is rarely varied. Yet learned components can lose their advantage once the surrounding search is strong enough: heatmaps and graph neural networks (GNNs) have been matched by untrained or simple alternatives inside tree search and with local-search post-processing (Boether et al., 2022; Xia et al., 2024; Sun & Yang, 2023).

We study one parameter of the search loop that can reverse this comparison. After a move is applied, the loop either selects the next move immediately, as learned improvement heuristics do, or first runs best-improvement descent, as iterated local search (ILS) does (Lourenço et al., 2003). We call the number of descent steps between an applied move and the next selection the *commit depth* (cd): $\mathrm{cd}{=}0$ is the default of learned improvement heuristics, and descent to a local optimum is the default of ILS. The basic idea of ILS is that descent turns random perturbations into an effective search. To our knowledge, the effect of descent on the evaluation of learned local-search methods has not been measured; we measure it here.

Descent helps random selection far more than learned move selection: a random move followed by descent is an ILS step, whereas a learned selector helps only if it ranks the candidates correctly by

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.

their value after descent, and in our JSSP study a scorer trained on one-step post-descent outcomes prefers moves that change the solution least. Fine-tuning a published policy with descent in the training loop helps where that policy is still ahead of random selection with descent: it then learns moves from which descent finds larger improvements. Our results fall into three parts (Figure 1).

(i) A controlled study. In job-shop scheduling (JSSP), a random-forest (RF) scorer's gain over random selection is 0.4 makespan units at $\mathrm{cd}{=}0$, $+220.2$ at $\mathrm{cd}{=}1$, and -125.6 with descent to a local optimum (Section 3). The reason is that the scorer prefers moves that change the schedule least, and after such moves, descent to a local optimum returns to the same makespan.

(ii) Published methods. We vary the commit depth in six published methods without retraining them (Section 4). When random selection is run with the same descent and each selector uses its best setting, the advantage of the learned selector falls by a factor of 34 to 69 in ECO-DQN and 42 in NeuOpt, and it disappears or reverses in DACT, L2S, and the learned SAT heuristic. The released MIStar policies assign almost the same probability to every feasible move. Their advantage comes from evaluating the exact makespan of 50 candidates per step, and it is small against a random selector that does the same. Descent without a tabu restriction often reverses the selected move, for learned and random selection alike (96 to 100% of steps in DACT); a tabu restriction prevents this and helps both, random selection more so. With it, the released DACT policy, without retraining, reaches a 1.7% optimality gap on TSP-200 in the time its original loop needs to reach 8.2%, and random selection with the same descent does better still. At equal wall-clock time, random selection with descent is as good as or better than DACT and L2S. These methods were designed and evaluated for a loop without descent, where they are far ahead of random selection. With descent, random moves become a competitive iterated local search, a baseline that their evaluations did not need to include.

(iii) Fine-tuning with descent in the training loop. Before fine-tuning, we compare the published policy with random selection, each at its best setting and at equal wall-clock time; we call it the *pre-fine-tuning comparison* (Section 5). In ECO-DQN and NeuOpt, where the published policy is ahead, the fine-tuned policy (the checkpoint with the best validation score) beats all four baselines on the instance sizes used for fine-tuning: the published policy with and without descent, random selection with descent, and a control fine-tuned without descent. In ECO-DQN it closes 45 to 87% of the gap between the published policy with its best descent setting and the best known cut (on Barabási–Albert (BA) graphs with 60 vertices, from 1.9% to 0.3%), and on the 800-vertex GSET graphs it beats the original loop by 0.4 to 0.5 percentage points (pp) but only ties the published policy with descent. In DACT, MIStar, and L2S, where the published policy is not ahead, the fine-tuned policy does not beat random selection with descent at equal wall-clock time. A successful fine-tuned policy selects moves from which descent finds larger improvements: in ECO-DQN the cut improvement from descent per step rises from 0.85–1.38 to 1.33–2.32, and on Erdős–Rényi graphs with 60 and 200 vertices (ER60, ER200) the share of selected flips that lower the cut rises from 24–31% to 46–54% (random selection: 94–98%). Where fine-tuning does not beat random selection, the policy's entropy rises toward that of uniform random selection (DACT, L2S). We recorded two of these outcomes as predictions before the runs: ECO-DQN on BA graphs and L2S; a third prediction, on GSET, held only in part.

Contributions.

- We show that the commit depth changes the sign and size of learned-versus-random comparisons, in a controlled JSSP study with learned and greedy scorers and three acceptance criteria (Section 3) and in six published methods (Section 4).
- We show that two evaluation choices enlarge the measured advantage of learned selectors: comparing at $\mathrm{cd}{=}0$ only, and letting the learned selector compute the exact objective of more candidates per step than the random baseline (Section 4).
- We fine-tune five published policies with descent in the training loop, compare them with four baselines at equal wall-clock time, and find that the fine-tuned policy beats random selection with descent in the two methods where the published policy already does, and in none of the other three (two of these outcomes were predicted before the runs). Where fine-tuning helps, the policy learns moves from which descent finds larger improvements (Section 5).

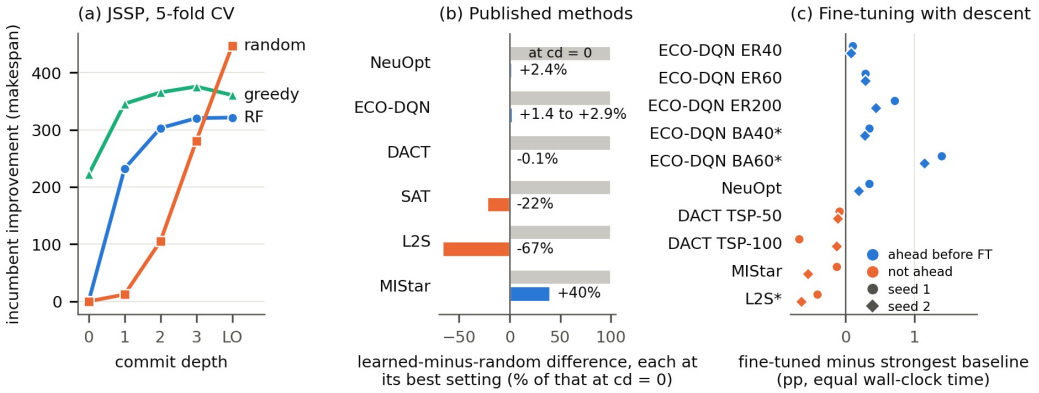

Figure 1: **Commit depth changes learned-versus-random comparisons.** *(a)* JSSP, 5-fold cross-validation (n=50): improvement of the incumbent (best makespan found) over the start after 100 steps for the RF scorer, the greedy selector, and random selection; LO = descent to a local optimum after each move. The RF gains most over random selection at cd=1, but random selection with descent to a local optimum, i.e., ILS with random perturbations, is best overall. *(b)* Published methods (first test set of each method in Table 2; for ECO-DQN the range over its graph families): learned-minus-random difference when both selectors use their best descent setting, as a share of the difference at cd=0, the original loop (gray, 100%); orange: the advantage is reversed; for MIStar the random selector applies one move per step, and against one that evaluates the same 50 candidates the difference is +0.3 at cd=0 and +0.9 at the best setting (makespan units). *(c)* Fine-tuning five published policies with descent in the training loop: fine-tuned policy minus the strongest of four baselines at equal wall-clock time (best-validation checkpoints; circles and diamonds are the two seeds). *Outcome predicted before the runs. Blue: the published policy beats random selection with descent before fine-tuning (column "Before FT" in Table 3); orange: it does not.

2 SETUP

Loop and commit depth. All methods share one loop (Algorithm 1, Appendix A.1). At each step a selector picks one move from the current solution's neighborhood: a learned scorer ranks a sampled candidate list, a learned policy samples from its distribution, and random selection picks uniformly. The move is applied, and then up to cd best-improvement descent steps are run before the next selection; cd=LO runs descent to a local optimum (with a step cap). Unless stated otherwise, the new solution is always accepted (random-walk acceptance), as in the published methods, and we report the best solution found (the incumbent). Descent can undo the selected move. For some methods we therefore also use descent with a tabu restriction: the descent that follows a selected move is not allowed to restore what that move removed (e.g., the two edges removed by a 2-opt move). Tabu search uses such restrictions to keep a move from being reversed (Glover, 1989). We call descent without this restriction *plain descent*.

Comparisons. The *learned-minus-random difference* is the difference in solution quality between a learned selector and random selection with the same loop, number of steps, and instances, paired over (instance, seed). Both use the same commit depth, except when we compare *each selector at its best descent setting* (or its best setting): the commit depth and descent variant (e.g., plain or tabu-restricted) with the best mean result for that selector on the test instances. Both selectors are tuned by the same procedure. Since descent costs time, we also compare at equal wall-clock time, by reading each method's anytime curve at a common time (in Section 5, the time the fine-tuned policy needs for its step budget). All p-values are from paired t-tests over (instance, seed) pairs unless stated otherwise; Wilcoxon tests and win/tie/loss counts are in the appendices.

JSSP study. Instances are Taillard ta01–10 (15×15) (Taillard, 1993). Candidate moves are random compound moves (pairs of elementary SWAP or INSERT moves on two machines); an elementary move

Table 1: **JSSP, 5-fold CV (n=50).** Difference in incumbent improvement (makespan units; positive = first selector better) after 100 steps, by commit depth. LO: descent to a local optimum after each move (at most 100 descent steps). $^{***}p<0.001$, $^{**}p<0.01$, $^{*}p<0.05$, ns = not significant; —: not run. Per-cell CIs and effect sizes: Appendix A.3.

	cd=0	cd=1	cd=2	cd=3	LO
RF − random	+0.4 ns	**+220.2** ***	+198.0 ***	+39.6 **	−125.6 ***
GNN − random	+0.5 ns	**+249.2** ***	+206.2 ***	+37.6 **	—
RF − greedy	−221.5 ***	−113.4 ***	−62.4 ***	−55.3 ***	−39.2 **
RF (absolute)	0.4	232.2	303.1	320.4	321.5
random (absolute)	0.0	12.0	105.1	280.8	**447.2**

that would create a cycle is skipped, so some candidates leave the schedule unchanged. These *null moves* (35 to 39% of raw candidates) are removed from the candidate lists and from the training data (Appendix A.4 reports results with them). Descent applies the best improving critical-block boundary swap ($\mathcal{N}_5$; Nowicki & Smutnicki, 1996) or, failing that, the best improving critical-operation insertion. The RF (100 trees, depth 12) predicts δ_1, the makespan decrease after the move and one descent step, from 51 state and 14 move features; we also train a message-passing GNN on the same target and use a greedy selector that evaluates every candidate's immediate makespan exactly. At each step the selector scores 30 candidates; runs have 100 steps from a shared start (a random schedule after 20 descent steps), 5 seeds per instance, and 5-fold cross-validation over instances (n=50 pairs per cell).

3 A Controlled Study: Job-Shop Scheduling

The sign of the difference depends on the commit depth. In Table 1 only the commit depth varies; the scorer, its training target, and the number of steps are fixed. At cd=0 the RF and random selection almost never improve on the start: random compound moves rarely shorten the critical path (van Laarhoven et al., 1992), and without descent a worsening move is never undone. At cd=1 the RF gains +220.2 over random selection (95% CI [195.3, 245.0], Cohen's d=2.52, $p<10^{-22}$; every fold between +159 and +250). With descent to a local optimum it falls 125.6 below random selection (d=−1.60, $p<10^{-14}$; negative in 47 of 50 runs). The GNN gives similar differences at cd=0 to 3 (it was not run at LO). The pattern is similar under non-worsening and simulated-annealing acceptance: at cd=1 the differences are +52.1 and +137.1 ($p<10^{-6}$), and at cd=3 they are −17.1 (ns) and −42.8 (Appendix A.5). The contrast between cd=0 and cd=1 also holds on three larger Taillard sizes and three other JSSP benchmark sets, for candidate lists of 10 to 100 moves, and for step budgets up to 5000 (Appendices A.11, A.12, A.9, A.8).

Why the sign changes. Descent improves the solution whichever candidate is chosen; the scorer adds value only if it ranks the candidates of a state correctly. Trained on δ_1, the RF ranks candidates by their outcome after one descent step (Spearman correlation with the true δ_1 among the candidates of a state: +0.46; Appendix D), so it helps most at cd=1. The same target favors moves that change the schedule least, because descent removes the makespan increase they cause. In the LO runs, descent returns to the makespan before the move after 95.2% of the RF's steps and 79.1% of random selection's. The RF thus stops exploring, while random selection keeps perturbing, as the perturbation step of ILS does. The greedy selector, which also prefers the least disruptive move, falls below random selection at LO as well (−86.4). With null moves kept, this preference is extreme: the RF chooses a null move in 57% of its steps and falls below random selection already at cd=3 (Appendix A.4).

Each selector at its best commit depth. The last two rows of Table 1 give the absolute improvements. Random selection is best with descent to a local optimum (447.2), which is ILS with random perturbations; the RF is best at LO or cd=3 (321.5). Compared at their best settings, random selection is ahead by 125.6. The greedy selector, which evaluates all 30 candidates exactly, beats the RF at every depth. The RF's advantage at cd=1 is significant, but random selection is weak at that depth.

Table 2: **Learned-minus-random difference in six published methods** (positive = learned better) at $\mathrm{cd}{=}0$, the original loop, and with each selector at its best descent setting, chosen on the test instances with the same step budget for both. Units: percentage points (pp) of optimality gap (TSP, CVRP, JSSP), of approximation ratio (Max-Cut), and of successful trials (SAT; 100 formulas, 25 trials each); for MIStar, makespan improvement over the initial schedule (makespan units), relative to one random move per step. TSP/CVRP: traveling salesman / capacitated vehicle routing problem; ER/BA: Erdős–Rényi/Barabási–Albert graphs; FJSP: flexible job shop; SD2: a synthetic FJSP instance set used by MIStar; ns: not significant. Settings and reproduction checks: Appendix F.

Method	Problem (test set)	at $\mathrm{cd}{=}0$	best setting
DACT (Ma et al., 2021)	TSP-50 (128)	$+23.1$	-0.03 ns
	TSP-200, TSP-100 model ($128{\times}5$)	$+16.7$	-0.16
NeuOpt (Ma et al., 2023)	CVRP-50 (1000)	$+180.3$	$+4.3$
	CVRP-200 ($256{\times}3$ seeds)	$+416.2$	$+10.9$
ECO-DQN (Barrett et al., 2020)	Max-Cut ER/BA, 40–200 (100 each)	$+69$ to $+85$	$+1.1$ to $+2.5$
L2S (Zhang et al., 2024)	JSSP ta $15{\times}15$ ($10{\times}3$ seeds)	$+2.48$	-1.65
MIStar (Wang et al., 2025)	FJSP SD2 $10{\times}5$ ($10{\times}3$ seeds)	$+19.4$	$+7.7$
SAT (Yolcu & Póczos, 2019)	3-SAT, our models, 50 var., 213 cl. (100)	$+57.2$	-12.6

4 Published Methods at Their Best Descent Setting

We add descent after each move to six published learned local-search methods. We use the authors' code, released checkpoints, test instances, and acceptance criteria; for SAT no checkpoint is released, so we trained models with the authors' code. Appendix F checks our $\mathrm{cd}{=}0$ loop against the published results and lists the remaining differences. The random baseline uses the same loop, descent procedures, and step budget. Table 2 compares the learned-minus-random difference at $\mathrm{cd}{=}0$ (the original loop) with the difference when each selector uses its best descent setting.

Advantages shrink or reverse. In every method, descent helps random selection far more than the learned selector, because a random move followed by descent is an ILS step (perturbation plus local search). In the original loop ($\mathrm{cd}{=}0$) the learned selector is far ahead; with each at its best setting, the advantage is smaller by a factor of 42 in NeuOpt and 34 to 69 in ECO-DQN, not significant in DACT, and reversed in L2S and in our SAT models. On larger instances the pattern is the same: on TSP-200 (with the released TSP-100 checkpoint and the settings the authors recommend for larger instances) the DACT advantage reverses, and on CVRP-200 NeuOpt keeps 2.6% of its $\mathrm{cd}{=}0$ advantage, which is still larger in absolute terms than on CVRP-50 (10.9 vs 4.3 pp). In L2S (Appendix F), one descent step after each move lowers the optimality gap on Taillard $15{\times}15$ at 5000 steps from 6.75% (our run of the original loop; the authors' repository stores 5.49% for this checkpoint) to 4.67%.

A simple baseline with descent. Random selection with the same descent is a standard iterated local search, and for DACT and L2S it is as good as or better than the original loop at equal wall-clock time. On TSP-100 it reaches a 0.46% optimality gap, against 1.27–1.45% for DACT in our reproduction of its loop (Appendix F.2; we use no test-time augmentation, and the authors also report results with eight augmented copies of each instance). On Taillard $15{\times}15$ it reaches 2.52% after 5000 steps, in less time than L2S needs to reach 6.75%. On the four larger Taillard sets up to $50{\times}15$ it is ahead of L2S with its best descent setting at equal wall-clock time by 1.2 to 2.1 pp on three sets and tied on $50{\times}15$ (Appendix F.6). In MIStar, random selection that evaluates the same 50 candidates is close to the published policy (see below). These results do not contradict the published results, which were obtained without descent; they show that once descent is part of the loop, the relevant baseline changes.

Plain descent undoes the selected move. Learned policies often choose worsening moves to escape local optima, and plain descent often reverses them; it reverses random moves as often or more often. In DACT, the solution after descent equals the solution before the move in 96–100% of steps (TSP-50 to TSP-200). In ECO-DQN the first descent flip reverses the policy's flip in 26–76% of steps (random selection: 75–97%), and in NeuOpt (CVRP-50) and in L2S the rates with plain descent are 23–30% and 36–78% (Appendix F). A tabu restriction removes these reversals and makes descent useful to the learned policies (most of their best settings in Table 2 use it): on TSP-200 the released DACT policy with tabu-restricted descent to a local optimum reaches 1.66% after 500 steps

(27 s), where its original loop is at 8.19% at the same time and at 5.99% after 5000 steps (182 s; Appendix F.2). The restriction helps random selection even more (1.11% at 27 s; Appendix F.2). Descent costs time: in our fine-tuning code it adds about 6% to the time of a NeuOpt policy step, and descent to a local optimum makes a DACT step take 2.0 to 2.8 times as long, so Section 5 compares methods at equal wall-clock time.

Matching the number of evaluated candidates. The released MIStar checkpoints for 10×5, 15×10, and 20×10 instances assign almost the same probability to every feasible move (in the authors' evaluation code, every move probability is within 0.02% of $1/n$, where n is the number of feasible moves), because their tanh layers saturate. At each step MIStar samples up to 50 distinct candidates, evaluates each exactly, and applies the best, whereas the random baseline of Table 2 applies a single random move. A random selector that samples and evaluates 50 candidates in the same way draws from the same distribution as the published policy. Over 30 runs it trails MIStar by 0.3 makespan units at $\mathrm{cd}{=}0$ ($p{=}0.79$) and by at most 2.9 at larger depths ($p{\geq}0.02$, uncorrected for multiple comparisons), against 19.4 for a single random move (Appendix G). Learned-versus-random comparisons should therefore let both selectors compute the exact objective of the same number of candidates per step.

5 Fine-Tuning Policies with Descent in the Training Loop

The published policies were trained without descent, so they may not suit a loop that runs descent after each move. We fine-tune five of the published policies (all except SAT, whose models we trained ourselves) with descent in their training loop: the policy acts, descent runs (with the tabu restriction where the published policy is best with it), and the reward and next state are computed on the solution after descent. A control run fine-tunes the same checkpoint with the same data, seeds, learning rate, and number of updates, without descent. We use the authors' training code and algorithm (PPO for DACT, NeuOpt, and MIStar; DQN for ECO-DQN; REINFORCE for L2S) and train for 1.6 to 8.7 hours per run on one GPU, with two seeds per method. We evaluate the final checkpoint and the best-validation checkpoint, which is chosen on validation instances disjoint from the test instances by a rule fixed before evaluation. Each checkpoint runs with the descent used in its fine-tuning and is compared at equal wall-clock time with four baselines: the published policy at $\mathrm{cd}{=}0$ given more steps, the published policy with its best descent setting, random selection with its best descent setting, and the control with its best descent setting (Appendix H).

Comparison before fine-tuning. Before fine-tuning, we compare the published policy with random selection, each at its best descent setting and at equal wall-clock time (the *pre-fine-tuning comparison*). If random selection is at least as good, the policy's choices add nothing once descent runs, and we expect fine-tuning not to make the policy better than random selection with descent. If the policy is ahead, its choices matter after descent, and fine-tuning can adapt them to the descent. Before starting the corresponding runs, we recorded two predictions based on this comparison run with the same number of steps rather than equal time (Appendix H.2; Table 3 gives the equal-time values). For L2S (difference with the same number of steps: -1.65) we predicted no gain over random selection with descent. For ECO-DQN on Barabási–Albert (BA) graphs with 40 and 60 vertices (differences with the same number of steps: $+1.34$ and $+1.07$), a graph family not used in the other fine-tuning runs, we predicted a gain.

Results. The results in Table 3 agree with the pre-fine-tuning comparison. Where the published policy is ahead (ECO-DQN, NeuOpt), the fine-tuned policy beats all four baselines; where it is not (DACT, MIStar, L2S), it does not beat random selection with descent.

- *ECO-DQN.* The best-validation checkpoints beat all four baselines on ER40, ER60, and ER200 graphs for both seeds. Against the published policy with its best descent setting, they remove 45 to 87% of the remaining gap to the best known cut (Appendix H.3). On ER500 graphs, larger than any used in training, they beat the published policy and random selection but not consistently the control. On the GSET benchmark used by ECO-DQN (G1–G10, 800 vertices), the fine-tuned policy beats the published policy in its original loop by 0.38 and 0.54 pp ($p{\leq}0.004$) and random selection with descent by 2.0 to 2.1 pp, as we had also predicted before the GSET runs; contrary to that prediction, it is only tied with the published policy with its best descent setting ($+0.12$ to $+0.17$ pp, $p{\geq}0.13$; Appendix H.4). On BA graphs, where we recorded the prediction before the

Table 3: **Fine-tuning with descent in the training loop.** Before FT: published policy minus random selection before fine-tuning, each at its best descent setting, at equal wall-clock time (Appendix H.2). Other columns: fine-tuned policy (best-validation checkpoint) minus each baseline at equal wall-clock time, seed 1 / seed 2; positive = fine-tuned better. pub.: published policy; desc.: best descent setting. Units as in Table 2; for MIStar, optimality gap in pp, and random selection evaluates the same 50 candidates per step. All ECO-DQN, NeuOpt, and L2S entries have $p<0.05$; for MIStar only $+0.41$ and -0.54 do (per-cell statistics and final checkpoints: Appendix H). *Outcome predicted before the runs. ‡With the descent used in fine-tuning, random selection rarely reaches a capacity-feasible solution on CVRP-100, and the random column compares at the same number of steps; with a descent that penalizes capacity excess (Appendix F.3), the published policy is ahead by 7.9 pp.

			fine-tuned minus …			
Method	Problem	Before FT	pub. cd=0	pub. + desc.	random + desc.	control + desc.
ECO-DQN	Max-Cut ER40	+1.5	+0.20 / +0.17	+0.10 / +0.08	+1.49 / +1.54	+0.12 / +0.11
	Max-Cut ER60	+1.4	+0.62 / +0.67	+0.61 / +0.63	+1.82 / +1.83	+0.29 / +0.29
	Max-Cut ER200	+2.1	+0.95 / +0.64	+0.96 / +0.83	+2.89 / +2.96	+0.71 / +0.44
	Max-Cut BA40*	+1.4	+0.58 / +0.62	+0.58 / +0.59	+1.76 / +1.85	+0.34 / +0.28
	Max-Cut BA60*	+1.1	+1.46 / +1.49	+1.64 / +1.69	+2.43 / +2.56	+1.39 / +1.14
NeuOpt	CVRP-100	+126‡	+1.05 / +0.90	+0.34 / +0.19	+128‡ / +128‡	+0.42 / +0.33
DACT	TSP-50	−0.05	−0.03 / −0.04	−0.04 / −0.07	−0.09 / −0.11	−0.06 / −0.06
	TSP-100	−0.35	+0.15 / +0.83	−0.25 / +0.26	−0.67 / −0.13	−0.17 / +0.32
MIStar	FJSP SD2 10×5	−0.34	+0.13 / −0.30	+0.41 / −0.01	−0.13 / −0.54	+0.28 / −0.01
L2S	JSSP ta 15×15*	−2.2	+4.06 / +3.87	+1.81 / +1.62	−0.41 / −0.64	+1.06 / +1.06

runs, the fine-tuned policy beats all four baselines for both sizes, both seeds, and both checkpoints, by 0.3 to 2.6 pp (all $p<10^{-7}$). We also trained ECO-DQN from scratch with descent in the training loop (the authors' schedule, two seeds). This network beats the published policy with its best descent setting on ER60 ($+0.52$ to $+0.65$ pp, $p<10^{-14}$) and ties it on ER40 (-0.03 to $+0.01$ pp, $p\geq 0.41$); on both sizes it beats a network trained from scratch without descent and random selection with descent (Appendix H.5).

- *NeuOpt.* The best-validation checkpoints beat all four baselines for both seeds and reduce the optimality gap of the published policy with its best descent setting by 11 to 19%.
- *ECO-DQN and NeuOpt, final checkpoints.* In these two methods the checkpoint rule (fixed before evaluation) matters only for the comparison with the published policy. On the instance sizes used for fine-tuning, the final checkpoints also beat random selection with descent and the control in every ECO-DQN and NeuOpt run. Against the published policy with its best descent setting, they are tied on ER40 and ahead by only 0.12 pp ($p=0.09$) for the second NeuOpt seed (Appendix H).
- *DACT.* Fine-tuning beats the published policy with descent only on TSP-100 for one seed. Random selection with descent, which is cheaper per step, stays better at equal wall-clock time for both seeds and both sizes.
- *MIStar.* The best-validation checkpoints are taken after 10 of 80 episodes, while the policy is still almost uniform. They are tied with the published policy at equal wall-clock time (one of four comparisons significant) and do not beat a random selector that evaluates the same 50 candidates. The final checkpoints are worse than the published policy, the 50-candidate random selector, and a single random move with descent.
- *L2S.* We fine-tune on 10×10 instances and test on Taillard 15×15. As predicted, the fine-tuned policy is worse than random selection with descent at equal wall-clock time: by 0.41 and 0.64 pp with the best-validation checkpoints and by 0.56 and 0.64 pp with the final ones (all $p<0.05$). It is 1.6 to 1.8 pp better than the published policy with its best descent setting, and with the same number of steps it ties random selection with descent (differences of 0.02 to 0.36 pp in its favor, $p\geq 0.12$).

What fine-tuning changes when it helps. With descent in the training loop, the reward is computed on the solution after descent, not after the move. The fine-tuned policies choose moves that descent turns into improvements. In ECO-DQN the cut improvement from descent after each policy flip rises from 1.16, 0.85, and 1.38 per step for the published policy with the same descent

to 1.33–1.52, 1.48–1.58, and 2.20–2.32 (ER40, ER60, ER200; two seeds). On ER60 and ER200 the share of flips that lower the cut rises from 24–31% to 46–54%; the control, fine-tuned without descent, stays close to the published values. The fine-tuned policy thus chooses cut-lowering flips more often than the published policy but far less often than random selection, whose flips lower the cut in 90–98% of steps (ER40 to ER200); this resembles the ILS guideline that a perturbation should be neither so weak that local search undoes it nor so strong that it approaches a random restart (Lourenço et al., 2003). On ER40 the share does not rise (49% to 42–47%), but the improvement from descent does. In NeuOpt (CVRP-100), plain descent undoes 17% of the published policy's non-null moves, 11–19% of the control's, and 7–8% of the fine-tuned policy's.

When fine-tuning does not beat random selection. In DACT, where random selection with descent is already as good, fine-tuning with descent raises the normalized entropy (entropy divided by that of the uniform distribution). In the three runs that do not improve, it rises from 0.25–0.32 to a peak of 0.70–0.99 and ends at 0.41–0.76; in the run that improves (TSP-100, second seed), it stays near 0.3. In L2S the policy entropy on validation states (not normalized) rises from 0.39 to about 1.0, and the validation gap reaches its final level within the first 10 of 480 training batches. The fine-tuned L2S policy is better than the published one and has higher entropy, but each of its steps takes 2.4 times as long as a random step with the same descent. In MIStar, whose published policy is almost uniform, fine-tuning lowers the entropy instead: the normalized entropy falls from 1.0 to 0.02–0.05 within 25 to 50 of the 80 episodes, and the validation makespan rises from 370 to 392–396. A sharper policy yields fewer distinct moves among the 50 candidates sampled per step, so exact evaluation helps less. We conjecture that in all three methods the learning signal is weak because descent, not the policy's choice, determines the outcome.

6 Related Work

Learned local search. Learned improvement heuristics are policies trained with reinforcement learning inside a fixed search loop: for routing (Wu et al., 2022; Ma et al., 2021; 2023; Hottung & Tierney, 2020), scheduling (Zhang et al., 2024; Wang et al., 2025), Max-Cut (Barrett et al., 2020), SAT (Yolcu & Póczos, 2019), and program rewriting (Chen & Tian, 2019). Construction heuristics (Vinyals et al., 2015; Kool et al., 2019; Kwon et al., 2020; Dai et al., 2017) and learned dispatching rules (Zhang et al., 2020; Park et al., 2021) are often combined with local-search post-processing. Surveys: Bengio et al. (2021); Cappart et al. (2023); Mazyavkina et al. (2021).

Iterated local search and memetic algorithms. With random-walk acceptance, our loop is an ILS in which the selector picks the perturbation and the local search is cut off after cd steps; ILS and variable neighborhood search (VNS) (Lourenço et al., 2003; Hansen & Mladenović, 2001) descend to a local optimum instead (for JSSP, Lourenço, 1995). Memetic algorithms tune the depth of local search after each variation step, the analog of our commit depth, for solution quality (Ishibuchi et al., 2003; Krasnogor & Smith, 2005); we ask how the depth changes a learned selector's value over random selection. That descent turns random perturbations into an effective search is the premise of ILS; what we add is that descent shrinks or reverses learned-versus-random comparisons in published learned methods. Our tabu restriction is a standard tabu-search rule (Glover, 1989), applied only during the descent that follows a selected move.

Evaluation of learned heuristics. Boether et al. (2022) and Xia et al. (2024) show that learned components can be replaced by simple or untrained ones inside tree search or with heatmap-guided search, and Sun & Yang (2023) report maximum independent set (MIS) results without local search, noting that with it all models would perform similarly. We make the loop parameter explicit, show that the comparison can reverse, and give a comparison, made before fine-tuning, that agreed with the outcome of fine-tuning with descent in all five methods we fine-tuned. Our emphasis on controlled comparisons follows Hooker (1995).

7 Discussion

Recommendations. (i) Report learned-versus-random comparisons at $\mathrm{cd}{=}0$, $\mathrm{cd}{=}1$, and descent to a local optimum, with plain and tabu-restricted descent where descent can undo the selected move,

and compare each selector at its best setting. (ii) Let the random baseline compute the exact objective of as many candidates per step as the learned selector. (iii) Before fine-tuning a policy with descent, check whether it beats random selection with descent at equal wall-clock time; if not, the fine-tuned policy is unlikely to beat random selection with descent. In our experiments this comparison, which needs only evaluation runs, separated the two methods whose fine-tuned policy beat random selection with descent from the three whose fine-tuned policy did not. (iv) One-step targets computed after descent, such as δ_1, can favor moves that change the solution least; check how much the moves a scorer selects change the solution.

Limitations. Our fine-tuning runs are short (at most 9 GPU-hours per run, two seeds per method) and, except for the ECO-DQN runs from scratch on ER40 and ER60, start from released checkpoints. The fine-tuning instances are small (TSP-50/100, CVRP-100, graphs of up to 200 vertices, 10×10 JSSP, 10×5 FJSP); we evaluate larger instances (TSP-200, CVRP-200, ER500, GSET, Taillard up to 50×15) but do not fine-tune on them. The gains of fine-tuning in ECO-DQN and NeuOpt over the strongest baseline are small in absolute terms (0.1 to 1.4 pp and 0.2 to 0.3 pp), although in ECO-DQN they remove 45 to 87% of the published policy's remaining gap. Our JSSP loop is a controlled testbed, not a competitive solver: after 5000 steps the RF selector at $\mathrm{cd}{=}1$ is 13.1% above optimal, while tabu search over $\mathcal{N}_5$ is near-optimal on these instances (Nowicki & Smutnicki, 1996). Our SAT models are weaker than the authors'.

8 Conclusion

When both selectors run with the descent used in ILS and evaluate the same number of candidates per step, descent helps random selection far more than learned move selection: the advantages of five of the six published learned local-search methods we tested shrink by large factors or disappear, and in MIStar the advantage over a single random move comes from evaluating 50 candidates. Two effects contribute: a random move followed by descent is an ILS step, and, in our JSSP study, a scorer trained on one-step post-descent outcomes prefers moves that change the solution least. Where the published policy is still ahead of random selection with descent, fine-tuning with descent in the training loop makes the policy select moves from which descent finds larger improvements; the fine-tuned policy (best-validation checkpoint) then beat all four baselines on the instance sizes used for fine-tuning; where the published policy was not ahead, the fine-tuned policy did not beat random selection with descent. For ECO-DQN on BA graphs and for L2S we recorded these outcomes as predictions before the runs; a third prediction, on GSET, held only in part. We see these results as an initial study; open questions include a theory of when learned move selection beats random selection with descent, training with descent from scratch at scale, and fine-tuning on larger instances.

AI use statement

In this work, we used generative AI tools (LLM-based assistants) to write the experimental and analysis code and to discuss and refine the experimental design. We also used these tools to draft and edit the manuscript text, to produce figures, to search the literature, and to format references. We did not use generative AI tools to generate synthetic data, to develop theoretical models, to formulate or prove mathematical claims, to propose or refine hypotheses, or to interpret results. Translation, dataset cleaning or reformatting, and qualitative or thematic data analysis do not apply (the manuscript was written in English from the outset). We have reviewed all AI-assisted work: we verified all AI-assisted code and all reported numbers, checked every citation against its source, and read all AI-written and AI-edited text. We take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

Reproducibility statement

All experiments use public benchmark instances (Taillard, SWV, ABZ, and YN for JSSP), the test sets released with the published methods (DACT, NeuOpt, ECO-DQN, L2S, MIStar), or instances produced by generator scripts (random 3-SAT and the other problems of the appendix); all runs use fixed random seeds (some baseline runs derive seeds from Python string hashes, so reruns can differ slightly). The loop, the commit depth, and the comparisons are specified in Section 2, and the JSSP protocol in Sections 2 and 3. The protocols and reproduction checks for the published methods are

in Appendix F; the fine-tuning protocol, the checkpoint rule, and the equal-wall-clock evaluation are in Appendix H. For the published methods we use the authors' code and released checkpoints without modification; for the learned SAT heuristic no checkpoint is released, and we trained models with the authors' code. Fine-tuning and evaluation scripts and all result files will be released upon acceptance.

REFERENCES

Thomas D. Barrett, William R. Clements, Jakob N. Foerster, and A. I. Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Maximilian Boether, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *International Conference on Learning Representations (ICLR)*, 2022.

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

Marco Caserta, Silvia Schwarze, and Stefan Voß. A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research*, 219 (1):96–104, 2012.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 32, pp. 6278–6289, 2019.

Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, volume 30, pp. 6348–6358, 2017.

Fred Glover. Tabu search—part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

Pierre Hansen and Nenad Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

John N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42, 1995.

André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *24th European Conference on Artificial Intelligence (ECAI)*, 2020.

Hisao Ishibuchi, Tadashi Yoshida, and Tadahiko Murata. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation*, 7(2):204–223, 2003.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.

Natalio Krasnogor and Jim Smith. A tutorial for competent memetic algorithms: Model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of Metaheuristics*, pp. 321–353. Kluwer Academic Publishers, 2003.

Helena Ramalhinho Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83(2):347–364, 1995.

Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-Opt. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. ScheduleNet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021.

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

Paolo Toth and Daniele Vigo (eds.). *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2 edition, 2014.

Peter J. M. van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Jiaqi Wang, Zhiguang Cao, Peng Zhao, Rui Cao, Yubin Xiao, Yuan Jiang, and You Zhou. Learning memory-enhanced improvement heuristics for flexible job shop scheduling. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Runqing Wang, Gang Wang, Jian Sun, Fang Deng, and Jie Chen. Flexible job shop scheduling via dual attention network-based reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 35(3):3091–3102, 2024.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 33 (9):5057–5069, 2022.

Yifan Xia, Xianliang Yang, Zichuan Liu, Zhihao Liu, Lei Song, and Jiang Bian. Position: Rethinking post-hoc search-based neural approaches for solving large-scale traveling salesman problems. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Emre Yolcu and Barnabás Póczos. Learning local search heuristics for Boolean satisfiability. In *Advances in Neural Information Processing Systems*, volume 32, pp. 7990–8001, 2019.

ChaoYong Zhang, PeiGen Li, ZaiLin Guan, and YunQing Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229–3242, 2007.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Cong Zhang, Zhiguang Cao, Wen Song, Yaoxin Wu, and Jie Zhang. Deep reinforcement learning guided improvement heuristic for job shop scheduling. In *International Conference on Learning Representations (ICLR)*, 2024.

A SEARCH LOOP AND JSSP ROBUSTNESS CHECKS

Unless stated otherwise, all JSSP experiments with random compound moves in this appendix use the main setting of Section 2: every scorer is trained on data without null moves, and null moves are removed from every candidate list. Appendix A.4 reports the original setting, with null moves in the training data and in the candidate lists.

A.1 SEARCH LOOP

Algorithm 1 Local search with a learned scorer and commit depth cd. The only difference from the conventional loop is the optional descent on lines 4–5.

Require: state x_t, candidate list $\mathcal{C}(x_t)$, scorer f, descent operator D, commit depth $\mathrm{cd} \in \mathbb{N}$

1: $m_t \leftarrow \arg\max_{m \in \mathcal{C}(x_t)} f(x_t, m)$
2: $y_t \leftarrow x_t \oplus m_t$
3: $x_{t+1} \leftarrow y_t$
4: **if** $\mathrm{cd} \geq 1$ **then**
5: $\quad x_{t+1} \leftarrow D^{\mathrm{cd}}(x_{t+1})$ ▷ post-move descent
6: **end if**

A.2 SUMMARY OF ROBUSTNESS CHECKS

Table 4 lists the robustness checks of the JSSP result and of the side analysis of the saturation ratio (Appendix E), with the condition varied and the outcome.

Table 4: Robustness checks. JSSP entries are learned-minus-random differences in the improvement of the incumbent. JSSP rows use the main setting (no null moves in the training data or the candidate lists) unless the row says otherwise.

Condition	Variation	Outcome	Where
Candidate-list size	$C \in \{10, 30, 60, 100\}$	JSSP cd=1 difference $+182$ to $+223$; peak at cd=1 for every C	App. A.9
Step budget	100 to 5000 steps (2 instances $\times$ 2 seeds)	cd=1 difference grows from $+302$ to $+555$; cd=3 difference small, negative only at 500 steps	App. A.8
Descent budget	300 descent steps at every cd	cd=1 difference $+326.1$	App. A.7
Acceptance	non-worsening; simulated annealing	cd=0 difference ns; cd=1 difference $+52$, $+137$ ($p<10^{-6}$)	App. A.5
Null moves	kept in the training data and the candidate lists	difference 0 / $+165$ / $+10$ / -102 at cd=0–3; cd=1 gain only under random-walk acceptance; ratio still <1	App. A.4, A.5
Instance sets	Taillard 15×15 to 30×15; SWV, ABZ, YN	difference $\approx$0 at cd=0 and positive at cd=1 in every set	App. A.11, A.12
Instances per domain	30 instances, five non-JSSP domains	cd=0 regime in all five	App. C.1
Training target and seed	δ_0, δ_1, δ_2; three seeds in three domains	JSSP peak at cd=1 for every target and seed; peaks unchanged across seeds	App. A.6
Scorer model	RF, MLP, RankNet, GNN	JSSP difference $\approx$0 at cd=0 for all; GNN also positive at cd=1–3	App. D, A.3
Descent operator	two implementations of RCPSP, CVRP, BRP	only RCPSP changes regime, as the ratio predicts	App. E.7
Move size	28 families, 368 (family, move size) points	ratio falls with move size in $27/28$; transition ratio family-specific; threshold of 1 agrees with the observed regime at 46.5% of points (majority regime: 55.7%)	App. E.2
Ratio sample size	50, 100, 200 pairs	JSSP correct in 30.6%, 55.5%, 58.3% of draws (synthetic pool; Taillard sample: 52.5%, 82.9%, 91.8%); seven other families 100%	App. E.6

A.3 STRONGER BASELINES ON JSSP: FULL STATISTICS

We compare the RF with three other selectors under the 5-fold CV of Section 3 (n=50): a greedy baseline that evaluates every candidate's immediate cost exactly and picks the best, an SA-type selector (greedy candidate with Metropolis acceptance, else a random one), and a graph neural network (GNN; 3-layer edge-conditioned message passing, hidden size 64, trained on δ_1 with the

RF's CV folds). Table 5 gives the scorer-minus-baseline differences, and Table 6 adds Cohen's d, the paired-test p-value, and the SA-type rows.

Table 5: Scorer − baseline difference in incumbent improvement (positive = scorer better) on JSSP ta01–10, 5-fold CV (n=50); scorers trained without null moves, null moves removed from the candidate lists. Stars as in Table 21; "ns"=p>0.05.

Scorer	Baseline	cd=0	cd=1	cd=2	cd=3
RF^{δ_1}	random	+0.4 ns	+220.2 ***	+198.0 ***	+39.6 **
RF^{δ_1}	greedy	−221.5 ***	−113.4 ***	−62.4 ***	−55.3 ***
GNN	random	+0.5 ns	+249.2 ***	+206.2 ***	+37.6 **
GNN	greedy	−221.4 ***	−84.4 ***	−54.2 ***	−57.3 ***

The greedy baseline is above both scorers at every commit depth from 0 to 3 (all $p<10^{-3}$). Its own gain over random selection is +221.9 at cd=0, +333.6 at cd=1, and +260.4 at cd=2, so greedy selection also gains more over random selection at cd=1 than at cd=0. The GNN's difference from random selection is similar to the RF's at every depth and slightly larger at cd=1 (+249.2 vs. +220.2). With descent to a local optimum after each move (same runs), the RF is below random selection (−125.6), below greedy (−39.2, p=0.0015), and below the SA-type selector (−23.7, p=0.05), and greedy is also below random selection (−86.4, $p<10^{-8}$).

Table 6: Full statistics for Table 5. For each scorer–baseline pair, the first row gives the mean difference and the second row Cohen's d with the p-value in parentheses.

Scorer	Baseline	stat	cd=0	cd=1	cd=2	cd=3
RF	random	mean	+0.4	+220.2	+198.0	+39.6
		d (p)	0.24 (0.098)	2.52 ($<10^{-22}$)	2.01 ($<10^{-18}$)	0.46 (0.0022)
	greedy	mean	−221.5	−113.4	−62.4	−55.3
		d (p)	−2.87 ($<10^{-24}$)	−1.38 ($<10^{-12}$)	−0.73 ($<10^{-5}$)	−0.73 ($<10^{-5}$)
	SA	mean	−215.8	−117.8	−51.7	−24.8
		d (p)	−2.87 ($<10^{-24}$)	−1.50 ($<10^{-13}$)	−0.72 ($<10^{-5}$)	−0.35 (0.017)
GNN	random	mean	+0.5	+249.2	+206.2	+37.6
		d (p)	0.26 (0.075)	2.66 ($<10^{-23}$)	2.32 ($<10^{-20}$)	0.38 (0.0095)
	greedy	mean	−221.4	−84.4	−54.2	−57.3
		d (p)	−2.88 ($<10^{-24}$)	−1.16 ($<10^{-10}$)	−0.62 ($<10^{-4}$)	−0.58 ($<10^{-3}$)
	SA	mean	−215.7	−88.7	−43.5	−26.8
		d (p)	−2.86 ($<10^{-24}$)	−1.20 ($<10^{-10}$)	−0.55 ($<10^{-3}$)	−0.35 (0.016)

The CV training data for each GNN were collected on the 8 training instances of each fold from random compound-move candidates (none of them a null move) labeled with the δ_1 target; each GNN was trained for 30 epochs with gradient clipping at 5.0. The architecture uses edge-conditioned message MLPs, sum aggregation, and residual updates with layer normalization. The greedy baseline picks the candidate with the smallest makespan right after the move (largest δ_0). The SA-type selector applies the same greedy candidate if it passes a Metropolis test under geometric cooling ($T_t = 0.99^t\, T_0$, T_0 = 5% of the starting makespan) and a random candidate otherwise, so the loop always moves; it is not a full simulated annealing.

JSSP learned-minus-random difference: full statistics. Table 7 reports 95% confidence intervals, paired t-tests, and Cohen's d for the learned-minus-random difference at each commit depth under 5-fold cross-validation on Taillard ta01–10 (n=50 runs: 5 folds × 2 test instances × 5 seeds).

The cd=1 and cd=2 differences are significant with large effect sizes (d=2.52 and 2.01; positive in all 50 runs). The cd=3 difference is smaller (d=0.46, p=0.0022; positive in 33 of 50 runs). With descent to a local optimum the difference is negative (d=−1.60; negative in 47 of 50 runs), 346 below the cd=1 difference of the same runs. Each of the five folds shows a cd=1 difference between +158.7 and +250.1 (at cd=2, between +148.2 and +239.8), so the effect is stable across folds.

Table 7: JSSP learned-minus-random difference: 5-fold CV statistical summary. LO: descent to a local optimum after each move (at most 100 descent steps), in the same runs and seeds.

cd	Mean diff.	95% CI	t	p	Cohen's d
0	$+0.4$	$[-0.1, +0.9]$	1.69	0.098	0.24
1	$+220.2$	$[+195.3, +245.0]$	17.79	$<10^{-22}$	2.52
2	$+198.0$	$[+170.1, +225.9]$	14.25	$<10^{-18}$	2.01
3	$+39.6$	$[+15.0, +64.1]$	3.24	0.0022	0.46
LO	-125.6	$[-147.9, -103.4]$	-11.34	$<10^{-14}$	-1.60

A.4 NULL MOVES IN THE JSSP CANDIDATE LISTS

When an elementary SWAP or INSERT would create a cycle, it is skipped, so a compound move can leave the solution unchanged (a null move). In the main setting, null moves are excluded from both the training data and the candidate lists: the scorer's training data contain none (a sampled null move is replaced by another random compound move, at most 20 times), and every candidate list is refilled with moves that change the solution. This appendix compares the main setting with the original setting, in which the RF is trained on data that contain null moves and the candidate lists keep them.

In the original training data 38% of candidates are null moves; their mean δ_1 is $+70.7$ (one descent step from an undisturbed state), against -3.7 for the other moves, so the RF learns to prefer them: it applies a null move in 82% of steps at $\mathrm{cd}{=}0$ and in about 57% at $\mathrm{cd}{\geq}1$, while null moves make up 35–39% of the candidate lists. At $\mathrm{cd}{=}1$ a null move followed by one descent step is just one descent step from the current solution. With deeper descent the learned loop repeatedly applies a null move and descends, so it stops perturbing and stalls near a local optimum, whereas random selection keeps perturbing, as the kick (perturbation) step of iterated local search does.

Table 8 compares the two settings on the same instances and seeds. The $\mathrm{cd}{=}0$ difference is ≈ 0 in both, and the $\mathrm{cd}{=}1$ difference is large and significant in both ($+164.7$ and $+220.2$). With null moves, the difference is not significant at $\mathrm{cd}{=}2$ ($+9.9$) and negative at $\mathrm{cd}{=}3$ (-102.3); without them, it stays positive at both depths ($+198.0$ and $+39.6$). The loss of the advantage at $\mathrm{cd}{=}2$ and its reversal at $\mathrm{cd}{=}3$ therefore come from the preference for null moves. With descent to a local optimum after each move, the difference is negative in both settings (-113.7 and -125.6). Without null moves the scorer still prefers moves after which descent returns to the previous makespan: in the local-optimum runs, the makespan after the move and the descent equals the makespan before the move in 95.2% of the learned scorer's steps, against 79.1% for random selection (greedy 94.3%, SA-type 94.4%; with null moves: learned 95.8%, greedy 97.0%, random 86.8%). Such steps do not change the makespan, so deep descent stalls the learned loop in both settings; greedy selection stalls in the same way and is also below random selection there (Appendix A.3). With null moves, the $\mathrm{cd}{=}1$ gain also depends on random-walk acceptance (Appendix A.5).

The saturation ratio of JSSP is 0.82 on Taillard states without null moves in the move sample (Table 26) and 0.72 with them; on the synthetic pool of Appendix E.6 it is 0.94 and 0.81. All four values are below 1, so the predicted regime ($\mathrm{cd}{\geq}1$) is the same.

Table 8: JSSP learned-minus-random difference with null moves kept (RF trained on data with null moves, original candidate lists) and excluded (main setting: RF trained without null moves, null moves removed from the candidate lists); 5-fold CV, $n{=}50$ per cell, improvement of the incumbent within 100 steps, same instances and seeds. LO: descent to a local optimum after each move. Null share: fraction of steps in which the learned scorer applies a null move. Stars as in Table 21.

	null moves kept				null moves excluded		
cd	learned	random	diff.	null share	learned	random	diff.
0	0.0	0.1	-0.1 ns	0.82	0.4	0.0	$+0.4$ ns
1	242.0	77.3	$+164.7$ ***	0.57	232.2	12.0	$+220.2$ ***
2	253.4	243.5	$+9.9$ ns	0.57	303.1	105.1	$+198.0$ ***
3	244.2	346.5	-102.3 ***	0.56	320.4	280.8	$+39.6$ **
LO	294.8	408.5	-113.7 ***	0.55	321.5	447.2	-125.6 ***

A.5 ACCEPTANCE CRITERIA IN JSSP

Our loop accepts every selected move after its descent (random-walk acceptance, one of the standard acceptance criteria of iterated local search). We re-ran the JSSP CV of Table 8 with two other acceptance criteria for the post-descent solution: *non-worsening* (keep it only if its makespan is not larger than the current one) and *simulated annealing* (Metropolis rule with temperature $T_t = 0.05\, C_0 \cdot 0.99^t$, where C_0 is the initial makespan and t the step). Candidate generation, the RF, and the seeds are unchanged, so every trace equals the random-walk trace up to the first rejected move. Table 9 reports the differences. With null moves excluded, the learned scorer gains over random selection at $\mathrm{cd}{=}1$ under all three criteria ($+220.2$, $+52.1$, $+137.1$; all $p{<}10^{-6}$) and at $\mathrm{cd}{=}0$ under none. With null moves kept, the $\mathrm{cd}{=}1$ gain appears only under random-walk acceptance, where random selection accepts every worsening move (improvement 77.3 at $\mathrm{cd}{=}1$, against 234.1 under non-worsening acceptance) while the learned scorer mostly applies null moves; under the other two criteria random selection is not hurt in this way and the gain vanishes. Negative differences at $\mathrm{cd}{\geq}2$ appear under every criterion with null moves kept; with them excluded, the only significant negative difference is under simulated annealing at $\mathrm{cd}{=}3$ (-42.8, $p{=}0.0014$).

Table 9: JSSP learned-minus-random difference under three acceptance criteria, with null moves excluded (main setting) or kept (RF trained with null moves, original candidate lists; Appendix A.4); 5-fold CV, $n{=}50$ per cell; improvement of the incumbent within 100 steps; paired t-test, stars as in Table 21.

Acceptance	Null moves	cd=0	cd=1	cd=2	cd=3
Random walk	excluded	+0.4 ns	+220.2 ***	+198.0 ***	+39.6 **
	kept	−0.1 ns	+164.7 ***	+9.9 ns	−102.3 ***
Non-worsening	excluded	+1.8 ns	+52.1 ***	−8.7 ns	−17.1 ns
	kept	−4.8 *	−23.9 ns	−61.8 **	−98.8 ***
Simulated annealing	excluded	−0.3 ns	+137.1 ***	+7.3 ns	−42.8 **
	kept	−0.4 ns	−10.2 ns	−87.1 ***	−123.8 ***

A.6 ROBUSTNESS OF THE PEAK DEPTH AND OF THE RATIO THRESHOLD

Insensitivity to the training target. Table 10 reports the learned-minus-random difference when the same RF architecture is trained with different training targets (δ_0, δ_1, δ_2) on JSSP ta01–10 (single fold, 2 test instances $\times$ 5 seeds). The peak depth remains at $\mathrm{cd}{=}1$ regardless of the training target.

Table 10: JSSP learned-minus-random difference under different training targets. The peak depth is $\mathrm{cd}{=}1$ in all cases. One seed for data collection and training (five search seeds per instance); stability across training seeds is reported in Table 11, whose seed-0 JSSP row is the δ_2 row here (those runs start without descent warm-up).

Training target	cd=0	cd=1	cd=2	peak
δ_0 (immediate)	+0.0	+226.7	+181.5	1
δ_1 (1 descent step)	+0.0	+259.2	+198.5	1
δ_2 (2 descent steps)	−4.4	+421.7	+311.5	1

Seed stability of δ_2-trained scorers. Table 11 reports the learned-minus-random difference and the observed peak commit depth for three independent seeds (data collection and training) of a δ_2-trained scorer in each of three domains. Within each domain, the peak depth is identical across all three seeds (nine runs in total).

Leave-one-family-out cross-validation. Under leave-one-family-out CV, the threshold is refit on the other eleven families as the midpoint of the interval that separates them, and the left-out family's ratio is compared with it; in all 12 cases the left-out family falls on the side of its observed regime, but ten of the twelve families are in the $\mathrm{cd}{=}0$ regime, so this is a weak test. MIS (ratio 1.58) has the smallest margin (refit threshold 1.54, from the interval $(0.82, 2.26)$); the fixed threshold of 1 also puts MIS on the side of its observed regime. Threshold sensitivity is analyzed in Appendix E.5.

Table 11: Seed stability of δ_2-trained scorers. Each row reports the learned-minus-random difference at $\mathrm{cd} \in \{0, 1, 2, 3\}$ for one independent seed (data collection and training); the rightmost column gives the observed peak depth. The JSSP runs start from a random solution without descent warm-up, so, unlike in the main setting, their $\mathrm{cd}{=}0$ difference is not forced to be close to 0.

Domain	seed	cd=0	cd=1	cd=2	cd=3	peak
JSSP	0	−4.4	+421.7	+311.5	+91.3	1
JSSP	1	+8.9	+528.3	+240.1	+12.5	1
JSSP	2	−54.5	+435.3	+217.2	+95.1	1
Knapsack	0	+970.0	+7.7	−15.8	−9.6	0
Knapsack	1	+961.9	+12.6	−33.0	−3.0	0
Knapsack	2	+815.3	−1.2	+12.6	−0.2	0
MaxSAT	0	+10.7	−0.2	−1.4	−0.2	0
MaxSAT	1	+11.1	−0.7	−1.4	−1.4	0
MaxSAT	2	+11.6	+0.9	−0.8	−0.7	0

A.7 JSSP CV AT A MATCHED DESCENT BUDGET

To control for the fact that larger commit depths spend more descent computation per step, we re-ran the JSSP CV ($n{=}50$) with the total descent budget fixed at 300 descent steps for $\mathrm{cd}{\geq}1$: $\mathrm{cd}{=}1$ runs 300 search steps with 1 descent step each; $\mathrm{cd}{=}2$, 150 search steps with 2 each; $\mathrm{cd}{=}3$, 100 search steps with 3 each (300 descent steps in every case). The $\mathrm{cd}{=}0$ setting runs 300 search steps with no descent and serves as the no-descent baseline.

Table 12: JSSP CV at a matched descent budget ($n{=}50$; 300 descent steps for $\mathrm{cd}{\geq}1$; $\mathrm{cd}{=}0$ is the no-descent baseline): scorer − baseline difference.

Scorer	Baseline	cd=0	cd=1	cd=2	cd=3
RF^{δ_1}	random	+0.4 ns	**+326.1** ***	+200.0 ***	+39.6 **
RF^{δ_1}	greedy	−367.7 ***	−108.3 ***	−63.0 ***	−55.3 ***

The $\mathrm{cd}{=}1$ difference grows under the matched budget (difference $+326.1$, $d{=}3.88$, $p{<}10^{-30}$), but the learned scorer stays below the exact-evaluation greedy baseline ($\Delta{=}-108.3$, $p{<}10^{-13}$). Under a fixed-step budget, the random selector's absolute improvement rises with cd because each step performs more descent; the $\mathrm{cd}{=}1$ peak persists at an equal descent budget, so it is not explained by the number of descent steps (candidate evaluations still differ between depths).

One asymmetry in the above setup is that $\mathrm{cd}{=}0$ runs no descent and therefore does less total work than the $\mathrm{cd}{\geq}1$ settings. To check that $\mathrm{cd}{=}0$ is not simply limited by compute, we ran an extended $\mathrm{cd}{=}0$ budget sweep on JSSP (ta09–ta10, 2 seeds each) at 300, 600, 900, and 1500 steps (up to $5\times$ the $\mathrm{cd}{=}1$ step budget, i.e., $5\times$ the number of candidate evaluations). The $\mathrm{cd}{=}0$ incumbent never improves on the warm-up makespan mk_0 under any of these budgets for either selector (learned or random): the candidate compound moves are randomly generated, and without a subsequent descent almost none of them shortens the makespan, so neither selector finds one; only exact evaluation of every candidate does (the greedy baseline improves at $\mathrm{cd}{=}0$ in all 50 runs, Table 5). At the same 300-step budget, $\mathrm{cd}{=}1$ already reduces the incumbent makespan under learned selection from $\mathrm{mk}_0{=}1995$ to 1590.5 (optimality gap 26.5%). The $\mathrm{cd}{=}1$ benefit therefore cannot be explained by $\mathrm{cd}{=}0$ receiving too few candidate evaluations.

A.8 JSSP BUDGET SCALING (100–5000 STEPS)

To check whether the JSSP pattern of Table 1 (difference ≈ 0 at $\mathrm{cd}{=}0$, peak at $\mathrm{cd}{=}1$) is a short-budget artifact, we ran the JSSP sweep at step budgets $\{100, 500, 1000, 2000, 5000\}$ using an RF scorer trained in the same way and the held-out instances ta09–ta10 (2 instances $\times$ 2 seeds, 30 candidates per step, single fold). Table 13 reports the learned-minus-random makespan difference at each step count, and the learned scorer's final optimality gap.

Table 13: JSSP budget scaling on Taillard ta09–ta10 (RF scorer; n=4 runs per cell). Learned-minus-random makespan difference (positive = learned better) at each step count; the final column gives the learned scorer's optimality gap (%) at 5000 steps. The cd=0 incumbent stays at the warm-up solution (mk_0=1995); the cd=1 advantage grows 1.84× from 100 to 5000 steps; the cd=3 difference is small and negative only at 500 steps.

cd	100	500	1000	2000	5000	opt. gap (%)
0	+0.00	+0.00	+0.00	+0.00	+0.00	58.70
1	+301.75	+435.75	+476.00	+536.25	+555.00	13.12
2	+258.25	+267.75	+267.50	+280.25	+293.00	12.38
3	+20.75	−25.25	+24.50	+39.25	+67.00	12.27

We make four observations. (i) The cd=0 incumbent never changes from the warm-up solution at any budget: no individual trajectory (any instance or seed) improves on mk_0 within 5000 steps, because the selected compound move is applied unconditionally and, without descent, almost never improves the makespan; the ≈0 difference at cd=0 in Table 1 is therefore not budget-limited. (ii) The cd=1 advantage *grows* with budget (301.75 → 555.00, a 1.84× increase), so the cd=1 difference at the 100-step budget of Table 1 is conservative, not a short-budget artifact. (iii) The ordering of the differences (cd=1 largest, then cd=2, then a small cd=3 difference) holds at every step count up to 5000 steps; the cd=3 difference lies between −25.25 and +67.00 and is negative only at 500 steps. (iv) In absolute terms, learned selection ends with a lower optimality gap at 5000 steps than random selection at every cd≥1: 13.12%, 12.38%, and 12.27% at cd=1, 2, 3, against 57.27%, 35.73%, and 17.61% for random selection. The lowest learned optimality gap is at cd=3, where the learned-minus-random difference is smallest. Under the matched descent budget of Appendix A.7, learned selection at cd=1 (300 search steps) improves the incumbent by 338.1, against 280.8 for random selection at cd=3 (100 search steps). The learned-minus-random difference measures the value of the scorer at a given commit depth, not the best absolute configuration.

A.9 Robustness to the candidate-list size (C=10–100)

The main experiments hold the number of candidates scored per step fixed at C=30. Here we re-ran the JSSP commit-depth sweep at $C \in \{10, 30, 60, 100\}$, holding everything else fixed: the same five CV folds on ta01–10, the same RF training procedure (one RF per fold, reused for every C), five seeds per test instance, and the 100-step search budget. Candidates are drawn with common random numbers across C: for a given selector, whenever two runs are at the same state, the smaller candidate list is a prefix of the larger one; trajectories may diverge after different candidates are selected.

Table 14: Learned-minus-random difference by candidate-list size C (n=50 paired (instance, seed) measurements per cell). Significance tests use the ten per-instance means (seeds averaged within each instance).

C	cd=0	cd=1	cd=2	cd=3	peak
10	0.0	+181.7	+173.9	+15.8	cd=1
30	−0.1	+201.1	+159.1	−23.2	cd=1
60	+0.4	+223.1	+159.6	−6.3	cd=1
100	+0.2	+212.9	+144.1	+8.8	cd=1

The difference peaks at cd=1 for every tested C: the cd=1 difference is positive at every C (all $p<10^{-5}$ on the per-instance means), the cd=2 difference is also positive at every C (all $p<10^{-4}$), and the cd=3 difference is small and not significant at any C (between −23.2 and +15.8, all $p>0.05$). At cd=0 the difference remains approximately zero across the range for a structural reason: in 47–50 of the 50 runs per setting, neither selector ever improves on the starting solution (after warm-up), so ranking ten times as many candidates does not let the cd=0 loop improve. Both selectors scored exactly C feasible candidates at every step, so the comparison is not affected by different rates of infeasible candidates. This check uses a different random-number stream, so the C=30 row is not an exact numerical replication of Table 1, but its qualitative pattern (difference ≈0 at cd=0, the cd=1 peak, and a large positive cd=2 difference) is the same.

A.10 CLUSTER-ROBUST AND INSTANCE-AVERAGED STATISTICS

The main-text statistics treat each (instance, seed) pair as an independent unit. Because each instance is run with five seeds, residuals are not fully independent. We therefore re-analyze the main JSSP result (n=10 instances, 5 seeds per instance) under two cluster-aware procedures:

(a) Instance-averaged (n=10). For each instance we average the difference over its 5 seeds, then run a one-sample t-test across the 10 instance means.

(b) Wild cluster bootstrap. Rademacher weights at the instance level, B=10,000 replications, paired t-statistic with instance as cluster.

Table 15: JSSP cluster-robust statistics. The naive 95% CI uses n=50 and assumes independent units; the instance-averaged test uses n=10; the wild cluster bootstrap (WCB) p-value is two-sided.

cd	Mean diff.	Naive 95% CI	**Inst-avg p**	**Inst-avg d**	**WCB p**	**Cluster-robust 95% CI**
0	+0.4	[−0.1, +0.9]	0.17	0.47	0.18	[−0.2, +1.0]
1	+220.2	[+195.3, +245.0]	3.1×10^{-8}	5.50	$< 10^{-4}$	[+191.6, +248.8]
2	+198.0	[+170.1, +225.9]	6.5×10^{-7}	3.87	$< 10^{-4}$	[+161.4, +234.6]
3	+39.6	[+15.0, +64.1]	0.019	0.90	0.023	[+8.2, +70.9]

Signs and significance are unchanged under both procedures: the difference is not significant at cd=0 and positive at cd=1, 2, and 3. The cluster-robust CIs are wider than the naive n=50 CIs (1.15×, 1.31×, and 1.28× at cd=1, 2, 3), but all three exclude zero.

A.11 COMMIT-DEPTH SWEEP ON LARGER JSSP INSTANCES

Table 16 reports the learned-minus-random difference at each commit depth for four Taillard instance sizes. At every size the difference is ≈0 at cd=0 and large and positive at cd=1 and cd=2 (positive in 40 of 40 runs at cd=1 and in 39 of 40 at cd=2). The peak is at cd=1 for 15×15 and 20×20 and at cd=2 for 20×15 and 30×15. The ta11–20 and ta31–40 runs use the RF trained on ta01–10 (15×15) states; ta21–30, whose machine count differs from that of the other three sets, uses an RF trained on its own training instances (ta21–28). For ta11–20, the mean within-state ρ at K=1 matches that of ta01–10 to within 0.004 (0.458 vs. 0.462), so the within-state signal is unchanged at the larger size.

Table 16: JSSP cd sweep across instance sizes (Taillard instances; n=2 test instances × 5 seeds per size; the ta01–10 row differs from the corresponding row of Table 1, which uses 5-fold CV with n=50).

Size	cd=0	cd=1	cd=2	Peak
15×15 (ta01–10)	+0.4	**+270.1**	+189.7	1
20×15 (ta11–20)	+0.0	+165.7	**+307.8**	2
20×20 (ta21–30)	+0.0	**+338.6**	+272.2	1
30×15 (ta31–40)	+0.0	+132.1	**+373.8**	2

A.12 NON-TAILLARD JSSP BENCHMARKS

To check for a Taillard-specific artifact, we evaluate the cd=0 versus cd=1 contrast on three benchmark sets by other authors (from JSPLIB): SWV (Storer–Wu–Vaccari), ABZ (Adams–Balas–Zawack), and YN (Yamada–Nakano), across five (collection, size) configurations. For each configuration, an RF is trained on two to seven instances of the collection and evaluated on one to three held-out instances (depending on collection size), with three seeds each, 60 steps, and 20 candidates per step. This is a binary contrast at cd $\in \{0, 1\}$, not a full sweep.

All five configurations reproduce the pattern of Table 1: a difference of ≈ 0 at cd=0 and a large positive difference at cd=1, growing with instance size within SWV (from +129.3 at 20×10 to +362.3 at 50×10). The per-configuration sample is small (three to nine runs), so we treat this as a replication check across independent benchmark sets rather than as additional primary evidence.

Table 17: Learned-minus-random difference on non-Taillard JSSP benchmarks; mean over held-out (instance, seed) runs.

Collection	Size	cd=0	cd=1
SWV	20×10	−2.3	+129.3
SWV	20×15	+0.0	+157.5
SWV	50×10	+0.0	+362.3
ABZ	20×15	+0.3	+65.0
YN	20×20	+0.0	+109.3

B PUBLISHED METHODS AND RULE-BASED SELECTORS

The commit-depth study of the six published methods in Section 4 is described in Appendix F, the MIStar analysis in Appendix G, and the fine-tuning runs of Section 5 in Appendix H. This appendix reports three related experiments: a large neighborhood search with learned repair operators (NLNS), rule-based selectors in a tabu search for JSSP, and local-search post-processing for MIS.

B.1 NLNS ON CVRP

For NLNS (Hottung & Tierney, 2020), we run the authors' released code and pretrained repair models (four destroy–repair pairs: tour-based and point-based destroy, each with degree of destruction $p \in \{0.2, 0.3\}$) on 200 CVRP20 instances from the authors' test set. We run the authors' batch search loop for a fixed 50 iterations rather than their time limit; the only other modification is cd passes of intra-route 2-opt after each repair (a pass applies every improving reversal found in one sweep of the route); cd=0 recovers the authors' default loop. Table 18 reports mean cost at each commit depth.

Table 18: NLNS (Hottung & Tierney, 2020) on CVRP (200 CVRP20 instances, 50 LNS iterations). cd: number of intra-route 2-opt passes after each repair. Last column: cost with random removal minus cost with NLNS destroy. Lower is better.

cd	Mean cost	Δ vs. cd=0	Δ%	random − NLNS
0	6.3475	—	—	+0.111
1	6.2979	−0.0496	−0.78%	−0.017
2	6.2825	−0.0650	−1.02%	—
3	**6.2531**	**−0.0944**	**−1.49%**	+0.030
5	6.2619	−0.0856	−1.35%	—

Cost falls up to cd=3 (paired $p<10^{-7}$ against cd=0, n=200) and does not change significantly from cd=3 to cd=5 (p=0.50). The last column compares the NLNS destroy operators with random removal (degree of destruction 0.3), both followed by the learned repair (positive = NLNS destroy is better): +0.111 at cd=0, −0.017 at cd=1, and +0.030 at cd=3, consistent with deeper post-repair descent equalizing the two destroy procedures.

B.2 RULE-BASED SELECTORS ACROSS COMMIT DEPTHS

We also ran rule-based selectors in a different JSSP loop: a tabu search over critical-block swap candidates (as in $\mathcal{N}_7$ (Zhang et al., 2007)), in which the selector shortlists its 10 top-scored candidates and the best non-tabu one by exact makespan is applied (tabu tenure 7–15, aspiration by the best makespan, restart from the best solution after 300 non-improving iterations, 2000 iterations), followed by cd descent steps. The selectors are BOUNDARY (prefers swaps at a critical-block boundary, then larger blocks), BLOCK-SIZE (prefers larger blocks), POS-DISTANCE (prefers a larger positional distance), random shortlisting, and a separately trained MLP move ranker. We use Taillard ta01–05 with 3 seeds per instance (n=15 runs) and cd $\in \{0, 1, 2\}$.

We make two observations. First, all three rule-based selectors are 8–12 pp *worse* than random shortlisting at cd=0; they recover most of that deficit at cd=1, and two of the three reach parity or better at cd=2, so in this loop, too, descent after each move changes whether a selector helps. Second,

Table 19: Mean optimality gap (%, *lower is better*) in the tabu-search loop on JSSP ta01–05. The last three columns give random − selector (positive = selector better than random shortlisting; negative = worse).

Selector	cd=0	cd=1	cd=2	random − selector		
				cd=0	cd=1	cd=2
random	9.40	7.30	9.17	0.00	0.00	0.00
boundary	17.30	7.59	7.88	−7.90	−0.28	+1.29
block-size	21.32	11.40	10.77	−11.92	−4.10	−1.60
pos-distance	20.42	10.38	6.82	−11.02	−3.08	+2.35
learned (MLP)	11.31	13.85	14.70	−1.91	−6.55	−5.53

relative to random the rule-based selectors improve monotonically up to $\mathrm{cd}{=}2$, the deepest setting tested, while the MLP ranker is worse than random at every depth. The loop, the candidates, and the depth range differ from those of Section 3, so these numbers are not directly comparable with Table 1.

B.3 Local-search post-processing on MIS

To check whether search effort after the scorer's choice masks the scorer's advantage beyond our setting, we test a simple analogue of the observation of Boether et al. (2022) that a learned MIS scorer can be replaced by one with random weights under deep search. On 100 random Erdős–Rényi graphs (n=200), we build greedy solutions from the classical min-degree rule and from random scores, then apply iterated local search with (1,2)-swaps for a varying number of iterations.

Table 20: MIS min-degree vs. random scores: the difference in independent-set size (Diff.) decays with the number of ILS iterations on ER graphs (n=200). %: difference relative to the min-degree value.

	Sparse (p=0.05)				Dense (p=0.15)			
ILS iter.	Min-deg.	Random	Diff.	%	Min-deg.	Random	Diff.	%
0	53.5	47.9	5.6	10.5	25.1	22.0	3.1	12.3
5	58.0	55.3	2.7	4.7	28.1	26.4	1.6	5.8
50	58.4	56.4	2.0	3.5	28.9	28.3	0.6	2.0
100	58.4	56.7	1.7	2.9	29.2	28.9	0.3	1.0

The difference between the min-degree rule and random decays rapidly with the number of ILS iterations, consistent with Boether et al. (2022): longer local search masks differences between starting solutions. This is not a commit-depth experiment, since the scorer builds the starting solution rather than selecting moves.

C Commit-depth sweep in six domains

We sweep $\mathrm{cd} \in \{0, 1, 2, 3\}$ in all six domains with trained scorers (JSSP, TSP, MaxSAT, Knapsack, GC, BP) under one protocol: 100 search steps with 30 candidates per step and 5-fold cross-validation with two test instances per fold (n=50 paired (instance, seed) measurements per commit depth per domain). Each fold trains the scorer on 8 instances and tests on the remaining 2; every instance appears in exactly one test fold, and each (instance, seed) pair is the unit of analysis (cluster-robust checks for JSSP: Appendix A.10). Only the commit depth varies; the scorer, its training target and architecture, the candidate generator, the move operator, and the number of search steps are held fixed. The *oracle* scorer of Appendices D.2 and E.3 selects, from each state's candidate list, the move with the highest true δ_1; it is an upper bound for one-step ranking within that list, not an oracle over the full neighborhood.

Table 21 reports the learned-minus-random difference. In JSSP (main setting of Section 2: RF trained without null moves, null moves removed from the candidate lists), the difference is $+0.4$ at $\mathrm{cd}{=}0$ (not significant), $+220.2$ at $\mathrm{cd}{=}1$, $+198.0$ at $\mathrm{cd}{=}2$, and $+39.6$ at $\mathrm{cd}{=}3$. In TSP the difference peaks

at cd=0 (+2.63, d=5.12, $p<10^{-36}$) and decays toward zero at larger cd. The MaxSAT difference also peaks at cd=0 (+6.82, d=1.89) and is slightly negative at cd≥1. Knapsack, GC, and BP also peak at cd=0. For GC and BP the table shows only cd=0 and cd=1: at cd=2, 3 the GC difference is exactly 0 and the BP difference is small and not significant (+0.12, +0.09). JSSP is the only one of the six domains whose difference peaks at cd≥1; Appendix E relates this to the saturation ratio.

Table 21: **Commit-depth sweep, 5-fold CV (n=50).** Learned-minus-random difference in incumbent improvement after 100 steps (objective units); d = Cohen's d. $^{***}p<0.001$; $^{**}p<0.01$; $^{*}p<0.05$; ns = not significant. Only commit depth varies; scorer, architecture, and number of steps are fixed. The last two rows (GC, BP) show cd=0 / cd=1 (other depths in the text). JSSP: RF trained without null moves, null moves removed from the candidate lists (with them: Appendix A.4).

Domain	cd	diff.	d	sig
JSSP	0	+0.4	0.24	ns
	1	**+220.2**	2.52	***
	2	+198.0	2.01	***
	3	+39.6	0.46	**
TSP	0	**+2.63**	5.12	***
	1	+0.77	2.15	***
	2	+0.10	0.63	***
	3	+0.10	0.70	***
MaxSAT	0	**+6.82**	1.89	***
	1	−0.78	−0.37	*
	2	−0.76	−0.42	**
	3	−0.76	−0.36	*
Knapsack	0	**+493.9**	3.66	***
	1	+4.7	0.18	ns
	2	−6.2	−0.23	ns
	3	−8.6	−0.33	*
GC	0/1	**+30.18** / 0.0	5.81 / 0	*** / —
BP	0/1	**+24.92** / +0.30	2.75 / 0.43	*** / **

C.1 Replication with 30 instances per domain

We replicate the sweep on 30 instances per domain (5-fold CV with six test instances per fold, n=150 paired measurements per cell) for the five non-JSSP domains with trained scorers (JSSP is excluded because its instance set is fixed by the Taillard ta01–10 benchmark). Table 22 reports the learned-minus-random difference.

Table 22: Replication with 30 instances per domain: learned-minus-random difference (mean over n=150 paired measurements) vs. the original n=50 estimates from Table 21.

Domain	cd=0 (n=150)	cd=0 (n=50)	cd=1 (n=150)	cd=1 (n=50)
TSP	+2.52	+2.63	+0.80	+0.77
MaxSAT	+8.35	+6.82	−0.98	−0.78
Knapsack	+481.42	+493.92	+7.73	+4.70
GC	+29.77	+30.18	+0.00	+0.00
BP	+22.05	+24.92	−0.30	+0.30

The cd=0 regime holds in all five domains, and the magnitudes are close to the n=50 estimates (BP's small cd=1 difference changes sign).

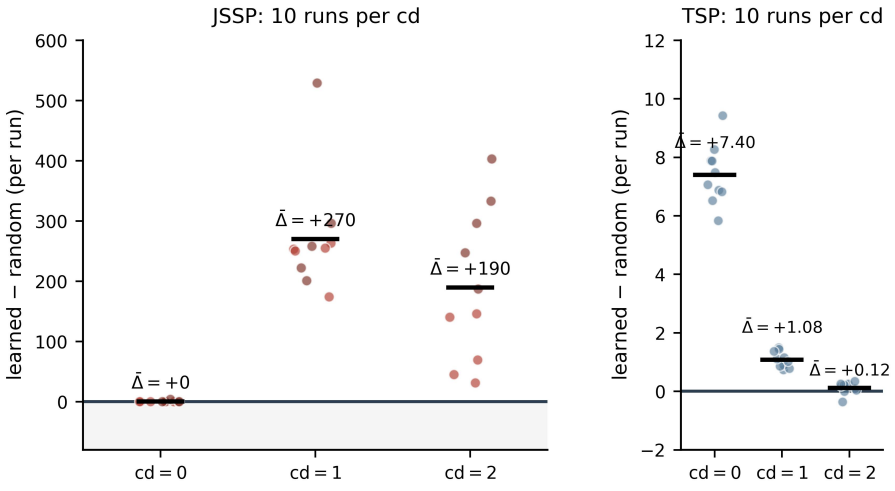

Figure 2: Per-run learned-minus-random difference across the 10 runs (2 test instances $\times 5$ seeds) at each commit depth, for JSSP and TSP. Each dot is one run; the black tick marks the mean. **JSSP** (RF trained without null moves, null moves removed from the candidate lists): $1/10$ runs positive at cd=0, $10/10$ at cd=1, $10/10$ at cd=2 (mean differences $+0.4$, $+270.1$, $+189.7$). **TSP**: $10/10$, $10/10$, $8/10$ across cd $\in \{0, 1, 2\}$. MaxSAT (not shown): $10/10$, $4/10$, $4/10$. The share of positive runs is highest at the peak commit depth in all three domains (tied with cd=2 for JSSP and with cd=1 for TSP).

C.2 Per-run consistency

C.3 Cross-architecture check on MIS

To test whether the predicted cd=0 regime on MIS is specific to the RF scorer, we ran a paired sweep with an RF and a residual message-passing GNN (3 layers, hidden dimension 64) trained on the same one-step target from the same 4,800 samples. Instances are Erdős–Rényi graphs with 50 nodes and edge probability 0.2; we use 8 training and 4 held-out test instances, 3 seeds per instance, and 60 search steps with 15 candidates per step.

Table 23: MIS cross-architecture sweep: mean improvement and learned-minus-random difference per commit depth (4 test instances $\times$ 3 seeds).

cd	random	RF	GNN	RF−random	GNN−random
0	+13.92	+127.17	+126.50	+113.25	+112.58
1	+131.33	+130.67	+130.83	−0.67	−0.50
2	+133.08	+132.75	+132.58	−0.33	−0.50

The two architectures produce learned-minus-random differences within one unit of each other at every depth, and both exhibit the predicted cd=0 regime: the difference collapses to ≈ 0 once a descent step follows the applied move. This does not establish architecture independence in general, but in the tested setting the regime depends on the problem and move operator, not on the model class.

D Within-state ranking

This appendix asks what the JSSP scorer predicts. It uses 150,000 (state, move) samples on ta01–10 (500 states per instance: random solutions, local optima, and perturbed local optima; 30 random compound moves each, none of them a null move) and an $80/20$ instance-level split that holds out 2 of the ten instances (random seed 0). Besides the RF, we train an MLP (hidden sizes 256 and 128) and a RankNet pairwise ranker (logistic pairwise loss, only pairs with $|\delta_1(i) - \delta_1(j)| \geq 1$).

Global metrics, pooled over states, overstate how well the JSSP scorer ranks moves. Training an RF on δ_K for $K \in \{1, 5, 20, 50\}$ raises the global Spearman correlation from 0.630 at $K{=}1$ to 0.871 at $K{=}20$, but a state-only model (move features dropped) also reaches 0.871 at $K{=}20$. The rise comes from predicting the value of *states*, not from ranking *moves*. A variance decomposition is consistent with this: the within-state share of $\mathrm{Var}(\delta_K)$ drops from 65.4% at $K{=}1$ to 11.6% at $K{=}5$ (Table 24), leaving little variance for ranking moves within a state.

The loop uses only the within-state residual. Decompose $\delta_K(s, e) = \mu_K(s) + \eta_K(s, e)$: the loop ranks candidates at a fixed state s, so only η_K drives the move choice. At $K{=}1$, the mean within-state Spearman correlation with η_1 is $+0.462$ (Table 24; $z{=}79.8$ against a permutation null, Appendix D.1). This is well above chance but far below the $+0.860$ of the TSP scorer (Appendix D.2).

Table 24: Within-state share of $\mathrm{Var}(\delta_K)$ and global vs. within-state Spearman (RF, two held-out instances; no null moves in the training or test data). The global column is the commonly reported pooled figure; the within-state column measures what the loop uses (Spearman ρ over each state's candidate moves, averaged over test states); $\%\,\rho{>}0$ is the share of states with $\rho > 0$; residual ρ is the pooled Spearman after subtracting state means from predictions and targets.

Target	within-state var.	global ρ	within-state ρ	$\%\,\rho{>}0$	residual ρ
δ_1	65.4%	0.630	$+0.462$	98.6%	$+0.475$
δ_5	11.6%	0.801	$+0.164$	77.1%	$+0.135$
δ_{20}	2.5%	0.871	$+0.010$	51.0%	$+0.010$
δ_{50}	3.6%	0.870	$+0.004$	50.9%	$+0.007$

An MLP reaches the same within-state ρ as the RF (0.46). The RF under CV (Section 3) and the MLP and RankNet scorers (in-sample, all ten instances) all yield a learned-minus-random difference of at most 0.5 makespan units at $\mathrm{cd}{=}0$ (RF $+0.4$, MLP $+0.48$, RankNet $+0.30$), so the lack of gain at $\mathrm{cd}{=}0$ is not specific to the model class. By contrast, TSP (within-state $\rho{=}0.86$) already gains at $\mathrm{cd}{=}0$, with 64% of the oracle's gain over random selection, against $\sim 0\%$ in JSSP (Appendix D.2).

D.1 Within-state ranking analyses and cross-validation stability

Pairwise accuracy vs. tie threshold ϵ. We report the per-state pairwise accuracy of the RF scorer on the two held-out ta01–10 instances as a function of the threshold ϵ on the makespan difference used to filter near-tied pairs. A permutation null (100 shuffles within each state, with the resulting means averaged) is included.

K	$\epsilon{=}0$	$\epsilon{=}1$	$\epsilon{=}2$	$\epsilon{=}5$	$\epsilon{=}10$	z (ρ vs. null)
δ_1	0.5896	0.6843	0.6853	0.6881	0.6929	$+79.8$
δ_5	0.4104	0.5711	0.5721	0.5750	0.5813	$+23.1$
δ_{20}	0.3579	0.4568	0.4568	0.4577	0.4579	$+1.6$

The $K{=}1$ row increases monotonically with ϵ (the jump comes from dropping exact ties). The last column is the z-score of the within-state Spearman correlation (not of the pairwise accuracy) against its permutation null: the observed mean of $+0.462$ has $z{=}79.8$; these are the values quoted in Appendix D.

Stability across five random splits. All metrics are computed at $K{=}1$. Each split holds out a randomly chosen pair of instances (pairs may overlap across splits) and trains on the other 8. Top-1 hit rate is the share of test states whose top-ranked move is the best candidate (chance $1/30$); normalized top-1 regret is the target shortfall of the top-ranked move divided by the range of the target within the state.

Metric	mean	std	min	max
global Spearman	0.6302	0.0078	0.6185	0.6392
within-state mean ρ	0.4631	0.0055	0.4557	0.4717
top-1 hit rate	0.1391	0.0118	0.1234	0.1584
normalized top-1 regret	0.4702	0.0074	0.4596	0.4825
between-state variance share	0.3463	0.0098	0.3346	0.3603

D.2 JSSP vs. TSP at cd=0

Table 25 compares JSSP and TSP under the default (cd=0) loop. A moderate global Spearman ρ (0.63 for JSSP) overstates JSSP's ranking quality; within-state metrics show a much weaker signal than in TSP, and the JSSP scorer gains almost nothing over random selection at cd=0.

Table 25: JSSP vs. TSP under the default (cd=0) loop. Global ρ overstates JSSP's ranking quality; within-state ρ and top-1 (chance $1/30$) show a much weaker signal than in TSP; the JSSP scorer gains almost nothing at cd=0. top-1: share of states whose top-ranked move is the best; Δ: gain over random selection. JSSP: one split (ta09–ta10 $\times$ 5 seeds; the RF was trained on the data of Appendix D, which include these instances); TSP: all 10 instances $\times$ 5 seeds (8 of them used to train the RF), 10 warm-up steps.

	global ρ	within-state ρ	top-1	learned Δ	oracle Δ	learned/oracle
JSSP	0.630	+0.462	14.4%	+0.4	+287.5	∼0%
TSP	0.953	+0.860	48.8%	+4.94	+7.69	64%

E The saturation ratio

This appendix reports a side analysis that the main text does not rely on: a statistic of a small move sample that separates the two regimes at the default operators of our problem families, and the limits of that separation. Let $\delta_0 = \mathrm{cost}(s) - \mathrm{cost}(s \oplus e)$ be the immediate cost change and δ_1 the change after one additional descent step, so $G_1 = \mathbb{E}[\delta_1 - \delta_0]$ is the mean first-step gain. Across our problem families, the depth at which the learned-minus-random difference peaks is associated with the ratio of the first-step gain to the spread of δ_1 within a state:

$$\text{regime} = \begin{cases} \text{cd=0} & \text{if } \dfrac{G_1}{\sigma_{\text{within}}(\delta_1)} > 1, \\ \text{cd}\geq 1 & \text{otherwise.} \end{cases}$$

Here $\sigma_{\text{within}}(\delta_1)$ is the standard deviation of δ_1 across the sampled moves of a state, averaged over states, and the regime is the commit depth with the largest learned-minus-random difference (the scorer's added value), not the depth with the lowest final cost. We call the ratio the *saturation ratio*. Above 1, one descent step adds a gain common to all candidates that exceeds the spread left between them, so descent gives most of its benefit whichever candidate is chosen, and the scorer adds most without descent (cd=0). Below 1, candidates still differ after one step, so ranking them before descent pays off (cd$\geq$1). Both quantities are estimated from a small set of (state, move) pairs drawn at random solutions after a few descent steps. The threshold is motivated by a break-even argument (Appendix E.4), not derived from a model of search dynamics. This analysis assumes that the loop descends with the operator used to measure the ratio and that every candidate changes the solution (for JSSP the ratio is measured without null moves; Appendix A.4). It predicts only cd=0 versus cd$\geq$1, not the best positive depth, and it is expected to fail when the move size or the descent operator changes (Appendices E.2, E.7) and near a ratio of 1, where sampling noise dominates (Appendix E.6).

Twelve families. Figure 3 and Table 26 report the ratio and the observed regime for twelve problem families. The threshold of 1 was chosen on the first three (JSSP, TSP, MaxSAT); the remaining nine (Knapsack; permutation flow shop, PFS; GC; BP; quadratic assignment, QAP; maximum independent set, MIS; resource-constrained project scheduling, RCPSP; capacitated vehicle routing, CVRP; block relocation, BRP, in a layout variant) each have their own instance generator, neighborhood, and

feature encoding. QAP and MIS were added after the threshold was fixed: we computed the ratio, recorded the regime it implies, and only then ran the commit-depth sweep. In the six domains with trained scorers (type L in Table 26), the ratio is below 1 only for JSSP, which is also the only one whose learned-minus-random difference peaks at cd≥1. The six other families (type O: oracle scorer only; type P: a single configuration) are also consistent with the threshold, on weaker evidence. Ten of the twelve families are in the cd=0 regime, so this agreement is a weak test of the threshold.

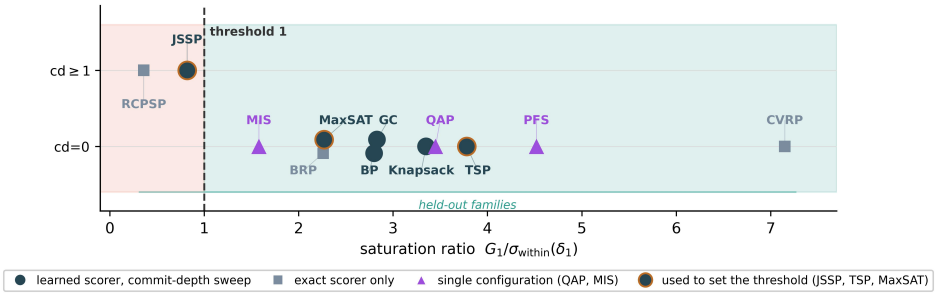

Figure 3: **Saturation ratio and observed regime for all twelve families.** x-axis: saturation ratio $G_1/\sigma_{\text{within}}(\delta_1)$; y-axis: observed regime, i.e., the cd with the largest learned-minus-random difference (cd=0 vs. cd≥1). The threshold of 1 (where G_1 equals the within-state spread) was chosen on JSSP, TSP, and MaxSAT before the other nine families were evaluated; it agrees with the observed regime in $9/9$ of them, but eight of the nine are in the cd=0 regime. Marker shape denotes the type of evidence: L = learned-scorer sweep, O = oracle scorer only, P = single configuration (QAP and MIS tested prospectively). JSSP is plotted at 0.82, its ratio without null moves. Threshold sensitivity: Appendix E.5.

Table 26: Saturation ratio $G_1/\sigma_{\text{within}}(\delta_1)$ and the regime implied by the threshold for the twelve families, grouped by type of evidence. The threshold of 1 separates the regimes. Evidence: **L**=learned-scorer sweep over several configurations (p-values in Appendix E.3); **O**=oracle scorer only; **P**=single configuration (QAP and MIS tested prospectively). JSSP: move sample without null moves (0.72 with them).

Family	Evidence	Saturation ratio	Pred	Obs
JSSP	L	0.82	≥1	≥1
TSP	L	3.78	0	0
MaxSAT	L	2.27	0	0
Knapsack	L	3.35	0	0
GC	L	2.83	0	0
BP	L	2.80	0	0
RCPSP*	O	0.36	≥1	≥1
CVRP*	O	7.15	0	0
BRP*	O	2.26	0	0
PFS	P	4.52	0	0
QAP	P	3.45	0	0
MIS	P	1.58	0	0

*Oracle scorer only; the oracle peak matches the prediction at the measured configuration (RCPSP: regime depends on instance size; Appendix E.3).

Families not used to choose the threshold. The threshold of 1 was chosen on JSSP, TSP, and MaxSAT before the other nine families were evaluated (any threshold in $(0.82, 2.27)$ separates these three) and was not fitted per family. It agrees with the observed regime in all 9 other families, but eight of the nine are in the cd=0 regime, so always predicting cd=0 would agree in 8 of 9; the only informative case is RCPSP, for which we have an oracle scorer only and whose regime depends on instance size (Appendix E.3). Along move-size sweeps the threshold agrees with the observed regime less often than always predicting the majority regime (below). Any threshold in $(0.82, 1.58)$ separates all twelve families; this interval was found post hoc (Appendix E.5). Refitting the threshold

with each family left out puts every left-out family on the side of its observed regime; MIS (ratio 1.58) has the smallest margin (Appendix A.6).

Estimation from a small sample. The ratio can be estimated from a small random sample without training a scorer or running a sweep: 50 random (state, move) pairs (10 states $\times$ 5 moves per state) yield G_1 and $\sigma_{\text{within}}(\delta_1)$. Across resamples, a 50-pair estimate falls on the correct side of 1 in every draw for the seven other tested families, whose ratios are well away from 1. JSSP is close to 1: a 50-pair estimate is on the correct side in 30.6% of draws from a synthetic 10×10 pool (pool ratio 0.94) and in 52.5% of draws from the Taillard sample of Table 26 (ratio 0.82); with 200 pairs the shares are 58.3% and 91.8% (Appendix E.6).

Varying the move size. The ratio also tracks regime changes as the move size varies within a family. In MaxSAT, increasing the number of simultaneous variable flips (n_{flip} from 1 to 15) pushes the saturation ratio monotonically below the threshold and shifts the observed peak from cd=0 to cd=1 (between n_{flip}=5 and 8; Figure 4, Appendix E.1). Graph Coloring shows the same trend: increasing k_{recolor} (vertices recolored per move) from 1 to 20 pushes the ratio below the threshold; the cd=0 advantage over cd=1, significant at small k_{recolor}, vanishes near the threshold (Appendix E.1). Across 28 families the ratio falls with the move size in 27, but the ratio at which the regime changes depends on the family: over all 368 (family, move size) points the threshold of 1 agrees with the observed regime at 46.5% of the points, less often than always predicting the majority regime (55.7%) (Appendix E.2).

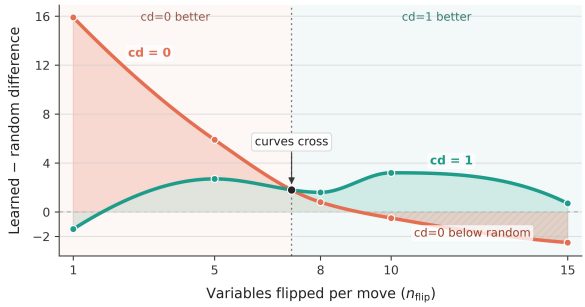

Figure 4: **Regime change in MaxSAT as moves flip more variables.** As the number of simultaneous variable flips grows, the cd=0 difference (red) falls below zero while the cd=1 difference (green) rises; the curves, interpolated between the swept values $n_{\text{flip}} \in \{1, 5, 8, 10, 15\}$, cross near $n_{\text{flip}} \approx 7$.

E.1 Regime changes within a family

Table 27 reports G_1, $\sigma_{\text{within}}(\delta_1)$, and their ratio for MaxSAT at each n_{flip}. As n_{flip} grows, $\sigma_{\text{within}}(\delta_1)$ rises (candidates differ more after descent) while G_1 rises only modestly, so the saturation ratio $G_1/\sigma_{\text{within}}(\delta_1)$ drops monotonically and crosses the threshold of 1 between n_{flip}=7 and 8, consistent with the cd=0→cd=1 transition observed between the swept values n_{flip}=5 and 8 (Figure 4).

Table 27: G_1, $\sigma_{\text{within}}(\delta_1)$, and the saturation ratio for MaxSAT at varying n_{flip}. The ratio decreases monotonically as moves flip more variables (σ_{w} abbreviates σ_{within}). Obs. peak from cd sweeps at $n_{\text{flip}} \in \{1, 5, 8, 10, 15\}$ (10 runs per cell); —: not swept.

n_{flip}	G_1	$\sigma_{\text{w}}(\delta_1)$	$G_1/\sigma_{\text{w}}(\delta_1)$	Obs. peak
1	3.34	1.49	2.24	cd=0
3	3.59	2.48	1.45	—
5	3.88	3.21	1.21	cd=0
7	3.90	3.76	1.04	—
8	3.92	4.27	0.92	cd=1
10	4.00	4.45	0.90	cd=1
15	4.17	5.07	0.82	cd=1

The direction seen across families in Appendix E, where the ratio was measured at one move size per family, also holds along the MaxSAT sweep: as n_{flip} grows, the saturation ratio falls below the threshold and the observed peak shifts from $\text{cd}{=}0$ to $\text{cd}{=}1$. The direction of the effect (a lower saturation ratio favors $\text{cd}{\geq}1$) is the same across families and within a family; Appendix E.2 extends the sweep to 28 families and shows that the trend holds throughout while the ratio at the transition depends on the family.

Graph Coloring. Table 28 reports the same quantities for Graph Coloring under multi-vertex recoloring (k vertices recolored per move). The saturation ratio decreases monotonically as more vertices are recolored. Consistent with the threshold, the $\text{cd}{=}0$ advantage is statistically significant while the saturation ratio is well above the threshold ($k{=}1$: $p < 10^{-100}$; $k{=}5$: $p{=}0.0024$ against $\text{cd}{=}1$), and no cd setting is significantly better once the ratio is near or just below 1 ($k{\geq}10$, all pairwise $p{>}0.05$).

Table 28: G_1, $\sigma_{\text{within}}(\delta_1)$, and the saturation ratio for Graph Coloring at varying k_{recolor}.

k_{recolor}	G_1	$\sigma_{\text{w}}(\delta_1)$	$G_1/\sigma_{\text{w}}(\delta_1)$	Obs. peak
1	4.91	1.74	2.81	$\text{cd}{=}0$
5	5.28	3.82	1.38	$\text{cd}{=}0$
10	5.43	4.99	1.09	$\text{cd}{=}2^{\dagger}$
15	5.60	5.73	0.98	$\text{cd}{=}2^{\dagger}$
20	5.70	6.03	0.95	$\text{cd}{=}2^{\dagger}$

$n{=}240$ runs per cell. For $k{=}1, 5$ the $\text{cd}{=}0$ peak is significant ($p < 0.003$). $^{\dagger}k{\geq}10$: the saturation ratio approaches the threshold of 1 and pairwise differences between cd settings are not significant ($p > 0.05$).

E.2 Regime changes with move size across 28 families

The MaxSAT and Graph Coloring sweeps above show a family moving from $\text{cd}{=}0$ to $\text{cd}{\geq}1$ as its moves get larger. To test how general this is, and how well a fixed threshold locates the transition, we swept the compound-move size k (the number of elementary moves composed into one candidate) for 28 families under the loop of Section 2 with 20 random candidates per step (30 for JSSP), one candidate picked by an RF scorer or an exact-evaluation oracle (ranking by cost immediately, after one descent step, or after cd descent steps), and cd best-improvement descent steps over the elementary-move neighborhood after each applied move. Each (family, k) point is a sweep over $\text{cd} \in \{0, \ldots, 5\}$ and descent to a local optimum, paired with random selection (100 steps, 8–12 instances, 4–8 seeds); the observed peak is the cd with the largest difference from random selection, and where several scorers were run at the same k we report the median of the ratio estimates and the largest peak depth. The set comprises eleven of the twelve families of Table 26 (all but BRP, which was swept at $k{=}1$ only; RCPSP and CVRP use Variant II of Appendix E.7) and seventeen further families, 368 (family, k) points in total. Ratios are recomputed with a shared estimation protocol and shared move definitions and therefore differ, sometimes substantially, from Table 26 (e.g., QAP 17.1 vs. 3.45 at $k{=}1$). The JSSP points use the main setting (no null moves in the move samples, the candidate lists, or the RF's training data) and the RF scorer only.

Findings. Figure 5 and Table 29 summarize the sweep. *(i) Larger moves favor larger commit depths.* As k grows, the saturation ratio falls in $27/28$ families (negative Spearman correlation with k; QAP is flat after $k{=}2$), and in every family whose peak ever moves off $\text{cd}{=}0$ the observed peak depth is positively rank-correlated with k ($22/22$). 21 families peak at $\text{cd}{=}0$ for small k and at $\text{cd}{\geq}1$ for larger k; 17 of them stay at $\text{cd}{\geq}1$ at every larger size swept, DS and SC return to 0 at one intermediate size, MaxSAT at 6 of 20 larger sizes, and GBP peaks at $\text{cd}{\geq}1$ at a single size only. Six families (MIS, MKP, NPP, SS, VC, and RCPSP) remain at $\text{cd}{=}0$ throughout the swept range, even where the ratio falls to 0.58; JSSP is at $\text{cd}{\geq}1$ from $k{=}1$; for small moves its difference becomes negative with deep descent ($\text{cd}{\geq}3$ at $k{=}1$, $\text{cd}{\geq}4$ at $k{=}2$; $p{\leq}0.001$), as with descent to a local optimum in Appendix A.3. *(ii) The transition point depends on the family.* The ratio interval bracketing the transition (Figure 6) overlaps the interval $(0.82, 1.58)$ of Appendix E in $9/21$ families and spans 0.56 (QKP) to 7.1 (QAP) overall. The fixed threshold of 1 places the transition too late in $17/21$ families (the ratio at the onset

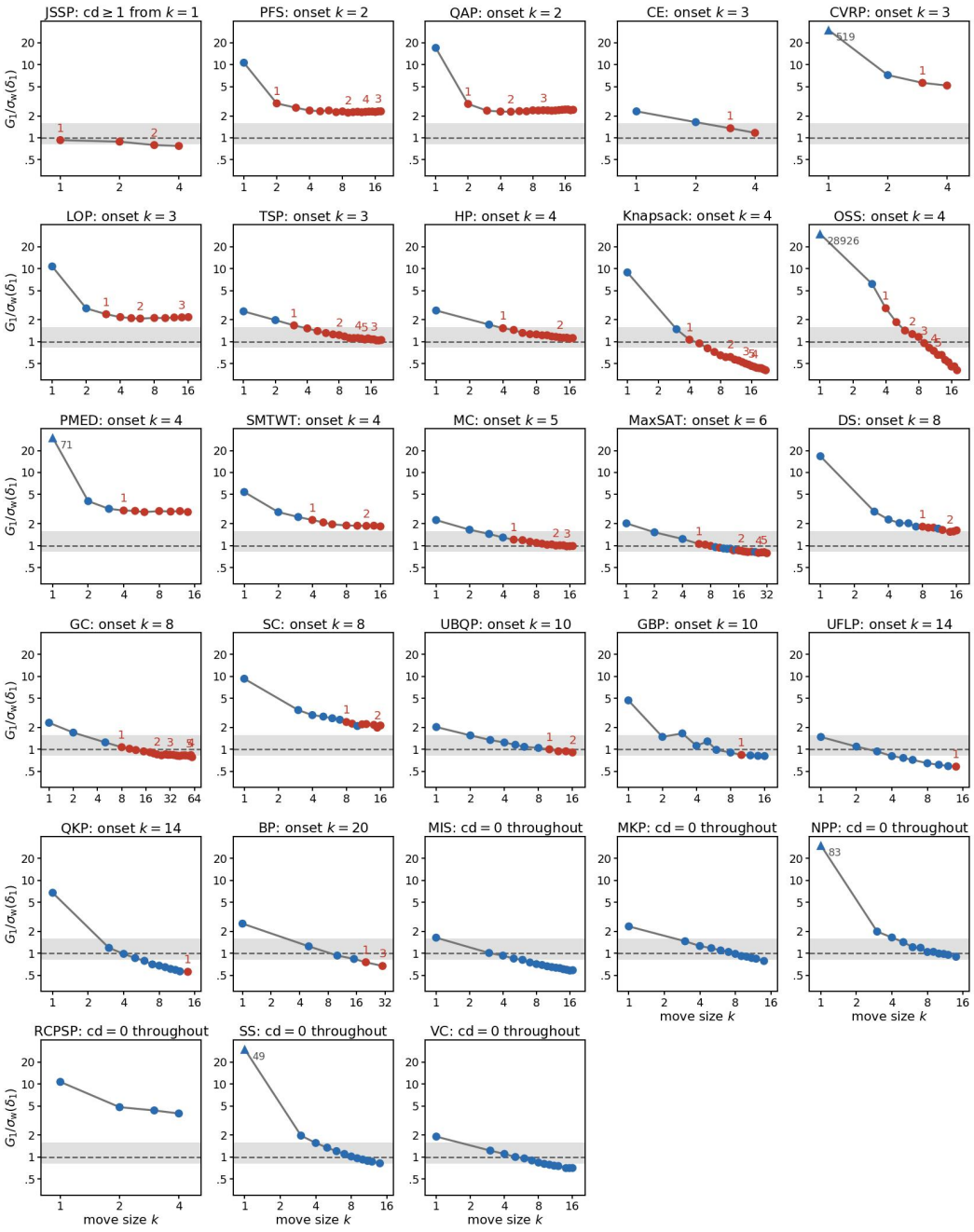

Figure 5: Saturation ratio $G_1/\sigma_{\mathrm{w}}(\delta_1)$ versus move size k for 28 families (log–log axes), ordered by onset k (the first k with a $\mathrm{cd}{\geq}1$ peak). Blue: observed peak at $\mathrm{cd}{=}0$; red: $\mathrm{cd}{\geq}1$, labeled with the peak depth wherever it changes. Dashed line: threshold of 1; shaded: the interval $(0.82, 1.58)$ that separates the twelve families of Table 26; triangles mark ratios above the axis range.

k is still above 1) and too early in $4/21$ (BP, UFLP, GBP, QKP). Over all 368 points its accuracy is 46.5%, below the 55.7% majority-class baseline.

Interpretation. As the move size grows, a positive peak depth is not confined to particular problem families: in $21/28$ families it appears beyond a family-specific move size, while the saturation ratio

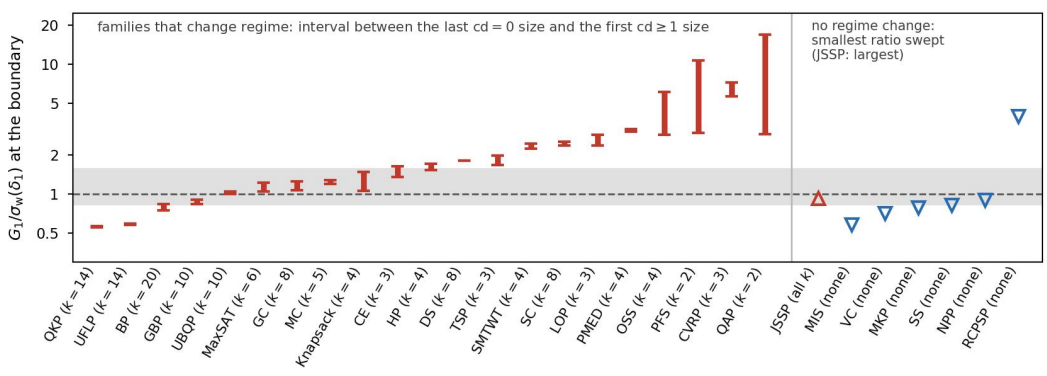

Figure 6: Saturation ratio at each family's transition. For each of the 21 families that change regime, the bar spans the ratios of the last cd=0 size and the first cd≥1 size (onset k in parentheses), sorted by midpoint. Right of the divider: JSSP (cd≥1 at every size; marker at its largest ratio) and the six families with no regime change (markers at their smallest ratio). Shaded: separating interval $(0.82, 1.58)$; dashed: threshold of 1.

falls as k grows. The threshold of 1 separates the families only at the default operators of Table 26. It is not a universal constant for the transition point along a move-size sweep; locating that point for a given family requires a direct sweep over k.

E.3 Statistical tests across all families

Table 30 reports statistical tests for the learned-minus-random difference at the peak commit depth for each family for which per-run or per-configuration data is available.

All seven families with repeated runs (the six domains and PFS) show $p < 0.01$ at the peak; QAP and MIS have only one configuration each but show large absolute differences. For RCPSP, the learned RF scorer is below random at every depth tested (-5.2, -7.1, -4.4 at cd=$0, 1, 2$; 3 test instances $\times$ 10 seeds), so it cannot distinguish candidates well enough for the cd≥1 advantage to appear. These learned runs therefore cannot test the prediction; an oracle scorer produces positive differences at every depth, showing that a usable within-state signal exists. The oracle peak itself depends on instance size: with 20 activities (where the ratio, 0.36, was measured) the oracle-minus-random difference is largest at cd=2 ($+4.0$ vs. $+3.1$ at cd=0), and with 40 activities (ratio 0.69) also at cd=2, but with 30 activities (ratio 0.24) at cd=0 ($+5.1$ vs. $+5.0$). These runs use unpaired initial solutions and no significance test, so the RCPSP entry of Table 26 rests on weak evidence. For CVRP and BRP we report oracle sweeps only: with the simplified neighborhoods of Table 26 (Variant I in Appendix E.7), the oracle-minus-random difference peaks at cd=0 and shrinks at every deeper setting (CVRP: $+88.2$, $+2.9$, $+1.2$, $+0.6$; BRP: $+4.60$, $+0.70$, $+0.56$, $+0.40$ for cd $\in \{0, 1, 2, 3\}$), matching the predicted cd=0 regime.

E.4 Motivation for the threshold value 1

The threshold value 1 has a simple break-even interpretation. The numerator and denominator of the saturation ratio are measured in the same objective units, so the ratio is dimensionless, and 1 is the point at which the two are equal. Consider the value of one descent step inside the loop: the step contributes an expected gain of G_1 to the objective (the first-step descent gain, by definition) but leaves residual within-state variation $\sigma_{\text{within}}(\delta_1)$ — the only signal a scorer can exploit when ranking candidates. The break-even point is where the marginal gain equals the remaining within-state variation:

$$G_1 \;=\; \sigma_{\text{within}}(\delta_1) \quad\Longleftrightarrow\quad \frac{G_1}{\sigma_{\text{within}}(\delta_1)} \;=\; 1.$$

When the ratio exceeds 1, the gain realized by the descent step exceeds the within-state variation it leaves: further descent has diminishing returns, and selecting the next move without descent (cd=0)

Table 29: Move-size sweep across 28 families (best-improvement descent over the full neighborhood). Onset: first k with observed peak $\mathrm{cd}{\geq}1$. Boundary: ratio interval between the last $\mathrm{cd}{=}0$ size and the onset. Overlap: whether that interval overlaps $(0.82, 1.58)$. Max depth: largest observed peak depth. Abbreviations: CE cluster editing, DS dominating set, GBP graph bisection, HP shortest Hamiltonian path, LOP linear ordering, MC max-cut, MKP multiple (multi-bin) knapsack, NPP number partitioning, OSS open-shop scheduling, PMED p-median, QKP quadratic knapsack, SC set cover, SMTWT single-machine total weighted tardiness, SS subset sum, UBQP unconstrained binary quadratic programming, UFLP uncapacitated facility location, VC vertex cover.

Family	k range	Onset k	Boundary ratio	Overlap	Max depth	Beyond onset
JSSP	1–4	1	—	—	2	$\geq$1 at all k (ratio 0.93 at k=1)
PFS	1–18	2	(2.96, 10.69)	no	4	stays $\geq$1
QAP	1–19	2	(2.91, 17.06)	no	3	stays $\geq$1
CE	1–4	3	(1.35, 1.64)	yes	1	stays $\geq$1
CVRP	1–4	3	(5.66, 7.23)	no	1	stays $\geq$1
LOP	1–16	3	(2.38, 2.86)	no	3	stays $\geq$1
TSP	1–20	3	(1.67, 1.97)	no	5	stays $\geq$1
HP	1–17	4	(1.53, 1.71)	yes	2	stays $\geq$1
Knapsack	1–22	4	(1.06, 1.48)	yes	5	stays $\geq$1
OSS	1–18	4	(2.87, 6.17)	no	5	stays $\geq$1
PMED	1–14	4	(3.02, 3.18)	no	1	stays $\geq$1
SMTWT	1–16	4	(2.24, 2.46)	no	2	stays $\geq$1
MC	1–17	5	(1.20, 1.29)	yes	3	stays $\geq$1
MaxSAT	1–32	6	(1.05, 1.23)	yes	5	0 again at 6/20 larger sizes
DS	1–16	8	(1.81, 1.82)	no	2	0 again at 1/7 larger sizes
GC	1–60	8	(1.07, 1.25)	yes	5	stays $\geq$1
SC	1–16	8	(2.37, 2.54)	no	2	0 again at 1/7 larger sizes
UBQP	1–16	10	(1.00, 1.05)	yes	2	stays $\geq$1
GBP	1–16	10	(0.84, 0.91)	yes	1	0 again at 3/3 larger sizes
UFLP	1–14	14	(0.58, 0.59)	no	1	stays $\geq$1
QKP	1–14	14	(0.56, 0.56)	no	1	stays $\geq$1
BP	1–30	20	(0.75, 0.83)	yes	3	stays $\geq$1
MIS	1–17	—	—	—	0	none (min ratio 0.58)
MKP	1–14	—	—	—	0	none (min ratio 0.78)
NPP	1–14	—	—	—	0	none (min ratio 0.90)
RCPSP	1–4	—	—	—	0	none (min ratio 3.96)
SS	1–14	—	—	—	0	none (min ratio 0.82)
VC	1–16	—	—	—	0	none (min ratio 0.71)

preserves more of the scorer's advantage. When the ratio is below 1, usable variation remains after the step, so running descent before the next selection is beneficial. We emphasize that this is a motivating interpretation, not a derivation: a formal result would require assumptions about the descent operator and the scorer error that we have not verified. The exact value 1 is not critical at the default operators: any threshold in the interval $(0.82, 1.58)$, found post hoc, separates the two regimes in all twelve families (Appendix E.5).

E.5 Threshold sensitivity

Across the twelve families (Table 26), any threshold in $(0.82, 1.58)$ separates the two regimes without error, a range spanning a factor of $\sim$2 around 1. The threshold was set on JSSP, TSP, and MaxSAT (any value in $(0.82, 2.27)$ separates them) and was not re-tuned on the other nine families. The interval $(0.82, 1.58)$ is found post hoc and holds for the default operators of Table 26; along move-size sweeps the transition ratio is family-specific (Appendix E.2).

E.6 Sampling variability of the ratio

The estimation procedure of Appendix E computes $G_1/\sigma_{\text{within}}(\delta_1)$ from $\sim$50 random (state, move) pairs. To quantify how sensitive the regime decision is to which pairs are drawn, we collect a large pool of pairs per family (120 states $\times$ 15 moves $=$ 1,800 pairs; for JSSP, synthetic $10{\times}10$ instances) and, for each sample size, draw 1,000 resamples of 50 pairs (10 states $\times$ 5 moves), 100

Table 30: Learned-minus-random difference at the peak cd, with p-values and effect sizes where available (n: runs; for PFS, configurations). Families above the line are in the $\mathrm{cd}{=}0$ regime; families below it are in the $\mathrm{cd}{\geq}1$ regime.

Family	Peak	n	Diff.	p	Cohen's d
TSP	0	50	$+2.63$	$<10^{-36}$	5.12
MaxSAT	0	50	$+6.82$	$<10^{-13}$	1.89
Knapsack	0	50	$+493.9$	$<10^{-25}$	3.66
GC	0	50	$+30.18$	$<10^{-38}$	5.81
BP	0	50	$+24.92$	$<10^{-23}$	2.75
PFS	0	9	$+16.17$	0.0023	1.46
QAP	0	1	$+2497$	—	—
MIS	0	1	$+954$	—	—
CVRP*	0	50	$+88.2$	0.0071	0.40
BRP*	0	50	$+4.60$	$<10^{-22}$	2.53
JSSP	1	50	$+220.2$	$<10^{-22}$	2.52
RCPSP*	≥1	—	>0 (oracle)	—	—

*Oracle scorer only (no learned-minus-random difference): for RCPSP the learned RF underperforms random (see text); for CVRP and BRP the entries are oracle-minus-random differences from a paired oracle sweep ($n{=}50$ instance–seed pairs).

pairs (10×10), or 200 pairs (20×10), recompute the ratio, and check whether the implied regime matches the observed one. For JSSP we repeat this on the Taillard sample of Table 26 (ta09–ta10, 200 states $\times$ 15 moves). Both JSSP samples contain no null moves.

Table 31: Fraction of 1,000 resampled estimates that predict the correct regime, by sample size. Pool-level ratios are computed on newly collected pools and can differ from the values in Table 26 (PFS: 2.52 vs. 4.52; JSSP: 0.94 vs. 0.82), but all lie on the same side of the threshold.

Family	Pool ratio	$n{=}50$	$n{=}100$	$n{=}200$
MaxSAT	2.68	100%	100%	100%
Knapsack	2.76	100%	100%	100%
GC	2.93	100%	100%	100%
BP	2.87	100%	100%	100%
QAP	3.44	100%	100%	100%
MIS	1.62	100%	100%	100%
PFS	2.52	100%	100%	100%
JSSP (synthetic pool)	0.94	30.6%	55.5%	58.3%
JSSP (Taillard sample)	0.82	52.5%	82.9%	91.8%

For the seven families whose ratios lie well away from the threshold, every one of the 1,000 fifty-pair estimates selects the correct regime. JSSP, the one family near the threshold, is sensitive to sampling: the resampled ratio falls on both sides of 1. On the synthetic pool (ratio 0.94), only 30.6% of fifty-pair estimates and 58.3% of 200-pair estimates select the correct regime; on the Taillard sample (ratio 0.82), the shares are 52.5% and 91.8%. This quantifies the caveat that predictions close to a ratio of 1 are less certain. A larger sample helps only when the ratio is far enough from 1; at a ratio of 0.94, 200 pairs still select the wrong regime in 41.7% of draws, and a direct commit-depth sweep on a few instances is the safer choice.

E.7 Implementation robustness

The saturation ratio depends on the descent operator because both of its terms are computed after one descent step. Different local-search implementations for the same problem can therefore fall into different regimes. We test three families under two implementation variants each: *Variant I*, the simplified neighborhood used in the main results (an adjacent swap in a precedence-feasible activity list for RCPSP; relocate moves with a sampled best-of-80 descent step for CVRP; a single block relocation with a sampled best-of-20 descent step for BRP), and *Variant II*, a standard neighborhood from the literature (Or-opt with $L{=}1, 2, 3$, exchange, and intra-route 2-opt for CVRP (Toth & Vigo,

2014); all single-block relocations for BRP; all pairwise swaps in a priority list, decoded by the serial schedule generation scheme, for RCPSP). Our BRP is a layout-search variant: a solution is a stack layout, a move relocates the top block of one stack onto another, and the cost is the number of relocations needed to retrieve all blocks in order under a fixed rule (a relocated block goes to the stack with the smallest top that is retrieved later, else to the stack with the latest top); the classical BRP instead optimizes the relocation sequence itself (Caserta et al., 2012).

Table 32: Implementation robustness: same family, two implementations. The threshold of 1 agrees with the observed regime for all 6 (family, variant) pairs. For RCPSP, Variant II's stronger descent leaves less within-state spread ($\sigma_{\mathrm{w}}(\delta_1)$ shrinks) and shifts the regime to $\mathrm{cd}{=}0$; CVRP and BRP are in the $\mathrm{cd}{=}0$ regime under both variants. Ratios are computed from unrounded statistics.

Family	Variant	G_1	$\sigma_{\mathrm{w}}(\delta_1)$	Saturation ratio	Pred	Obs
RCPSP	I (adjacent swap)	0.43	1.20	0.36	≥ 1	≥ 1
RCPSP	II (all pairwise swaps)	1.96	0.55	3.56	0	0
CVRP	I (relocate, sampled)	270.6	37.9	7.15	0	0
CVRP	II (Or-opt, exchange, 2-opt)	3933	3.83	1027	0	0
BRP	I (single relocation)	1.87	0.82	2.26	0	0
BRP	II (all relocations)	0.91	0.43	2.13	0	0

RCPSP switches regime when descent becomes more thorough, and its saturation ratio rises accordingly ($0.36 \rightarrow 3.56$): a stronger descent step raises G_1 and shrinks $\sigma_{\mathrm{w}}(\delta_1)$, so the ratio rises through the threshold. CVRP and BRP lie above the threshold under both variants and stay at $\mathrm{cd}{=}0$. The dependence is by construction: $\sigma_{\mathrm{w}}(\delta_1)$ is measured after a descent step, so any change in the descent algorithm propagates into the ratio. A ratio computed for one implementation is therefore a statement about *that* implementation, not a claim about the problem family in general.

F COMMIT DEPTH IN SIX PUBLISHED METHODS

This appendix gives the protocol and the full results of the study in Section 4. MIStar is treated in Appendix G and the fine-tuning runs of Section 5 in Appendix H.

F.1 COMMON PROTOCOL

We use the authors' code, released checkpoints, test instances, initial solutions, and acceptance criteria. No checkpoints are released for the learned SAT heuristic, so we train models with the authors' code (Appendix F.5). The only change to the search loop is a descent of up to cd steps after each applied move (Algorithm 1); $\mathrm{cd}{=}0$ is the original loop. Each descent step applies the best improving move of the neighborhood in Table 33, evaluated exactly, and descent stops early at a local optimum (for MIStar, see Appendix G). $\mathrm{cd}{=}\mathrm{LO}$ descends to a local optimum, up to the cap in Table 33. Descent steps do not count toward the step budget T.

Random selection replaces the learned choice by a uniformly random move from the same action space, with the same masks. Everything else is shared: the loop, the descent, the step budget, the instances, the initial solutions, and the random seeds. Comparisons are paired over instances (over (instance, seed) pairs for L2S and MIStar). We report paired t-tests unless stated otherwise. W/T/L counts the instances on which the first-named selector is better, equal, or worse. "Each selector at its best descent setting" uses, for each selector separately, the (descent, cd) setting with the best mean on the test instances; this setting is chosen on the test instances, in the same way for both selectors.

Plain and tabu-restricted descent. Plain descent (descent without a tabu restriction) can undo the selected move. A tabu restriction forbids this during the descent that follows a selected move and is lifted at the next selection. For the edge-exchange moves of DACT and NeuOpt we use two restrictions. Under the *edge tabu*, no descent move may re-add an edge that the selected move removed. Under the *reversal tabu*, no single descent move may re-add all removed edges; for a 2-opt move this forbids only the reverse move. For ECO-DQN and SAT the vertex or variable just flipped by the selector may not be flipped again during the descent. For L2S the descent may not reverse the selected swap. We say that descent *undoes* a move when the solution after the descent equals the

solution before the move (ECO-DQN, SAT, L2S: when a descent flip or swap reverses the selected one).

Table 33: Setup of the commit-depth study. T: policy steps per run. Runs: independent runs per instance. Reference: solution used for the optimality gap or approximation ratio.

Method	Test instances	T	Runs	Descent step	LO cap	Reference
DACT	TSP-50, TSP-100: first 128 instances of the authors' test files	500	5 / 3	best improving 2-opt move	50	LKH-3 (10 runs)
NeuOpt	CVRP-50: 1000 instances; CVRP-100: 500 instances (authors' test files)	500	1	best improving intra-route 2-opt or inter-route relocation that keeps capacity	—	HGS (PyVRP), best known
ECO-DQN	Max-Cut ER and BA, 40 and 60 vertices, ER 200 vertices: 100 graphs each (authors' evaluation sets)	$2\|V\|$	50 (10 for ER200)	best improving single-vertex flip	$\|V\|$	authors' reference cuts
SAT	random 3-SAT, 25 variables/106 clauses and 50/213: 100 formulas each	750 flips	25	flip that most reduces the number of unsatisfied clauses	100	—
L2S	Taillard 15×15 ta01–10	5000	3	best improving $\mathcal{N}_5$ swap (exact makespan)	50	optimal makespans
MIStar	FJSP SD2 10×5+mix, 10 instances	100	3	best of the first 50 feasible moves (Appendix G)	—	initial makespan

F.2 DACT ON TSP

DACT (Ma et al., 2021) selects a 2-opt move (a pair of nodes) with a policy network and samples from it. We use the released TSP-50 and TSP-100 checkpoints, the repository's greedy initial tours, its restart rule (restart after 250 steps without improvement), and no test-time augmentation. With $\mathrm{cd}{=}0$ our loop reproduces the repository's rollout function exactly: the best tour length is identical at every step of the anytime curve (TSP-50, 128 instances, one seed). At 1000 steps the learned policy reaches a gap of 0.11% (one seed). A re-run of the published policy on an RTX 4090 gave 0.289% at 500 steps against 0.280% in the stored RTX 2080 Ti runs ($p{=}0.75$).

Table 34 gives the TSP-50 results. Plain descent undoes almost every move of both selectors (97–100%), so every plain-descent setting gives a gap of 1.7–2.3%, far worse than the learned policy without descent (0.28%). Under edge tabu, descent helps the learned policy (best: 0.079% at LO), while random selection needs LO to reach 0.23%. Under reversal tabu, random selection with descent to a local optimum reaches 0.050%, the best result of either selector. With each selector at its best setting, random selection is ahead by 0.028 pp, which is not significant (learned 0.079%, random 0.050%; t-test $p{=}0.050$, Wilcoxon $p{=}0.12$, W/T/L $32/58/38$). At $\mathrm{cd}{=}0$ the learned policy is ahead by 23.1 pp.

Table 35 gives TSP-100 (3 runs). The pattern is the same. Plain descent undoes 96–100% of the moves of both selectors. The learned policy is best with edge tabu at LO (0.833%) and random selection with reversal tabu at LO (0.835%); with the same number of steps the two are tied. At equal wall-clock time random selection is ahead, because its steps are cheaper (Appendix H.2). In the four evaluations of the fine-tuning study (Appendix H.7), random selection with this descent reaches 0.46% in the fine-tuned policy's time, and the original loop at $\mathrm{cd}{=}0$, run for three times as many steps, reaches 1.27–1.45% (in the two first-seed evaluations its 1500 steps end before that time).

Compute. Descent is batched over the 128 instances on the GPU. On an RTX 4090 (TSP-50; one step of all 128 instances), a step of the published policy takes 6.3 ms at $\mathrm{cd}{=}0$ and 17.6 ms with edge tabu descent to a local optimum; a random step with reversal tabu descent to a local optimum takes 12.1 ms. On TSP-100 the three figures are 9.1, 23.2, and 12.9 ms. Relative to $\mathrm{cd}{=}0$, a step of the learned policy costs 1.34, 1.47, and 1.36 times as much with plain descent at $\mathrm{cd}{=}1$, 2, LO, and 2.63 and 2.01 times with edge tabu and reversal tabu at LO (TSP-50, RTX 2080 Ti, median of three rounds).

TSP-200. DACT, like the other methods, was designed for a loop without descent. No TSP-200 checkpoint is released, so we run the TSP-100 checkpoint on the first 128 instances of the authors' TSP-200 file with five seeds; the reference is LKH-3 (10 and 50 runs give the same tour lengths,

Table 34: DACT, TSP-50 (128 instances, 5 runs, 500 steps; gap to LKH-3 in %, lower is better). Diff.: random minus learned (pp; positive = learned better). W/T/L over instances (mean over runs), learned first. Undone: share of steps after which the tour after descent equals the tour before the selected move (L/R: learned/random; at cd=0 and under edge tabu this counts only moves that leave the tour unchanged). Desc.: descent moves per policy step.

Descent	cd	Learned	Random	Diff.	p	W/T/L	Undone L/R	Desc. L/R
none	0	**0.280**	23.401	+23.12	$<10^{-62}$	128/0/0	0.02/0.04	0/0
plain	1	1.693	2.286	+0.59	$<10^{-8}$	99/2/27	0.98/0.97	0.95/0.97
	2	1.668	2.275	+0.61	$<10^{-7}$	92/12/24	0.99/0.98	0.96/0.98
	LO	1.695	2.250	+0.56	$<10^{-6}$	87/16/25	0.99/1.00	0.97/0.98
edge tabu	1	0.169	18.636	+18.47	$<10^{-69}$	128/0/0	0.01/0.00	0.82/1.00
	2	0.128	6.106	+5.98	$<10^{-81}$	128/0/0	0.01/0.00	1.15/2.00
	LO	**0.079**	0.232	+0.15	$<10^{-10}$	104/9/15	0.02/0.01	1.46/6.95
reversal tabu	1	0.187	18.706	+18.52	$<10^{-68}$	128/0/0	0.01/0.00	0.83/1.00
	2	0.188	4.810	+4.62	$<10^{-76}$	128/0/0	0.02/0.00	1.16/2.00
	LO	0.205	**0.050**	−0.15	$<10^{-3}$	25/43/60	0.08/0.63	1.41/3.63

Table 35: DACT, TSP-100 (128 instances, 3 runs, 500 steps; gap in %). Columns as in Table 34, without the Undone and Desc. columns.

Descent	cd	Learned	Random	Diff.	p	W/T/L
none	0	2.290	25.218	+22.93	$<10^{-76}$	128/0/0
plain	1	3.115	4.862	+1.75	$<10^{-24}$	112/0/16
	LO	3.267	3.833	+0.57	$<10^{-7}$	93/1/34
edge tabu	1	1.233	22.920	+21.69	$<10^{-79}$	128/0/0
	2	1.017	15.509	+14.49	$<10^{-87}$	128/0/0
	LO	**0.833**	1.378	+0.54	$<10^{-18}$	104/0/24
reversal tabu	1	1.228	22.903	+21.68	$<10^{-79}$	128/0/0
	2	1.320	14.042	+12.72	$<10^{-86}$	128/0/0
	LO	1.393	**0.835**	−0.56	$<10^{-8}$	35/0/93

and no run of ours is shorter). We follow the repository's advice for larger instances: the positional encoding of the training size is resampled to 200 nodes, and the restart patience is P=5, the best value for the original loop at cd=0 in a sweep over $P \in \{1, 2, 3, 5, 10, 20, 50, 250\}$ on one seed (8.95% after 500 steps, against 17.50% with the default P=250). Random selection with tabu-restricted descent to a local optimum is best at P=1 in the same sweep (1.19% against 1.45% at P=5), so this choice favors DACT. Table 36 gives the results. At cd=0 the original loop reaches 8.86% after 500 steps and random selection 25.53% (+16.7 pp). Each at its best setting (edge tabu, LO, for both), random selection is ahead by 0.16 pp (1.50% against 1.66%, p=5×10^{-4}, W/T/L 52/0/76). In the time the original loop needs for 500 steps (18.3 s for all 128 instances on an A100 GPU), random selection with the same descent reaches 1.31%; the original loop reaches 5.99% after 5000 steps (three seeds). Without retraining, the published policy with tabu-restricted descent to a local optimum reaches 1.66% after 500 steps (27.2 s); at that time the original loop is at 8.19% (746 steps; the policy with descent is better on all 128 instances, $p<10^{-80}$), and random selection with the same descent is at 1.11% (991 steps). With the repository's default encoding (that of the target size) the published policy reaches 15.27% at cd=0 and 1.91% at its best descent setting, against 1.50% for random selection ($p<10^{-16}$). Plain descent undoes 97.5–99.8% of the moves of both selectors.

F.3 NeuOpt on CVRP

NeuOpt (Ma et al., 2023) selects a k-opt move ($k \leq 4$) by a factorized policy. It allows capacity-infeasible solutions during the search and steers back to feasible ones through its features and reward (guided infeasible region exploration, GIRE). We use the released CVRP-50 and CVRP-100 checkpoints, sampling, and no dynamic data augmentation. Random selection samples uniformly

Table 36: DACT on TSP-200 (TSP-100 checkpoint, 128 instances, 5 runs, 500 steps; gap to LKH-3 in %). Columns as in Table 34, without the Undone and Desc. columns. A step with edge tabu descent to a local optimum takes 55.2 ms for the learned policy and 26.8 ms for random selection (36.7 ms for the learned policy at cd=0; all 128 instances, A100).

Descent	cd	Learned	Random	Diff.	p	W/T/L
none	0	8.855	25.530	+16.68	$<10^{-71}$	128/0/0
plain	1	3.973	11.051	+7.08	$<10^{-80}$	128/0/0
	2	3.740	4.701	+0.96	$<10^{-22}$	112/0/16
	LO	3.849	4.668	+0.82	$<10^{-19}$	113/0/15
edge tabu	1	2.973	16.380	+13.41	$<10^{-93}$	128/0/0
	2	2.382	9.489	+7.11	$<10^{-82}$	128/0/0
	LO	**1.659**	**1.497**	−0.16	$<10^{-3}$	52/0/76
reversal tabu	1	2.990	16.377	+13.39	$<10^{-92}$	128/0/0
	2	2.325	8.307	+5.98	$<10^{-85}$	128/0/0
	LO	2.099	1.629	−0.47	$<10^{-16}$	24/0/104

from the same factorized action space with the same masks. The incumbent is the best capacity-feasible solution found. With cd=0 our loop reproduces the repository's rollout exactly (identical incumbent and current solution at every step). With the repository's default augmentation at 1000 steps, the published policy reaches a gap of 0.59% to HGS on our 1000 CVRP-50 instances; the paper reports 0.61% on 10,000 instances. A re-run of every plain-descent cell on another machine gave identical trajectories on 994 of the 1000 instances and no significant change in the mean gap (all $p \geq 0.13$).

Besides plain and tabu-restricted descent we use a *repair descent*, whose objective is the route length plus 1000 times the capacity excess, so that it can restore feasibility. Table 37 gives CVRP-50. Descent helps the learned policy (best: edge tabu, cd=50, 0.642% against 1.046% at cd=0); random selection gains a great deal (from 181.3% to 4.94% with repair descent at cd=50). With each at its best setting, the learned policy is ahead by 4.30 pp (W/T/L 983/0/17), against 180.3 pp at cd=0, a ratio of 41.9 (95% bootstrap interval $[40.5, 43.2]$). The learned policy's best setting at 500 steps is level with the learned policy at cd=0 run for 1000 steps (0.617%, p=0.53). At 200 steps the learned policy's best setting is again edge tabu at cd=50 (1.28%) and random selection's is repair descent at cd=50 (6.04%); the ratio is 38.0.

Table 37: NeuOpt, CVRP-50 (1000 instances, 500 steps; gap to the best known solution in %; L/R: learned/random). Every learned-minus-random difference has $p<10^{-100}$ (paired t-test). Undone: share of non-null selected moves after which the solution after descent equals the solution before the move (plain descent only).

	plain		edge tabu		reversal tabu		repair		plain undone
cd	L	R	L	R	L	R	L	R	L/R
0	1.046	181.30	—	—	—	—	—	—	—
1	0.721	63.26	0.684	65.09	0.845	63.97	1.956	68.68	0.23/0.02
2	0.710	33.29	0.748	35.86	0.683	33.48	2.103	32.56	0.28/0.01
5	0.766	11.60	0.699	13.95	0.726	11.86	2.203	9.96	0.30/0.03
10	0.699	7.83	0.666	8.72	0.736	7.84	—	—	0.29/0.13
50	0.766	6.78	**0.642**	7.37	0.682	6.68	2.393	**4.94**	0.30/0.18

On CVRP-100 (500 instances, 500 steps) the learned policy is at 2.57% and random selection at 270.7% at cd=0. Random moves almost never yield a capacity-feasible solution, and plain descent does not restore feasibility, so random selection with plain descent stays at 56.0% even at cd=50. With repair descent at cd=50 it reaches 9.76%. The learned policy's best setting is plain descent at cd=1 (1.82%). Each at its best, the learned policy is ahead by 7.94 pp (500/0/0), against 268.1 pp at cd=0.

CVRP-200. We run the released CVRP-200 checkpoint on the first 256 instances of the authors' CVRP-200 file with three seeds, with the protocol of the CVRP-50 study. The reference is the per-instance best of five PyVRP HGS runs (20,000 to 400,000 iterations) and all our runs; no run of ours is better than HGS, and the best single HGS run is 0.08% above this reference on average. Table 38 gives the results. At $\mathrm{cd}{=}0$ the published policy reaches 5.18% after 500 steps and random selection 421.4% ($+416.2$ pp). Only 0.8% of random moves leave a capacity-feasible solution, and plain or tabu-restricted descent rarely restores feasibility, so random selection needs the repair descent: with it, it reaches 14.36% at $\mathrm{cd}{=}200$. Each at its best setting (published: reversal tabu, $\mathrm{cd}{=}2$, 3.48%; random: repair, $\mathrm{cd}{=}200$), the published policy is ahead by 10.89 pp (W/T/L $256/0/0$), which is 38.2 times less than at $\mathrm{cd}{=}0$ (95% bootstrap interval $[37.4, 39.0]$). At equal wall-clock time the original loop at $\mathrm{cd}{=}0$ (5.18% after 500 steps, 95.7 s) stays well ahead of random selection with repair descent (16.27% in the same time). Plain descent undoes 10% ($\mathrm{cd}{=}1$) and 19% ($\mathrm{cd}{=}50$) of the published policy's non-null moves. In the implementation used here descent at $\mathrm{cd} \in \{1, 2\}$ adds 62–72% to the time of a step (191 ms at $\mathrm{cd}{=}0$ for 256 instances on an A100); in the implementation of the fine-tuning study it adds 5–10%.

Table 38: NeuOpt on CVRP-200 (256 instances, 3 runs, 500 steps; gap to the best known solution in %; L/R: learned/random). Every learned-minus-random difference has $p<10^{-37}$ (paired t-test). Random selection with repair descent at $\mathrm{cd}{=}200$: 14.36%.

	plain		edge tabu		reversal tabu		repair	
cd	L	R	L	R	L	R	L	R
0	5.178	421.40	—	—	—	—	—	—
1	3.694	341.22	3.573	345.21	3.588	345.24	5.378	260.73
2	3.586	353.24	3.525	354.34	**3.475**	352.48	5.739	154.71
10	3.733	349.37	3.606	341.66	3.666	348.77	6.988	33.08
50	3.758	81.15	3.663	64.40	3.729	81.20	7.033	14.84

Compute. One forward pass of the NeuOpt actor costs $9.75{\times}10^7$ floating-point operations per instance (matrix products only). One evaluation of the full descent neighborhood of a CVRP-50 solution costs $3.9{\times}10^4$ floating-point operations, so descent accounts for less than 0.09% of the operations of a run. In wall-clock time (batch of 1000 instances, median of three interleaved rounds) a learned step takes 243 ms at $\mathrm{cd}{=}0$ and 367 ms and 404 ms with plain and tabu-restricted descent at $\mathrm{cd}{=}1$. The descent itself takes 4–7 ms; most of the difference comes from recomputing the solution cost after the descent (152 ms) with the repository's cost function. In the fine-tuning evaluation on CVRP-100 (Appendix H; batch of 500 instances, RTX 4090), a step of the published policy takes 101.5 ms at $\mathrm{cd}{=}0$ and 107.7 ms with plain descent at $\mathrm{cd}{=}1$ ($+6\%$).

F.4 ECO-DQN ON MAX-CUT

ECO-DQN (Barrett et al., 2020) flips one vertex per step, chosen greedily by a Q-network; every episode has $2|V|$ steps from a random initial spin configuration, and the result of an episode is its best cut. We use the released checkpoints for each graph family and the authors' evaluation graphs and reference cuts. The metric is the approximation ratio (best cut divided by the reference cut), averaged over graphs and starts. Random selection flips a uniformly random vertex. With $\mathrm{cd}{=}0$ we reproduce the repository's own evaluation: mean single-episode ratio 0.9961 (ours) against 0.9962 (repository) on ER40, 0.9891 against 0.9885 on ER60, 0.9886 against 0.9887 on BA40, 0.9782 against 0.9794 on BA60, and best of 50 starts 1.0000 in all four; the evaluation code of Appendix H reproduces our stored episodes exactly (5000 of 5000 on ER40).

Table 39 summarizes the results. Plain descent lowers the learned policy's ratio by 2.5 to 4.6 pp, because its first flip reverses the policy's flip in 26–76% of steps (random selection: 75–97%). With the tabu restriction, descent raises the learned policy's ratio by 0.21 to 0.33 pp (at its best cd) and random selection's by 67 to 82 pp (at LO). With each selector at its best setting, the learned policy remains ahead by 1.07 to 2.46 pp, which is smaller by a factor of 34 to 69 than at $\mathrm{cd}{=}0$.

Compute. On an RTX 4090 (ER40, 50 starts in lockstep, one graph), 80 steps take 0.42 s at $\mathrm{cd}{=}0$ and 0.59 s with tabu-restricted descent to a local optimum for the published policy; 80 random steps

Table 39: ECO-DQN on Max-Cut (100 graphs per family; 50 starts per graph, 10 for ER200; $2|V|$ steps). Mean approximation ratio in % (higher is better). Best: each selector at its best setting over $\mathrm{cd} \in \{0, 1, 2, \mathrm{LO}\}$ and plain or tabu-restricted descent (tabu-restricted descent at LO unless marked). Diff.: learned minus random (pp). All differences have $p<10^{-17}$ (paired t-test over graphs). Undone: share of steps whose first descent flip reverses the selected flip (plain descent, $\mathrm{cd} \in \{1, 2, \mathrm{LO}\}$).

	cd=0			plain LO		each at its best				
Family	L	R	Diff.	L	R	L	R	Diff.	W/T/L	Undone L/R
ER40	99.61	30.42	+69.19	96.72	93.82	99.83	97.98	+1.85	100/0/0	0.70–0.76/0.77–0.89
ER60	98.91	24.37	+74.53	95.20	92.83	99.23	97.62	+1.61	99/0/1	0.39–0.42/0.78–0.92
BA40	98.86	28.35	+70.51	96.36	93.37	99.13	97.79	+1.34	98/0/2	0.38–0.39/0.76–0.87
BA60	97.82	23.86	+73.96	95.19	92.37	98.09[1]	97.02	+1.07	87/1/12	0.26–0.28/0.75–0.87
ER200	98.18	13.64	+84.54	93.57	91.52	98.39	95.93	+2.46	100/0/0	0.58–0.65/0.78–0.97

[1] tabu, cd=1.

with the same descent take 0.46 s. On ER200 (10 starts, 400 steps) the figures are 1.75, 2.07, and 1.25 s.

F.5 A LEARNED SAT HEURISTIC

The heuristic of Yolcu & Póczos (2019) is a graph neural network that picks the variable to flip in a WalkSAT-style search: with probability $1/2$ the flip is a random-walk step (a random variable of a random unsatisfied clause), otherwise the network samples a variable. Random selection replaces the network's choice by a uniformly random variable; the random-walk steps are unchanged. Our loop follows the authors' evaluation code. The budget is 750 flips per trial, counting random-walk and selector flips but not descent flips, with 25 trials per formula from the same initial assignments for every selector and cd. The metric is the share of trials that reach a satisfying assignment within the budget. Tests are Wilcoxon signed-rank tests over the 100 formulas.

No trained models are released. We trained models with the authors' code and curriculum on random 3-SAT with 5, 10, 25, and 50 variables, but with 2000, 2000, 1500, and 300 iterations per stage instead of 10,000; a second 50-variable model continues the last stage for 1500 iterations. By the paper's criterion (median number of flips below the budget), our 50-variable model solves 57% of the 50-variable formulas (continued model: 61%), against 84% reported by the authors. Re-running blocks of the study on another machine reproduced every trial exactly, except one trial in each of two blocks.

Table 40 gives the results. With plain descent the learned policy is behind random selection at every $\mathrm{cd} \geq 1$ (by 1.7 to 9.8 pp; not significant in one of nine cells), and random selection with cd=1 comes within 12 to 18 pp of the learned policy at cd=0. With the tabu restriction (the variable just flipped by the selector may not be flipped in the descent), random selection at LO solves more trials than the learned policy at its best setting. Plain descent undoes 36–49% of the network's flips (15–17% for the continued model) and 82–86% of random flips.

Table 40: Learned SAT heuristic: share of successful trials (%, higher is better) for 100 formulas × 25 trials. Models trained by us (Appendix F.5); "50 cont.": the continued 50-variable model. Diff.: learned minus random at each selector's best setting, with Wilcoxon p.

		cd=0		best, plain descent			best, with tabu restriction		
Formulas	Model	L	R	L	R	Diff.	L	R	Diff. (p)
25/106	25	88.0	14.6	88.0[0]	75.6[1]	+12.5	96.8	99.5	−2.8 (0.002)
50/213	50	57.2	0.0	57.2[0]	42.7[1]	+14.4	74.6	87.2	−12.6 ($<10^{-9}$)
50/213	50 cont.	60.8	0.0	60.8[0]	42.7[1]	+18.0	73.3	87.2	−13.9 ($<10^{-12}$)

[0] at cd=0; [1] at cd=1. With the tabu restriction both selectors are best at LO.

F.6 L2S ON JSSP

L2S (Zhang et al., 2024) selects an $\mathcal{N}_5$ swap with a graph neural network policy, starting from an FDD/MWKR dispatching solution; its environment keeps a tabu list of length one. We use the released 10×10 checkpoint on Taillard 15×15 (ta01–10), 5000 policy steps, and three seeds. Descent steps are applied through the environment, so they update the tabu list: at $\mathrm{cd}{=}1$ the environment's tabu list already forbids undoing the policy's swap, and from the second descent step on plain descent can undo it. Our implementation computes makespans with an exact critical-path routine instead of the repository's message-passing evaluator; it gives the same makespans and is faster.

Weight loading. In a first run of this study, eight projection matrices of the graph-attention layers kept their random initial values instead of the released weights. Our loader, reused from earlier code, renamed parameters for an older version of the graph library and loaded the checkpoint without strict key matching; under the library version of this run the renamed keys did not match. We fixed the mapping and now load with strict matching. All L2S results in this paper use the correctly loaded weights. An earlier run with one seed under the older library version, under which the renamed keys match (learned 7.74% at $\mathrm{cd}{=}0$ and 4.09% at $\mathrm{cd}{=}1$ after 5000 steps), does not differ significantly from the current runs on the same ten instances and seed (paired $p{=}0.27$ and 0.24).

Table 41 gives the results. At $\mathrm{cd}{=}0$ the learned policy is ahead of random selection by 2.48 pp ($p{<}10^{-8}$, $28/0/2$). One descent step lowers the learned policy's gap from 6.75% to 4.67%, below the 5.49% stored in the authors' repository for this checkpoint at 5000 steps (the paper reports 6.2% with its 15×15 checkpoint). Our $\mathrm{cd}{=}0$ runs are above the repository values at every step count. Our loop evaluates one instance at a time with batch normalization in evaluation mode; the authors' test loop, which evaluates a batch of instances with batch statistics, gives 9.43, 7.83, 6.95, and 5.92% after 500, 1000, 2000, and 5000 steps in our re-run with correctly loaded weights. Random selection gains more from descent and is ahead at every $\mathrm{cd}{\geq}1$. The learned policy is best with tabu-restricted descent to a local optimum (4.17%). Over all 30 pairs random selection is best with tabu-restricted descent to a local optimum (2.52%), and the learned policy's best setting is 1.65 pp worse ($p{=}5.1{\times}10^{-6}$, W/T/L $3/0/27$). On the ten (instance, seed) pairs on which every setting was run, random selection is best with plain descent at $\mathrm{cd}{=}2$ (2.13% against 2.16% with tabu LO); over all 30 pairs the learned policy's best setting is 1.52 pp worse than this setting ($p{=}3.0{\times}10^{-5}$, $5/0/25$). Plain descent undoes the policy's swap in 36% ($\mathrm{cd}{=}2$) and 78% (LO) of steps (random selection: 42% and 75%). If the environment's tabu list is also ignored during descent, one descent step undoes 99% of the policy's swaps and the gap rises to 12.0% (one seed).

Table 41: L2S on Taillard 15×15 (ta01–10, 3 seeds, 30 pairs): mean optimality gap (%, lower is better) after 500 to 5000 policy steps. Repository: the $\mathrm{cd}{=}0$ test results stored in the authors' repository for this checkpoint. Diff.: random minus learned at 5000 steps (pp; positive = learned better), paired over (instance, seed).

Selector	Descent	500	1000	2000	5000	Diff.	p (W/T/L)
repository	none	8.83	6.34	5.92	5.49		
learned	none	10.50	9.38	7.94	6.75	+2.48	$<10^{-8}$ (28/0/2)
	plain, cd=1	7.54	6.43	5.65	4.67	−0.86	0.003 (8/1/21)
	plain, cd=2	6.73	5.65	5.07	4.47	−1.82	$<10^{-7}$ (2/0/28)
	tabu, cd=2	6.89	5.97	5.39	4.85	−2.11	$<10^{-5}$ (2/1/27)
	plain, LO	8.27	7.79	7.58	7.30	−1.76	0.017 (8/0/22)
	tabu, LO	6.60	5.67	5.13	**4.17**	−1.65	$<10^{-5}$ (3/0/27)
random	none	12.98	11.60	10.39	9.23		
	plain, cd=1	6.80	5.58	4.87	3.81		
	plain, cd=2	5.39	4.46	3.47	2.65		
	tabu, cd=2	5.12	4.02	3.35	2.74		
	plain, LO	6.36	5.91	5.62	5.53		
	tabu, LO	4.58	3.74	3.09	**2.52**		

The same pattern holds on 100 synthetic 10×10 instances (one seed): after 5000 steps the learned policy is ahead by 1.74 pp at $\mathrm{cd}{=}0$, and random selection is ahead by 0.71 to 1.06 pp under every descent setting (all $p{<}0.001$).

Larger Taillard sets. L2S, like the other methods we study, was designed and evaluated for a loop without descent, and its evaluation did not include random selection with descent as a baseline. Table 42 extends the comparison to the Taillard sets used in the L2S paper (one seed, 10 instances per set, 5000 steps). Following the paper, we use the released checkpoint trained on the closest size (15×15 for ta01–10 and 20×15 for the larger sets); we also run the 10×10 checkpoint. Our re-run of the authors' test loop with correctly loaded weights is within about 0.6 pp of the results stored in the repository, except for the 15×15 checkpoint on ta01–10 (7.13 against 6.16%). With its best descent setting, L2S improves on its original loop (cd=0) on every set. With each selector at its best descent setting, random selection is ahead of L2S at equal wall-clock time by 1.2 to 2.1 pp on 20×20, 30×15, and 30×20, and not significantly different on 50×15 (p=0.29), where both are below 0.4%. Plain descent at cd=2 stalls with random selection on 30×15 and 50×15; there the tabu-restricted descent is the better random baseline. On 50×15 all random runs ended (16,000 steps) before the time of the L2S runs, so their final values are used, which favors L2S. On the four larger sets a step takes 37–79 ms for L2S with tabu-restricted descent to a local optimum and 15–32 ms for random selection with the same descent.

Table 42: L2S on larger Taillard sets (ta01–10, ta21–30, ta31–40, ta41–50, ta51–60; one seed, 10 instances per set, 5000 steps): mean optimality gap (%). Paper: the values reported by Zhang et al. (2024) (closest-size checkpoint). Authors' loop: our re-run of the authors' test loop with correctly loaded weights. Authors' loop and L2S columns: closest-size / 10×10 checkpoint. Last two columns: best random minus best L2S (over both checkpoints; each at its best descent setting; negative = random better) with the same number of steps and at equal wall-clock time, with paired t-test p.

			L2S		random		best random − best L2S	
Size	Paper	Authors' loop	cd=0	tabu, LO	plain, cd=2	tabu, LO	same steps	equal time
15×15	6.2	7.13 / 5.92	7.07 / 6.92	5.09 / 3.77	2.13	2.16	−1.64 (0.027)	−2.15 (0.005)
20×20	9.0	8.84 / 8.66	8.93 / 9.49	4.98 / 6.18	4.56	4.69	−0.42 (0.55)	−1.20 (0.016)
30×15	9.0	9.24 / 9.02	9.05 / 9.81	5.29 / 7.14	9.33	3.75	−1.55 (0.005)	−2.10 (0.002)
30×20	12.6	12.76 / 12.93	12.70 / 13.68	7.61 / 6.78	7.63	5.89	−0.88 (0.011)	−1.62 ($<10^{-3}$)
50×15	4.6	4.20 / 4.33	3.27 / 4.17	0.24 / 0.31	5.42	0.15	−0.09 (0.35)	−0.18 (0.29)

Compute. Per policy step on one CPU thread, the learned policy takes 15.2 ms at cd=0, 18.9 ms at cd=1, and 20.5 ms with tabu-restricted descent to a local optimum; random selection takes 2.0, 5.6, and 10.5 ms. The learned policy at cd=0 given the time of 5000 steps with tabu LO (6300–6800 steps) reaches 6.37–6.44%, still 2.2 pp worse than with descent ($p<10^{-6}$). Run for 40,000 steps (8 times as many, one seed), it reaches 4.99%.

F.7 MISTAR ON FJSP

The published MIStar policy (Wang et al., 2025) is almost uniform over feasible moves, and almost all of its advantage over the single-move random baseline comes from computing the exact makespan of 50 candidates per step. Appendix G gives the analysis and the commit-depth sweep.

G MISTAR: AN ALMOST UNIFORM PUBLISHED POLICY

Setup. MIStar improves FJSP schedules with a graph neural network policy over the neighborhood of Wang et al. (2025). We use the authors' code, the released 10×5 checkpoint, and the first 10 SD2 10×5+mix instances. For each instance, 20 initial schedules are built with DANIEL (Wang et al., 2024), the learned construction policy used in the authors' setup, and the 5 best are improved in parallel for 100 policy steps. The metric is the initial makespan (best of 20) minus the best makespan found (best of the 5 runs), in time units; higher is better. We run three seeds; all selectors and commit depths of a seed share the initial schedules and the random seed. A descent step evaluates the first 50 moves of the environment's list of feasible moves exactly and applies the best of them; at cd $\in \{1, 2, 3\}$ this move is applied even if it does not improve the makespan, so, unlike in the other methods, a descent step can worsen the schedule. For LO a step is applied only if it strictly improves the makespan, and the descent stops otherwise. The tabu restriction used in Appendix H.8 forbids moving the operation that the selector moved back to the machine it came from.

The published policy is almost uniform. For the released 10×5, 15×10, and 20×10 checkpoints, the policy assigns almost the same probability to every feasible move: in the authors' own evaluation code, the maximum and the minimum of $p \cdot n$ over the n feasible moves are within 2×10^{-4} of 1. Over 400 states recorded by instrumenting that code, the maximum of $p \cdot n$ is 1.0002, the minimum 0.99995, and the entropy divided by $\log n$ is 1.0000, also with batch normalization in evaluation mode. The cause is saturation: the four stacked linear–tanh–linear blocks of the policy shrink the spread of the node embeddings from 0.13 to 0.004, 2×10^{-5}, and 2×10^{-7}. The 20×5 checkpoint is not uniform (maximum of $p \cdot n$: 13.6; entropy divided by $\log n$: 0.80). The weights we load are bit-identical to the released file (112 tensors); the 10×5 file uses older module names, which we rename. With cd=0, one training update of our loop gives the same losses, parameters, and solutions as the authors' training code.

How MIStar selects a move. At each step MIStar samples up to 50 distinct candidate moves from its policy (60 draws; duplicates are dropped), evaluates the makespan of each exactly, and applies the best. With a uniform policy this is the best of 50 uniformly sampled moves. The single-move random baseline (R_1) applies a uniformly random move without evaluation. We therefore add a matched random baseline (R_{50}) that samples 50 distinct candidates uniformly with the same sampler, evaluates them exactly, and applies the best. With a shared CPU random-number stream, R_{50} reproduces the published policy's trajectories exactly. In the sweep below the two arms draw from different streams (GPU and CPU), so their results differ by sampling noise. A first run of this sweep passed the state before descent to the policy at cd$\geq$1 (the descent itself was applied correctly); since the policy is almost uniform, this changed its move probabilities by at most 1.7×10^{-4} (relative), and the sweep below passes the state after descent. The learned-minus-random differences of that first run ($+20.5$ to $+4.1$ over cd $\in \{0, 1, 2, 3\}$) compared the published policy with R_1.

Commit-depth sweep. Table 43 gives the results. Against R_1, the published policy is ahead by 19.4 at cd=0, and the difference falls with cd (Page's trend test over cd $\in \{0, 1, 2, 3\}$ with the 10 instances as blocks: p=8.2×10^{-8}). Against R_{50}, the difference is $+0.3$ at cd=0 (p=0.79, W/T/L 13/4/13) and $+1.6$ to $+2.9$ at cd$\geq$1 (p=0.02 to 0.07), with no trend over cd (p=0.85). Almost all of the published policy's advantage over the single-move baseline comes from evaluating 50 candidates rather than applying one: R_{50} is ahead of R_1 by 19.1 at cd=0. The p-values of the small positive M $-$ R_{50} differences at cd$\geq$1 are not corrected for multiple comparisons; in the 88-instance evaluation of Appendix H.8, R_{50} with tabu-restricted descent at cd=1 is 0.54 pp ahead of the published policy with the same descent (p=0.009). Descent undoes the published policy's move in 80% of steps at cd=1, 38% at cd=2, 44% at cd=3, and 34% at LO (R_{50}: 81%, 39%, 44%, 38%; R_1: 26%, 20%, 17%, 8%).

Table 43: MIStar, commit-depth sweep (10 SD2 10×5+mix instances, 3 seeds, 30 pairs, 100 steps): mean makespan improvement over the initial schedule (higher is better). M: published policy. R_1: one uniformly random move per step, without evaluation. R_{50}: the best of 50 uniformly sampled candidates, evaluated exactly. Differences with paired t-test p over (instance, seed) pairs.

cd	R_1	R_{50}	M	M $-$ R_1	M $-$ R_{50}	R_{50} $-$ R_1
0	2.43	21.57	21.87	+19.43 ($<10^{-9}$)	+0.30 (0.79)	+19.13 ($<10^{-9}$)
1	8.50	18.63	21.50	+13.00 ($<10^{-8}$)	+2.87 (0.044)	+10.13 ($<10^{-7}$)
2	13.87	19.50	22.43	+8.57 ($<10^{-5}$)	+2.93 (0.024)	+5.63 (0.001)
3	14.70	18.27	20.40	+5.70 ($<10^{-3}$)	+2.13 (0.068)	+3.57 (0.025)
LO	14.27	19.77	21.40	+7.13 ($<10^{-4}$)	+1.63 (0.063)	+5.50 (0.003)

Each selector at its best setting. The published policy is best at cd=2 (22.43), R_1 at cd=3 (14.70), and R_{50} at cd=0 (21.57). With each selector at its best setting, the published policy is ahead of R_1 by 7.73 (p=2.8×10^{-5}, W/T/L 26/1/3), against 19.43 at cd=0, and not significantly different from R_{50} ($+0.87$, p=0.54, W/T/L 16/4/10).

H FINE-TUNING WITH DESCENT IN THE TRAINING LOOP

H.1 PROTOCOL

Training. We start from the released checkpoint and use the authors' training code and algorithm. The only change is in the environment step: the policy's move is applied, the descent runs, and the reward and the next state are computed on the solution after the descent. The descent is the one with which the published policy is best in the commit-depth study (DACT: edge tabu, LO; NeuOpt on CVRP-100: plain, $\mathrm{cd}{=}1$; ECO-DQN: tabu, LO). A control run starts from the same checkpoint and uses the same training data, seed, learning rate, and number of updates, with $\mathrm{cd}{=}0$. Table 44 lists the settings. Where the authors' settings do not specify fine-tuning, we keep their values and change only what is listed.

Checkpoints. Before any test result was available, we fixed the rule to evaluate two checkpoints per run: the final one and the one with the best validation score, measured with the descent used in that run. Validation instances are disjoint from the test instances. In DACT and NeuOpt the published policy is validated but not eligible as the best-validation checkpoint; in ECO-DQN it is eligible but was never selected.

Equal wall-clock time. All arms of one evaluation run in one process on one GPU, and the wall-clock time of every step is recorded, so each arm has an anytime curve over time. The time budget is the time the fine-tuned policy needs for its T policy steps. Each baseline is read at the last step it completes within this budget. Baselines that are cheaper per step are run for more steps: $3T$ for the published policy and the control at $\mathrm{cd}{=}0$ and for random selection in DACT; $3T$ at $\mathrm{cd}{=}0$ and $2T$ for random selection in NeuOpt; episodes of $3|V|$ and $4|V|$ steps for the published policy at $\mathrm{cd}{=}0$ and of $8|V|$ steps for random selection in ECO-DQN. When a baseline's run ends before the budget (marked †), we use its final value, which favors the fine-tuned policy. ECO-DQN's observation contains the fraction of the episode that has elapsed, so its episode length is fixed in advance and we report both the $3|V|$ and the $4|V|$ run (Table 3 uses $4|V|$). The four baselines are the published policy at $\mathrm{cd}{=}0$, the published policy with its best descent setting, random selection with its best descent setting, and the control with its best descent setting; we also report the control at $\mathrm{cd}{=}0$.

Table 44: Fine-tuning settings. Wall time: training time per run (fine-tuned / control). GPUs: RTX 4090 (24 GB) for the first seed, RTX PRO 6000 (96 GB) for the second seed of DACT and ECO-DQN. MIStar and L2S: Table 45.

	DACT	NeuOpt	ECO-DQN
Algorithm	PPO (authors' code)	PPO (authors' code)	double DQN (authors' code)
Training descent	edge tabu, LO (cap 50)	plain, $\mathrm{cd}{=}1$	tabu, LO (cap $\lvert V\rvert$)
Problems	TSP-50, TSP-100	CVRP-100	Max-Cut ER40, ER60, ER200; BA40, BA60
Learning rate	actor $4.16{\times}10^{-5}$, critic $1.25{\times}10^{-5}$, $\times 0.985$ per epoch	10^{-5} (both), $\times 0.985$ per epoch	10^{-4}, constant
Data and updates	TSP-50: 21 epochs of 20 batches of 600; TSP-100: 11 epochs of 20 batches of 400; 400 warm-up steps and 200 rollout steps per batch	16 epochs of 20 batches of 300; 250 rollout steps per batch	ER40 / ER60 / ER200: 9.95 / 10.4 / 8.45 million steps; BA40, BA60: 4 million steps; one update (batch 64) every 32 steps; $\epsilon{=}0.05$
Validation	256 instances (from index 5000 of the authors' test file), 500 steps, gap to LKH-3, every epoch	128 instances (from index 500), 500 steps, gap to HGS, every epoch	50 graphs of the authors' training-time test set, 4 starts, mean best cut, every 50,000 steps (200,000 for ER200)
Test	first 128 instances, 5 runs, $T{=}500$	first 500 instances, 1 run, $T{=}500$	100 evaluation graphs, 50 starts (10 for ER200), $T{=}2\lvert V\rvert$; ER500: 50 graphs, 10 starts
Wall time, seed 1	TSP-50 6.9 / 5.7 h; TSP-100 6.4 / 5.5 h	8.6 / 8.6 h	ER40 6.5 / 6.4 h; ER60 7.0 / 6.8 h; ER200 7.5 / 7.0 h (GPUs shared by runs); BA40 2.5 / 2.4 h; BA60 2.6 / 2.5 h
Wall time, seed 2	TSP-50 5.3 / 4.8 h; TSP-100 2.9 / 2.6 h	8.7 / 8.4 h (RTX 4090)	ER40 6.6 / 6.6 h; ER60 6.8 / 6.8 h; ER200 4.8 / 4.6 h; BA40 2.5 / 2.4 h, BA60 2.6 / 2.5 h (RTX 4090)

Changes during the runs. In DACT the TSP-100 runs use batches of 400 instead of 600, with the same 20 batches per epoch, because batches of 600 did not fit into the 24 GB of the RTX 4090. A first launch of the DACT runs with descent cached the distance matrix under the memory address

Table 45: Fine-tuning settings for MIStar and L2S (RTX 4090 for both seeds). Rows as in Table 44.

	MIStar	L2S
Algorithm	PPO (authors' code; clip 0.2, 3 epochs per update, entropy coefficient 0.01)	n-step REINFORCE (authors' code)
Training descent	tabu, cd=1 (plain descent at cd=1 ends at the solution before the move in 53–66% of training steps)	tabu, LO (cap 50)
Problems	FJSP SD2 10×5+mix	JSSP, generated 10×10 instances (durations in $[1, 99]$), from the released 10×10 checkpoint
Learning rate	5×10^{-5}, constant	5×10^{-5} (Adam, new optimizer state)
Data and updates	80 episodes, each of 20 generated instances and 100 steps with 50 candidates per step; an update every 10 steps	480 batches of 64 instances, 500 steps per episode, an update every 10 steps (30,720 episodes, 24% of the authors' training)
Validation	32 instances of the authors' validation set, 100 steps, incumbent makespan, every 10 episodes	100 instances of the authors' 10×10 validation set, 500 steps, incumbent makespan, every 10 batches
Test	first 88 of the 100 SD2 10×5+mix test instances, T=100, 5 runs from the 5 best of 20 DANIEL schedules; gap to OR-Tools CP-SAT (15 s per instance)	Taillard ta01–10 (15×15), 3 seeds, T=5000
Wall time	seed 1: 3.3 / 1.6 h; seed 2: 3.2 / 1.6 h	6.1 / 6.4 h per seed (one GPU and eight CPU processes per run; the controls continued to batch 670)
Checkpoints	best-validation: episode 10 for both fine-tuned runs and the first control, episode 0 (the published policy) for the second control; final: episode 80	best-validation among batches ≥10: batch 200 (seed 1) and 480 (seed 2, the final one) for the fine-tuned runs, 410 and 310 for the controls; final: batch 480 for all runs

of the coordinate tensor, which the GPU allocator reuses, so training batches after the first used the distances of the previous batch; we fixed the cache and restarted these runs from the released checkpoints, and all reported runs are the restarted ones (the controls and all evaluations did not use the cache). An earlier second-seed NeuOpt run on the RTX PRO 6000 used TF32 matrix products to make the PPO updates fast enough on that GPU; we do not use it and re-ran the second seed in full precision on the RTX 4090, the hardware of the first seed. In MIStar a first pair of runs with learning rate 5×10^{-4} collapsed within ten episodes (normalized entropy of the fine-tuned policy 0.03 in episodes 6–10; validation makespan of the control from 370.5 to 391.3) and was stopped; all reported runs use 5×10^{-5}. Before any evaluation we stopped all MIStar runs after episode 80 of the planned 140. The MIStar evaluation covers the first 88 of the 100 test instances, the number completed before our deadline.

H.2 Comparison before fine-tuning

Before fine-tuning, we compare the published policy with random selection, each at its best descent setting (the pre-fine-tuning comparison of Section 5). Table 46 gives this comparison with the same number of steps and at equal wall-clock time. The time budget is either the published policy's time for T steps or the fine-tuned policy's time for T steps in the evaluation; Table 3 uses the former, averaged over the two evaluations. In ECO-DQN and NeuOpt the published policy is ahead in all three columns; in DACT random selection is tied with it for the same number of steps and ahead at equal time, because its steps are cheaper; in MIStar random selection with 50 evaluated candidates per step is slightly ahead in all three columns, never significantly.

In DACT the differences with the same number of steps are not significant (p=0.10 and 0.11 on the first-seed hardware), and those at equal time are (p=0.003 and $p<10^{-12}$). On CVRP-100, random selection with plain or tabu-restricted descent rarely reaches a capacity-feasible solution, so the NeuOpt comparison is dominated by feasibility; with the repair descent of the commit-depth study, random selection reaches 9.76% and the published policy is still ahead by 7.9 pp (Appendix F.3). In L2S, a random step with tabu-restricted descent to a local optimum takes 10.5 ms against 20.5 ms for a learned step in the commit-depth study (Appendix F.6), so the difference with the same number of steps understates random selection's advantage at equal time: at the published policy's time for 5000 steps with this descent, random selection with the same descent reaches 1.94% against 4.17% (p=3.4×10^{-8}; the published policy is better on 1 of the 30 pairs). We recorded the L2S prediction before the first fine-tuning run. In MIStar the published policy is best at equal steps with tabu-restricted descent to a local optimum (7.26%) and at the fine-tuned policy's time at cd=0 with more steps (7.32–7.33%); R_{50} is best with tabu-restricted descent at cd=1 (7.07% at 100 steps). The MIStar differences have p=0.38 (equal steps), 0.10 (published policy's time; R_{50}'s runs end earlier, so their final values are used), and 0.25 and 0.26 (fine-tuned policy's time).

Table 46: Comparison before fine-tuning: published policy minus random selection, each at its best descent setting (positive = published better; units as in Table 2). Two values: evaluation on the first-seed (RTX 4090) / second-seed hardware. Same number of steps for ECO-DQN: from the commit-depth study ($2|V|$ steps). BA40, BA60: both evaluations on an RTX 4090. NeuOpt: two evaluations on an RTX 4090 (run with the first and the second seed's checkpoints); the runs of the published policy and of random selection are identical in the two evaluations; random selection's best setting is tabu, cd=10; its 500 steps take 56.2 / 55.2 s, those of the published policy with its best descent (tabu, cd=2) 56.6 / 55.8 s; at the fine-tuned policy's time (53.8 / 53.1 s, best-validation checkpoints) the published policy is best with tabu at cd=1. MIStar: one evaluation (RTX 4090); optimality gap to OR-Tools in pp; random selection evaluates 50 candidates per step (R_{50}); the two fine-tuned times are those of the two seeds' best-validation checkpoints. L2S: one evaluation (one CPU thread per run); the fine-tuned times are those of the two seeds' best-validation checkpoints.

Method	Problem	same no. of steps	equal time (published)	equal time (fine-tuned)
ECO-DQN	ER40	+1.85	+1.43 / +1.53	+1.39 / +1.47
	ER60	+1.61	+1.36 / +1.35	+1.21 / +1.20
	ER200	+2.46	+1.98 / +2.24	+1.93 / +2.13
	BA40	+1.34	+1.41 / +1.40	+1.19 / +1.26
	BA60	+1.07	+1.10 / +1.11	+0.76 / +0.84
NeuOpt	CVRP-100	+127.5	+126.0	+129.7 / +129.6
DACT	TSP-50	−0.027 / −0.021	−0.047 / −0.044	−0.052 / −0.046
	TSP-100	−0.069 / −0.035	−0.344 / −0.361	−0.422 / −0.394
L2S	JSSP ta 15×15	−1.65	−2.23	−2.22 / −2.26
MIStar	FJSP SD2 10×5	−0.18	−0.34	−0.25 / −0.25

H.3 Results: ECO-DQN

Tables 47 and 48 give the best-validation and the final checkpoints. With the best-validation checkpoint the fine-tuned policy beats every baseline on ER40, ER60, and ER200 for both seeds (all $p \leq 0.02$). With the final checkpoint it still beats every baseline on ER60 and ER200, but on ER40 it is tied with the published policy with descent (-0.02 and -0.01 pp, $p \geq 0.45$) and only slightly ahead of it at cd=0 ($+0.03$ to $+0.08$ pp, $p=0.04$ to 0.26). The control, fine-tuned without descent, degrades late in training: with descent, its ER60 ratio is 99.55% and 99.57% at the best-validation checkpoints and 97.70% and 98.38% at the final ones (with the same number of steps). On ER40 and ER60 the best of 50 starts is 100% for the fine-tuned policy and for the published policy, random selection, and the control with descent, so differences there appear only in the mean ratio.

BA graphs: outcome predicted before the runs. Before running them, we recorded the prediction that fine-tuning ECO-DQN on Barabási–Albert graphs with 40 and 60 vertices would beat all four baselines, because the pre-fine-tuning comparison is positive there ($+1.34$ and $+1.07$ pp with the same number of steps; Table 46). We fine-tuned the released BA40 and BA60 networks for 4 million steps with the ER protocol (Table 44; two runs per GPU, two CPU threads per run). The prediction holds: with both checkpoints and both seeds the fine-tuned policy beats every baseline (Tables 47 and 48), by 0.28 to 2.56 pp with the best-validation checkpoints and 0.42 to 2.35 pp with the final ones; the largest p is 1.4×10^{-8}. The best of 50 starts is 100% for all these arms. As on ER graphs, the fine-tuned policy selects more flips that lower the cut than the published policy (0.29–0.32 against 0.22 on BA40, 0.36–0.43 against 0.11 on BA60; best-validation checkpoints, tabu-restricted descent to a local optimum), and descent improves the cut more after them (0.94–1.17 against 0.57, and 1.31–1.46 against 0.54 per step).

What changes in the fine-tuned policy. Table 49 gives, for the arms with tabu-restricted descent to a local optimum, the mean cut gained by the descent after each selected flip and the share of selected flips that lower the cut. The fine-tuned policy chooses flips that lower the cut more often than the published policy and far less often than random selection, and descent improves the cut more after them. The control's values are below the published policy's on ER40 and within 0.04 of them on ER60 and ER200.

Table 47: ECO-DQN fine-tuning, best-validation checkpoints. First row of each block: the fine-tuned policy's mean approximation ratio and its time per graph. Other rows: fine-tuned minus baseline at equal wall-clock time (pp; positive = fine-tuned better), paired t-test p, and W/T/L over graphs. ER500*: 50 graphs with 500 vertices, networks fine-tuned on ER200 (Appendix H.4). BA40, BA60: Barabási–Albert graphs with 40 and 60 vertices (100 graphs each; both seeds evaluated on an RTX 4090); the prediction that fine-tuning beats all four baselines was recorded before these runs. Published + descent: tabu-restricted descent to a local optimum; on BA60 the published policy is slightly better with tabu at cd=1, against which the differences are +1.64 and +1.69 (the values in Table 3). Against the better of these descent settings, the fine-tuned policy removes 61/45% (ER40), 79/82% (ER60), 59/52% (ER200), 66/65% (BA40), and 85/87% (BA60) of the published policy's remaining gap to the best known cut (seed 1 / seed 2).

		seed 1			seed 2		
Graphs	Arm	diff.	p	W/T/L	diff.	p	W/T/L
ER40	fine-tuned (ratio, %)	99.93	(0.60 s/graph)		99.91	(0.19 s/graph)	
	published, cd=0, 3|V|	+0.16	$<10^{-6}$	47/47/6	+0.13	$<10^{-5}$	44/46/10
	published, cd=0, 4|V|	+0.20	$<10^{-5}$	50/47/3	+0.17	$<10^{-4}$	47/43/10
	published + descent	+0.10	0.001	32/59/9	+0.08	0.02	29/56/15
	random + descent	+1.49	$<10^{-29}$	99/1/0	+1.54	$<10^{-30}$	100/0/0
	control, cd=0	+0.26	$<10^{-9}$	63/35/2	+0.31	$<10^{-12}$	75/23/2
	control + descent	+0.12	$<10^{-5}$	47/45/8	+0.11	$<10^{-6}$	53/37/10
ER60	fine-tuned (ratio, %)	99.84	(0.96 s/graph)		99.86	(0.31 s/graph)	
	published, cd=0, 3|V|	+0.52	$<10^{-20}$	92/6/2	+0.56	$<10^{-21}$	93/6/1
	published, cd=0, 4|V|	+0.62	$<10^{-24}$	96/4/0	+0.67	$<10^{-25}$	96/4/0
	published + descent	+0.61	$<10^{-20}$	94/5/1	+0.63	$<10^{-22}$	95/5/0
	random + descent	+1.82	$<10^{-41}$	99/1/0	+1.83	$<10^{-41}$	99/1/0
	control, cd=0	+0.57	$<10^{-22}$	92/8/0	+0.55	$<10^{-22}$	95/5/0
	control + descent	+0.29	$<10^{-14}$	81/13/6	+0.29	$<10^{-16}$	82/16/2
ER200	fine-tuned (ratio, %)	99.35	(2.20 s/graph)		99.22	(0.72 s/graph)	
	published, cd=0, 3|V|	+0.88	$<10^{-26}$	95/0/5	+0.60	$<10^{-15}$	84/2/14
	published, cd=0, 4|V|	+0.95	$<10^{-27}$	96/0/4	+0.64	$<10^{-18}$	86/6/8
	published + descent	+0.96	$<10^{-31}$	95/0/5	+0.83	$<10^{-24}$	95/0/5
	random + descent	+2.89	$<10^{-53}$	100/0/0	+2.96	$<10^{-51}$	100/0/0
	control, cd=0	+0.86	$<10^{-26}$	96/0/4	+0.70	$<10^{-18}$	89/2/9
	control + descent	+0.71	$<10^{-25}$	96/0/4	+0.44	$<10^{-12}$	81/2/17
ER500*	fine-tuned (ratio, %)	97.98	(23.10 s/graph)		98.15	(11.96 s/graph)	
	published, cd=0, 2|V|	+0.39	$<10^{-5}$	43/1/6	+0.56	$<10^{-8}$	42/0/8
	published + descent	+0.21	0.003	32/0/18	+0.38	$<10^{-5}$	39/0/11
	random + descent	+2.73	$<10^{-31}$	50/0/0	+2.85	$<10^{-33}$	50/0/0
	control, cd=0	−0.16	0.008	17/0/33	+0.07	0.30	28/1/21
	control + descent	−0.20	0.008	16/0/34	+0.00	0.99	23/1/26
BA40	fine-tuned (ratio, %)	99.70	(0.53 s/graph)		99.68	(0.51 s/graph)	
	published, cd=0, 3|V|	+0.45	$<10^{-9}$	73/12/15	+0.47	$<10^{-10}$	74/13/13
	published, cd=0, 4|V|	+0.58	$<10^{-12}$	80/12/8	+0.62	$<10^{-13}$	81/12/7
	published + descent	+0.58	$<10^{-11}$	75/11/14	+0.59	$<10^{-12}$	79/12/9
	random + descent	+1.76	$<10^{-29}$	100/0/0	+1.85	$<10^{-30}$	100/0/0
	control, cd=0	+0.61	$<10^{-15}$	83/9/8	+0.50	$<10^{-13}$	80/15/5
	control + descent	+0.34	$<10^{-7}$	65/15/20	+0.28	$<10^{-8}$	62/17/21
BA60	fine-tuned (ratio, %)	99.72	(0.89 s/graph)		99.76	(0.87 s/graph)	
	published, cd=0, 3|V|	+1.31	$<10^{-24}$	98/2/0	+1.32	$<10^{-27}$	97/3/0
	published, cd=0, 4|V|	+1.46	$<10^{-22}$	96/4/0	+1.49	$<10^{-25}$	99/1/0
	published + descent	+1.67	$<10^{-25}$	99/1/0	+1.73	$<10^{-26}$	99/1/0
	random + descent	+2.43	$<10^{-38}$	100/0/0	+2.56	$<10^{-39}$	100/0/0
	control, cd=0	+1.57	$<10^{-28}$	99/1/0	+1.43	$<10^{-28}$	98/2/0
	control + descent	+1.39	$<10^{-27}$	98/2/0	+1.14	$<10^{-23}$	97/3/0

H.4 Generalization to larger graphs (ECO-DQN)

We evaluate the networks fine-tuned on ER200 on 50 ER graphs with 500 vertices (10 starts, 1000 steps), a size not used in training. The fine-tuned policy beats the published policy at cd=0 (+0.39 to +0.60 pp over both seeds and both checkpoints), the published policy with descent (+0.21 to +0.42 pp), and random selection with descent (+2.73 to +2.96 pp), all with $p \leq 0.003$. The comparison with the control is mixed: with the first seed's best-validation checkpoint the control is ahead (0.16

Table 48: ECO-DQN fine-tuning, final checkpoints. Columns as in Table 47.

		seed 1			seed 2		
Graphs	Arm	diff.	p	W/T/L	diff.	p	W/T/L
ER40	fine-tuned (ratio, %)	99.81	(0.56 s/graph)		99.82	(0.18 s/graph)	
	published, cd=0, $3\|V\|$	+0.03	0.26	35/37/28	+0.04	0.15	36/43/21
	published, cd=0, $4\|V\|$	+0.06	0.06	32/41/27	+0.08	0.04	34/45/21
	published + descent	−0.02	0.45	20/47/33	−0.01	0.76	23/54/23
	random + descent	+1.49	$<10^{-29}$	100/0/0	+1.52	$<10^{-31}$	99/1/0
	control, cd=0	+1.56	$<10^{-22}$	97/3/0	+0.99	$<10^{-15}$	95/5/0
	control + descent	+1.13	$<10^{-16}$	92/6/2	+0.73	$<10^{-12}$	82/16/2
ER60	fine-tuned (ratio, %)	99.76	(0.90 s/graph)		99.74	(0.30 s/graph)	
	published, cd=0, $3\|V\|$	+0.43	$<10^{-16}$	87/8/5	+0.38	$<10^{-15}$	85/6/9
	published, cd=0, $4\|V\|$	+0.54	$<10^{-21}$	90/6/4	+0.49	$<10^{-20}$	93/4/3
	published + descent	+0.53	$<10^{-19}$	93/6/1	+0.51	$<10^{-18}$	92/4/4
	random + descent	+1.87	$<10^{-41}$	99/1/0	+1.81	$<10^{-40}$	99/1/0
	control, cd=0	+2.56	$<10^{-42}$	100/0/0	+1.90	$<10^{-37}$	100/0/0
	control + descent	+2.05	$<10^{-37}$	99/1/0	+1.36	$<10^{-31}$	97/3/0
ER200	fine-tuned (ratio, %)	99.22	(2.03 s/graph)		99.33	(0.78 s/graph)	
	published, cd=0, $3\|V\|$	+0.65	$<10^{-20}$	92/1/7	+0.71	$<10^{-24}$	96/1/3
	published, cd=0, $4\|V\|$	+0.71	$<10^{-23}$	93/1/6	+0.74	$<10^{-24}$	93/0/7
	published + descent	+0.83	$<10^{-24}$	98/0/2	+0.94	$<10^{-29}$	98/0/2
	random + descent	+2.83	$<10^{-51}$	100/0/0	+3.00	$<10^{-57}$	100/0/0
	control, cd=0	+1.27	$<10^{-39}$	100/0/0	+1.22	$<10^{-37}$	100/0/0
	control + descent	+1.02	$<10^{-27}$	98/1/1	+1.03	$<10^{-29}$	97/2/1
ER500*	fine-tuned (ratio, %)	98.19	(21.73 s/graph)		98.11	(11.65 s/graph)	
	published, cd=0, $2\|V\|$	+0.60	$<10^{-11}$	44/2/4	+0.52	$<10^{-6}$	41/0/9
	published + descent	+0.42	$<10^{-7}$	37/0/13	+0.34	$<10^{-4}$	39/0/11
	random + descent	+2.96	$<10^{-33}$	50/0/0	+2.83	$<10^{-31}$	50/0/0
	control, cd=0	+0.16	0.06	34/0/16	+0.06	0.43	31/0/19
	control + descent	+0.16	0.04	30/1/19	+0.08	0.23	31/0/19
BA40	fine-tuned (ratio, %)	99.75	(0.53 s/graph)		99.66	(0.51 s/graph)	
	published, cd=0, $3\|V\|$	+0.49	$<10^{-13}$	77/14/9	+0.47	$<10^{-10}$	76/9/15
	published, cd=0, $4\|V\|$	+0.65	$<10^{-17}$	89/10/1	+0.59	$<10^{-13}$	83/7/10
	published + descent	+0.62	$<10^{-15}$	81/13/6	+0.56	$<10^{-11}$	80/12/8
	random + descent	+1.79	$<10^{-32}$	100/0/0	+1.82	$<10^{-29}$	100/0/0
	control, cd=0	+0.72	$<10^{-17}$	86/12/2	+0.58	$<10^{-14}$	79/11/10
	control + descent	+0.45	$<10^{-12}$	76/17/7	+0.42	$<10^{-10}$	68/12/20
BA60	fine-tuned (ratio, %)	99.35	(0.81 s/graph)		99.53	(0.84 s/graph)	
	published, cd=0, $3\|V\|$	+1.09	$<10^{-18}$	94/2/4	+1.18	$<10^{-24}$	96/2/2
	published, cd=0, $4\|V\|$	+1.26	$<10^{-20}$	96/1/3	+1.34	$<10^{-23}$	95/1/4
	published + descent	+1.35	$<10^{-19}$	97/2/1	+1.48	$<10^{-23}$	99/1/0
	random + descent	+2.25	$<10^{-37}$	100/0/0	+2.35	$<10^{-38}$	100/0/0
	control, cd=0	+2.24	$<10^{-35}$	100/0/0	+2.19	$<10^{-36}$	100/0/0
	control + descent	+1.99	$<10^{-33}$	100/0/0	+1.92	$<10^{-30}$	98/0/2

Table 49: ECO-DQN, tabu-restricted descent to a local optimum (best-validation checkpoints; seed 1 / seed 2). Gain: mean cut gained by the descent per policy step. Worse: share of selected flips that lower the cut.

	ER40		ER60		ER200	
Arm	gain	worse	gain	worse	gain	worse
published	1.16	0.49	0.85	0.24	1.38	0.31
control	0.72 / 0.67	0.26 / 0.25	0.81 / 0.85	0.26 / 0.28	1.37 / 1.42	0.29 / 0.33
fine-tuned	1.33 / 1.52	0.42 / 0.47	1.48 / 1.58	0.46 / 0.49	2.32 / 2.20	0.54 / 0.51
random	3.03	0.90	3.98	0.94	7.89	0.98

pp at cd=0 and 0.20 pp with descent, p=0.008), with the second seed they are tied ($p\geq$0.30), and with the final checkpoints the fine-tuned policy is ahead by 0.06 to 0.16 pp (p=0.04 to 0.43).

GSET. Before running it, we recorded the prediction that on the GSET graphs G1–G10 (800 vertices, best known cuts from the ECO-DQN repository) the ER200 policies fine-tuned with descent beat the published ER200 policy in its original loop, the published policy with its best descent setting,

and random selection with descent. We evaluate the best-validation checkpoints of both seeds with 10 starts per graph and tabu-restricted descent to a local optimum, and read every arm at the fine-tuned policy's wall-clock time (1600 steps). In mean approximation ratio, paired over graphs and starts, the fine-tuned policy beats the published policy in its original loop by $+0.38$ and $+0.54$ pp (p=0.004 and $p<10^{-5}$; with $4|V|$ steps, $+0.24$ and $+0.36$ pp, p=0.048 and 0.001) and random selection with descent by $+2.1$ and $+2.0$ pp ($p<10^{-11}$). Against the published policy with its best descent setting (tabu, cd=1 or LO) it is ahead by only $+0.12$ to $+0.17$ pp ($p\geq 0.13$), and against the control with descent by $+0.06$ and $+0.05$ pp ($p\geq 0.54$), so this part of the prediction fails. The best cut over the 10 starts, the measure of the ECO-DQN paper, averages 99.35% and 99.50% of the best known cut for the fine-tuned policy, against 99.04% for the original loop (99.30% with $4|V|$ steps) and 99.32% and 99.40% for the published policy with tabu at cd=1. Some baseline runs finish their steps before the fine-tuned policy's time and keep their final value (at most 9 of 10 graphs for the original loop with $2|V|$ steps, fewer for the other arms).

H.5 Training ECO-DQN from scratch with descent in the training loop

Fine-tuning starts from a network trained without descent. We therefore also train ECO-DQN from scratch with tabu-restricted descent to a local optimum in the training loop, with the authors' training schedule: 2.5 million steps on ER40 and 5 million on ER60; ϵ decreasing linearly from 1 to 0.05 over the first 150,000 (ER40) and 300,000 (ER60) steps; uniformly random actions until the replay buffer holds 500 transitions; a buffer of 5000 transitions; the target network updated every 1000 steps; the other settings as in Table 44. The control is trained from scratch in the same way with cd=0. Each pair of runs (with and without descent) shared one A100 GPU; we ran two seeds. A run took 1.6 h on ER40 and 3.2–3.3 h on ER60. We evaluate the best-validation and the final checkpoints as in the fine-tuning study, with the released network as the published policy; the evaluation reproduces the episodes we stored earlier for the published policy (4999 or 5000 of 5000 per arm).

Table 50 gives the results. On ER60 the network trained with descent beats every baseline with both checkpoints and both seeds, including the published policy with its best descent setting ($+0.52$ to $+0.65$ pp, $p<10^{-14}$). On ER40 it is tied with the published policy with descent (-0.03 to $+0.01$ pp, $p\geq 0.41$) and 0.02 to 0.10 pp ahead of the published policy at cd=0 (p=0.006 to 0.36). On both sizes it beats the control trained without descent, by 0.05 to 0.62 pp ($p\leq 0.048$), and random selection with descent, by 1.57 to 1.91 pp. With tabu-restricted descent, 0.33–0.37 (ER40) and 0.44–0.57 (ER60) of its selected flips lower the cut, against 0.49 and 0.24 for the published policy, and descent improves the cut by 0.84–1.19 and 1.53–1.90 per step, against 1.16 and 0.85.

H.6 Results: NeuOpt

Table 51 gives both seeds. With the best-validation checkpoints the fine-tuned policy beats every baseline for both seeds (by at least 0.34 and 0.19 pp, $p<10^{-3}$). With the final checkpoint it beats every baseline for the first seed. For the second seed it beats the published policy at cd=0, random selection, and the control, but it is ahead of the published policy with its best descent by only 0.12 pp (p=0.09, not significant). The published policy's best descent (tabu-restricted, cd=2) and the control's best descent were chosen on the test instances, which favors these baselines. Random selection with plain descent at cd=1 has a gap of 229% at the time budget. Its best setting in this evaluation (tabu-restricted, cd=10) reaches 129.1% after 500 steps, which take longer than the fine-tuned policy's 500 steps (56.2 against 53.8 s for the first seed's best-validation checkpoint, 55.2 against 53.1 s for the second seed's), so the fine-tuned policy is ahead of it by more than 127 pp with every checkpoint. Plain descent undoes 17% of the published policy's non-null moves, 11–19% of the control's (15% and 11% for the first seed's best-validation and final checkpoints, 19% and 15% for the second seed's), and 7–8% of the fine-tuned policy's (7% for both checkpoints of the first seed, 7% and 8% for the second seed's).

H.7 Results: DACT

Tables 52 and 53 give the best-validation and the final checkpoints. Random selection with reversal tabu descent to a local optimum beats the fine-tuned policy in every case (0.09 to 1.01 pp, all $p<10^{-5}$). On TSP-50 the fine-tuned policy is also behind the published policy with descent and the control with descent, with both checkpoints and both seeds (except the first seed's final checkpoint against the

Table 50: ECO-DQN trained from scratch with tabu-restricted descent to a local optimum. First row of each block: its mean approximation ratio and time per graph. Other rows: network trained with descent minus baseline at equal wall-clock time (pp; positive = trained with descent better), paired t-test p, and W/T/L over 100 graphs. Published: the released network. Control: trained from scratch with cd=0.

		seed 1			seed 2		
Graphs	Arm	diff.	p	W/T/L	diff.	p	W/T/L
ER40, val.-best	trained with descent (ratio, %)	99.78	(0.53 s/graph)		99.81	(0.52 s/graph)	
	published, cd=0, $3\|V\|$	+0.02	0.36	34/37/29	+0.05	0.05	38/35/27
	published, cd=0, $4\|V\|$	+0.08	0.04	38/36/26	+0.09	0.02	36/36/28
	published + descent	−0.03	0.41	21/44/35	−0.00	0.93	22/38/40
	random + descent	+1.57	$<10^{-31}$	100/0/0	+1.68	$<10^{-31}$	100/0/0
	control, cd=0	+0.30	$<10^{-11}$	75/18/7	+0.39	$<10^{-11}$	79/16/5
	control + descent	+0.05	0.048	38/40/22	+0.12	$<10^{-4}$	50/32/18
ER40, final	trained with descent (ratio, %)	99.83	(0.54 s/graph)		99.82	(0.53 s/graph)	
	published, cd=0, $3\|V\|$	+0.06	0.03	40/34/26	+0.06	0.02	38/35/27
	published, cd=0, $4\|V\|$	+0.10	0.01	37/39/24	+0.10	0.006	34/39/27
	published + descent	+0.01	0.75	20/44/36	−0.00	0.88	17/48/35
	random + descent	+1.61	$<10^{-31}$	100/0/0	+1.61	$<10^{-30}$	99/1/0
	control, cd=0	+0.47	$<10^{-13}$	76/17/7	+0.40	$<10^{-11}$	75/19/6
	control + descent	+0.25	$<10^{-6}$	55/32/13	+0.20	$<10^{-6}$	56/30/14
ER60, val.-best	trained with descent (ratio, %)	99.69	(0.94 s/graph)		99.86	(0.96 s/graph)	
	published, cd=0, $3\|V\|$	+0.39	$<10^{-11}$	83/7/10	+0.50	$<10^{-20}$	93/7/0
	published, cd=0, $4\|V\|$	+0.49	$<10^{-15}$	86/7/7	+0.62	$<10^{-16}$	95/5/0
	published + descent	+0.53	$<10^{-14}$	85/7/8	+0.65	$<10^{-21}$	95/5/0
	random + descent	+1.83	$<10^{-39}$	100/0/0	+1.89	$<10^{-39}$	100/0/0
	control, cd=0	+0.51	$<10^{-20}$	91/5/4	+0.58	$<10^{-24}$	96/4/0
	control + descent	+0.24	$<10^{-9}$	72/8/20	+0.32	$<10^{-14}$	80/12/8
ER60, final	trained with descent (ratio, %)	99.72	(0.94 s/graph)		99.81	(0.94 s/graph)	
	published, cd=0, $3\|V\|$	+0.42	$<10^{-13}$	87/7/6	+0.47	$<10^{-18}$	92/7/1
	published, cd=0, $4\|V\|$	+0.54	$<10^{-12}$	93/4/3	+0.58	$<10^{-16}$	96/4/0
	published + descent	+0.52	$<10^{-17}$	93/5/2	+0.60	$<10^{-19}$	93/6/1
	random + descent	+1.81	$<10^{-39}$	99/1/0	+1.91	$<10^{-40}$	100/0/0
	control, cd=0	+0.56	$<10^{-18}$	93/5/2	+0.62	$<10^{-23}$	95/5/0
	control + descent	+0.37	$<10^{-16}$	84/10/6	+0.32	$<10^{-15}$	86/10/4

control, p=0.23). On TSP-100 fine-tuning helps with the second seed: the fine-tuned policy beats the published policy with and without descent and both controls, but not random selection with descent (0.13 and 0.22 pp behind). With the first seed it is behind the published policy with descent (0.25 and 0.59 pp).

The validation gaps show the same pattern. For TSP-50 no fine-tuned epoch of either seed reached the published policy's validation gap (0.089% and 0.077% in the two seeds' runs; best fine-tuned epochs 0.112% and 0.118%). For TSP-100 the first seed's best epoch was worse than the published policy (1.084% against 0.855%), and the second seed's best epoch was better (0.553% against 0.794%).

H.8 Results: MIStar

Table 54 gives the gap of every arm and Tables 55 and 56 the paired comparisons. Gaps are to OR-Tools CP-SAT (15 s per instance; proven optimal on 41 of the 88 instances; mean reference makespan 328.97), averaged over instances; differences are in percentage points of this gap. The time budget is the fine-tuned checkpoint's time for 100 steps (about 144 s per batch of 4 instances), in which the published policy at cd=0 makes about 200 steps.

The best-validation checkpoints are taken after 10 of the 80 episodes, while the policy is still almost uniform (normalized entropy 0.9995 and 0.9998 on the test instances). At equal wall-clock time they are tied with the published policy at cd=0 (+0.13 and −0.30 pp, $p \geq 0.17$) and with the controls; with the same number of steps the first seed's checkpoint is ahead of the published policy at cd=0 (+0.76 pp, p=0.001), but in the same time it makes only half as many steps. The first seed's checkpoint is 0.41 pp ahead of the published policy with the same descent (p=0.04); the second seed's is tied with it. Neither beats R_{50} with the same descent (−0.13 pp, p=0.47; −0.54 pp, p=0.006). The final checkpoints are 3.8 to 6.3 pp behind the published policy and R_{50} with every setting, and 1.3 to 2.2

Table 51: NeuOpt fine-tuning on CVRP-100 (500 instances, T=500; both seeds trained and evaluated on an RTX 4090). First row of each block: the fine-tuned policy's gap (%) and its time for T steps. Other rows: fine-tuned minus baseline at equal wall-clock time (pp; positive = fine-tuned better), paired t-test p, W/T/L over instances. Best descent: chosen on the test instances (published: tabu, cd=2; control, seed 1: plain, cd=2 (final) and tabu, cd=1 (best-validation); control, seed 2: tabu, cd=2). Checkpoints: best-validation at epoch 10 for both fine-tuned runs and at epochs 3 and 1 for the two controls; final at epoch 15 (epochs 0–15). †The baseline's 500-step run ended before the budget; its final value is used.

	final checkpoint			best-validation checkpoint		
Arm	diff.	p	W/T/L	diff.	p	W/T/L
Seed 1						
fine-tuned (gap, %)	1.557	(53.0 s)		1.420	(53.8 s)	
published, cd=0	+0.91	$<10^{-32}$	362/0/138	+1.05	$<10^{-45}$	386/0/114
published + descent (cd=1)	+0.26†	$<10^{-4}$	286/0/214	+0.40	$<10^{-9}$	304/0/196
published + best descent	+0.20	$<10^{-3}$	280/0/220	+0.34	$<10^{-8}$	310/0/190
random + descent (cd=1)	+227.6	$<10^{-279}$	500/0/0	+227.8	$<10^{-279}$	500/0/0
control, cd=0	+1.52	$<10^{-78}$	426/0/74	+1.13	$<10^{-52}$	397/0/103
control + best descent	+0.66	$<10^{-25}$	346/0/154	+0.42	$<10^{-12}$	323/0/177
Seed 2						
fine-tuned (gap, %)	1.641	(53.1 s)		1.567	(53.1 s)	
published, cd=0	+0.83	$<10^{-25}$	343/0/157	+0.90	$<10^{-29}$	355/0/145
published + descent (cd=1)	+0.18†	0.006	281/0/219	+0.25†	$<10^{-4}$	292/0/208
published + best descent	+0.12	0.09	279/0/221	+0.19	$<10^{-3}$	279/0/221
random + descent (cd=1)	+227.6	$<10^{-279}$	500/0/0	+227.6	$<10^{-279}$	500/0/0
control, cd=0	+1.66	$<10^{-81}$	433/0/67	+1.07	$<10^{-43}$	382/0/118
control + best descent	+0.89	$<10^{-37}$	370/0/130	+0.33	$<10^{-7}$	298/0/202

Table 52: DACT fine-tuning, best-validation checkpoints (128 instances, 5 runs, T=500). First row of each block: the fine-tuned policy's gap to LKH-3 (%) and its time per step (all 128 instances). Other rows: fine-tuned minus baseline at equal wall-clock time (pp; positive = fine-tuned better), paired t-test p, W/T/L over instances. Descent: edge tabu, LO for the published policy and the control; reversal tabu, LO for random selection. †The baseline's 500-step run ended before the budget; its final value is used. Epochs: seed 1: TSP-50 fine-tuned 3, control 5; TSP-100 1, 10. Seed 2: TSP-50 1, 12; TSP-100 5, 9.

		seed 1 (RTX 4090)			seed 2 (RTX PRO 6000)		
Problem	Arm	diff.	p	W/T/L	diff.	p	W/T/L
TSP-50	fine-tuned (gap, %)	0.107	(21.6 ms/step)		0.130	(7.2 ms/step)	
	published, cd=0	−0.030†	0.10	27/36/65	−0.041†	0.03	33/30/65
	published + descent	−0.041†	0.05	25/36/67	−0.065†	0.003	22/37/69
	random + descent	−0.093	$<10^{-10}$	5/44/79	−0.112	$<10^{-11}$	6/41/81
	control, cd=0	+0.035†	0.06	59/22/47	+0.052†	0.07	48/23/57
	control + descent	−0.058†	$<10^{-4}$	21/37/70	−0.061†	$<10^{-3}$	21/36/71
TSP-100	fine-tuned (gap, %)	1.125	(27.8 ms/step)		0.595	(9.6 ms/step)	
	published, cd=0	+0.147†	0.01	72/0/56	+0.829	$<10^{-26}$	119/0/9
	published + descent	−0.245†	$<10^{-7}$	34/0/94	+0.262†	$<10^{-7}$	85/0/43
	random + descent	−0.667	$<10^{-46}$	2/0/126	−0.133	$<10^{-5}$	42/0/86
	control, cd=0	+0.420†	$<10^{-8}$	88/0/40	+1.165	$<10^{-28}$	122/0/6
	control + descent	−0.167†	$<10^{-4}$	46/0/82	+0.316†	$<10^{-7}$	89/0/39

pp behind a single random move with descent to a local optimum. They are ahead of the controls' final checkpoints at cd=0 and tied with (−0.33 pp, p=0.23) or ahead of (+0.88 pp, p=0.003) them with descent. With the tabu restriction, descent never undoes the selected move; it moves the selected operation again in 10% of steps for the best-validation checkpoints, the published policy, and R_{50}, and in 51% and 69% of steps for the final checkpoints. Plain descent at cd=1 ends at the solution before the move in 81% of the published policy's steps.

Table 53: DACT fine-tuning, final checkpoints (TSP-50: epoch 20; TSP-100: epoch 10). Columns as in Table 52.

		seed 1 (RTX 4090)			seed 2 (RTX PRO 6000)		
Problem	Arm	diff.	p	W/T/L	diff.	p	W/T/L
TSP-50	fine-tuned (gap, %)	0.138	(21.1 ms/step)		0.246	(7.7 ms/step)	
	published, cd=0	$-0.061^{\dagger}$	0.004	28/28/72	$-0.157^{\dagger}$	$<10^{-7}$	24/17/87
	published + descent	$-0.072^{\dagger}$	$<10^{-3}$	22/30/76	$-0.181^{\dagger}$	$<10^{-9}$	13/21/94
	random + descent	-0.124	$<10^{-11}$	3/39/86	-0.229	$<10^{-18}$	1/20/107
	control, cd=0	$+0.165^{\dagger}$	$<10^{-6}$	89/9/30	$+0.434^{\dagger}$	$<10^{-11}$	103/3/22
	control + descent	$+0.029^{\dagger}$	0.23	57/20/51	$-0.201^{\dagger}$	$<10^{-17}$	7/18/103
TSP-100	fine-tuned (gap, %)	1.473	(27.9 ms/step)		0.683	(9.1 ms/step)	
	published, cd=0	$-0.202^{\dagger}$	$<10^{-3}$	45/0/83	$+0.766$	$<10^{-25}$	115/0/13
	published + descent	$-0.593^{\dagger}$	$<10^{-23}$	12/0/116	$+0.174^{\dagger}$	$<10^{-4}$	83/0/45
	random + descent	-1.009	$<10^{-59}$	0/0/128	-0.222	$<10^{-12}$	31/0/97
	control, cd=0	$+0.072^{\dagger}$	0.26	63/0/65	$+1.104$	$<10^{-28}$	123/0/5
	control + descent	$-0.515^{\dagger}$	$<10^{-25}$	14/0/114	$+0.242^{\dagger}$	$<10^{-7}$	87/0/41

Table 54: MIStar fine-tuning evaluation (88 SD2 10×5+mix instances, 100 steps): mean optimality gap to OR-Tools (%) and time for 100 steps (s per batch of 4 instances, RTX 4090). R_1: one random move per step; R_{50}: the best of 50 uniformly sampled candidates, evaluated exactly. Descent: plain or tabu, cd=1 or LO. The published policy at cd=0 reaches 7.34% after 200 steps and 6.86% after 300.

Arm	gap	time	Arm	gap	time
fine-tuned, seed 1, val.-best	7.21	143.8	R_1, cd=0	14.89	11.9
fine-tuned, seed 2, val.-best	7.62	143.9	R_1 + plain, cd=1	12.53	90.5
fine-tuned, seed 1, final	13.30	148.6	R_1 + plain, LO	10.88	387.4
fine-tuned, seed 2, final	12.46	144.4	R_{50}, cd=0	7.83	71.1
published, cd=0	7.97	70.5	R_{50} + tabu, cd=1	**7.07**	143.8
published + plain, cd=1	8.63	144.2	control 1, val.-best, cd=0	7.89	70.7
published + tabu, cd=1	7.61	143.8	control 1, val.-best + descent	7.49	143.6
published + plain, LO	8.35	246.6	control 1, final, cd=0 / + descent	14.58 / 12.97	70.8 / 146.7
published + tabu, LO	7.26	266.8	control 2, final, cd=0 / + descent	14.98 / 13.34	69.6 / 147.7

Table 55: MIStar fine-tuning, best-validation checkpoints: fine-tuned minus baseline in gap (pp; positive = fine-tuned better) with the same number of steps and at equal wall-clock time, paired t-test p, and W/T/L over the 88 instances (equal time). The second control's best-validation checkpoint is the published policy.

	seed 1					seed 2				
Baseline	steps	p	time	p	W/T/L	steps	p	time	p	W/T/L
published, cd=0	+0.76	0.001	+0.13	0.58	38/15/35	+0.35	0.13	−0.30	0.17	33/10/45
published + plain, cd=1	+1.42	$<10^{-7}$	+1.43	$<10^{-7}$	59/12/17	+1.01	$<10^{-4}$	+1.02	$<10^{-4}$	51/14/23
published + tabu, cd=1	+0.41	0.04	+0.41	0.04	41/21/26	−0.00	0.99	−0.01	0.98	38/18/32
published + tabu, LO	+0.05	0.80	+0.83	$<10^{-3}$	52/15/21	−0.36	0.11	+0.42	0.07	43/16/29
R_1 + plain, LO	+3.67	$<10^{-18}$	+3.95	$<10^{-20}$	81/3/4	+3.26	$<10^{-15}$	+3.54	$<10^{-18}$	79/3/6
R_{50}, cd=0	+0.63	0.004	−0.00	0.98	39/14/35	+0.22	0.34	−0.39	0.06	26/12/50
R_{50} + tabu, cd=1	−0.13	0.46	−0.13	0.47	33/19/36	−0.54	0.006	−0.54	0.006	24/16/48
control, cd=0	+0.68	0.005	+0.00	0.99	37/16/35	+0.35	0.13	−0.30	0.17	33/10/45
control + descent	+0.29	0.20	+0.28	0.21	36/21/31	−0.00	0.99	−0.01	0.98	38/18/32

H.9 RESULTS: L2S

Prediction and rules. Before the first L2S fine-tuning run we recorded the prediction that fine-tuning would not beat random selection with descent at equal wall-clock time, because the pre-fine-tuning comparison is negative (−1.65 pp with the same number of steps), and that it might beat the published policy with descent. Before any evaluation we fixed the rules: "beats" means a lower mean gap with a two-sided paired t-test $p<0.05$ over the 30 (instance, seed) pairs; the checkpoints are the final one and the best-validation one among batches ≥ 10; at equal wall-clock time, each baseline is read at the time the fine-tuned policy needs for 5000 steps. We measured this time in three ways (the

Table 56: MIStar fine-tuning, final checkpoints (episode 80). Columns as in Table 55; the control rows use the controls' final checkpoints.

	seed 1					seed 2				
Baseline	steps	p	time	p	W/T/L	steps	p	time	p	W/T/L
published, cd=0	-5.33	$<10^{-27}$	-6.01	$<10^{-29}$	0/1/87	-4.49	$<10^{-22}$	-5.17	$<10^{-27}$	0/6/82
published + plain, cd=1	-4.68	$<10^{-23}$	-4.70	$<10^{-24}$	1/4/83	-3.84	$<10^{-19}$	-3.84	$<10^{-19}$	4/6/78
published + tabu, cd=1	-5.69	$<10^{-27}$	-5.69	$<10^{-27}$	1/2/85	-4.85	$<10^{-24}$	-4.85	$<10^{-24}$	2/5/81
published + tabu, LO	-6.04	$<10^{-25}$	-5.27	$<10^{-24}$	1/2/85	-5.20	$<10^{-22}$	-4.41	$<10^{-21}$	3/5/80
R_1 + plain, LO	-2.43	$<10^{-12}$	-2.15	$<10^{-10}$	16/9/63	-1.59	$<10^{-5}$	-1.30	$<10^{-4}$	21/10/57
R_{50}, cd=0	-5.47	$<10^{-28}$	-6.13	$<10^{-30}$	1/1/86	-4.63	$<10^{-21}$	-5.27	$<10^{-25}$	2/3/83
R_{50} + tabu, cd=1	-6.23	$<10^{-28}$	-6.29	$<10^{-28}$	1/1/86	-5.39	$<10^{-25}$	-5.41	$<10^{-25}$	0/6/82
control, cd=0	$+1.28$	$<10^{-7}$	$+1.28$	$<10^{-7}$	51/30/7	$+2.53$	$<10^{-12}$	$+2.50$	$<10^{-12}$	66/17/5
control + descent	-0.33	0.23	-0.33	0.23	35/15/38	$+0.88$	0.003	$+0.88$	0.003	47/15/26

elapsed time recorded in each run, the median time per step, and times rescaled by two reference arms re-run at the end of the evaluation); all three give the same conclusions, and we report the first.

Setup and verification. We fine-tune the released 10×10 checkpoint, the one used in Appendix F.6, with the authors' n-step REINFORCE code on generated 10×10 instances (Table 45). The descent in the loop is tabu-restricted descent to a local optimum, the published policy's best setting; the reward of a policy step is the incumbent makespan before the move minus the incumbent after the descent. With cd=0 our training step reproduces the authors' pipeline: feasible moves, sampled moves, rewards, and incumbents are identical, and the first gradient agrees to a relative error of 3.7×10^{-5}. The loaded weights reproduce the trajectories of Appendix F.6 exactly, and all baseline runs of this evaluation match that study to three decimals. The test protocol is that of Appendix F.6 (Taillard 15×15, three seeds, 5000 steps).

Results. Table 57 gives the comparisons. The prediction holds. At equal wall-clock time the fine-tuned policy is worse than random selection with tabu-restricted descent to a local optimum, by 0.41 and 0.64 pp with the best-validation checkpoints and by 0.56 and 0.64 pp with the final ones (all $p\leq0.042$), and worse than random selection with plain descent at cd=2 (slightly stronger than tabu-restricted descent at equal time) by 0.45 to 0.66 pp. With the same number of steps it is tied with random selection with descent ($+0.02$ to $+0.36$ pp in its favor, $p\geq0.12$). It is better than the published policy with its best descent ($+1.62$ to $+1.81$ pp at equal time) and than the published policy at cd=0 ($+3.87$ to $+4.06$ pp). It is better than the best-validation controls with descent ($+1.06$ pp for both seeds) and than the final controls by 0.64 (seed 1, $p=0.011$) and 0.37 pp (seed 2, $p=0.10$). A fine-tuned step with descent takes 20.5 ms, against 8.4 ms for a random step with the same descent, so random selection makes 2.4 times as many steps in the same time.

What changes. The gain over the published policy comes from weight changes: the released weights with the batch normalization statistics after the first validation give 3.84% against 4.17% with tabu-restricted descent ($p=0.37$). Without descent the fine-tuned policy is worse than the published one (7.90% against 6.75% at cd=0, seeds pooled, final checkpoints, $p<10^{-3}$): it has adapted to the loop with descent. Fine-tuning without descent (the control) also improves the result with tabu-restricted descent (3.03% against 4.17%, seeds pooled, final checkpoints, $p<10^{-3}$) but not at cd=0 (6.81% against 6.75%, $p=0.83$); we do not have an explanation for this. On the 10×10 validation instances the validation gap of the runs with descent falls from 3.00% at batch 0 to 1.77% and 1.79% at batch 10 and then stays between 1.57% and 2.49%; random selection with the same descent scores 2.01% on this set. The mean policy entropy on the validation states rises from 0.39 at batch 0 to 0.87 and 0.99 at batch 10 and stays between 0.59 and 1.17 afterwards, while that of the controls varies between 0.17 and 0.76 without a trend. The number of descent moves per policy step in training rises from 1.45 to about 1.6. The fine-tuned policy is thus better than the published one and has higher entropy, and its gain is reached within the first 10 of the 480 batches.

H.10 Policy entropy during fine-tuning

Table 59 gives the entropy of the DACT policy on the training rollouts, divided by $\log N^2$, the entropy of a uniform distribution over the N^2 node pairs. In the three runs that do not improve, the entropy

Table 57: L2S fine-tuning with tabu-restricted descent to a local optimum in the training loop (Taillard 15×15, 3 seeds, 30 pairs, 5000 steps). First row of each block: the fine-tuned policy's gap (%) and time per step. Other rows: fine-tuned minus baseline in gap (pp; positive = fine-tuned better) with the same number of steps and at equal wall-clock time, paired t-test p, and W/T/L over pairs (equal time). Published + descent and random + descent use tabu-restricted descent to a local optimum. The control uses the same descent at test time.

Checkpoint	Baseline	steps	p	time	p	W/T/L
seed 1, val.-best	fine-tuned (gap, %)	2.29	(20.8 ms/step)			
(batch 200)	published, cd=0	+4.46	$<10^{-15}$	+4.06	$<10^{-14}$	30/0/0
	published + descent	+1.88	$<10^{-6}$	+1.81	$<10^{-6}$	27/0/3
	random + descent	+0.23	0.38	−0.41	0.042	8/1/21
	random + plain, cd=2	+0.36	0.12	−0.45	0.044	10/2/18
	control (batch 410) + descent	+1.12	$<10^{-4}$	+1.06	$<10^{-5}$	23/1/6
seed 1, final	fine-tuned (gap, %)	2.44	(20.5 ms/step)			
(batch 480)	published, cd=0	+4.31	$<10^{-13}$	+3.91	$<10^{-13}$	30/0/0
	published + descent	+1.73	$<10^{-4}$	+1.66	$<10^{-4}$	27/0/3
	random + descent	+0.08	0.69	−0.56	0.003	11/0/19
	random + plain, cd=2	+0.21	0.32	−0.60	0.015	9/1/20
	control (batch 480) + descent	+0.72	0.008	+0.64	0.011	18/1/11
seed 2, val.-best = final	fine-tuned (gap, %)	2.50	(20.5 ms/step)			
(batch 480)	published, cd=0	+4.25	$<10^{-13}$	+3.87	$<10^{-12}$	30/0/0
	published + descent	+1.67	$<10^{-5}$	+1.62	$<10^{-5}$	26/1/3
	random + descent	+0.02	0.90	−0.64	$<10^{-3}$	3/3/24
	random + plain, cd=2	+0.15	0.44	−0.66	0.003	7/0/23
	control (batch 310) + descent	+1.10	$<10^{-5}$	+1.06	$<10^{-5}$	22/2/6
	control (batch 480) + descent	+0.40	0.09	+0.37	0.10	17/1/12

rises from 0.25–0.32 to a peak of 0.70–0.99 and is 0.41–0.76 at the end, so the fine-tuned policy moves toward uniform random selection, which is the best selector with this descent. In the run that improves (TSP-100, second seed) it stays between 0.27 and 0.34. The controls stay between 0.22 and 0.44. The number of descent moves per policy step in training rises with the entropy: in the first TSP-100 run from 3.1 in the first epoch to 8.3–9.4 in the last six, close to the 9.4 of random selection with the same descent on TSP-100; in the second TSP-50 run from 2.8 to a peak of 6.9, against 6.95 for random selection.

The released MIStar policy is almost uniform (normalized entropy 1.000, the entropy divided by $\log n$ for n feasible moves). Fine-tuning lowers its entropy, with and without descent, and the validation makespan rises as the entropy falls (Table 58). From episode 30 on, every validation makespan of the four runs is 22 to 33 units worse than at episode 0. PPO raises the probability of the best of the 50 sampled candidates, so the distribution sharpens and fewer distinct candidates are evaluated per step; the exact evaluation of 50 candidates per step, which is where the published policy's strength comes from, then helps less. The first pair of runs with learning rate 5×10^{-4} collapsed within ten episodes (Appendix H.1).

Table 58: MIStar fine-tuning: normalized entropy of the policy on the training rollouts (mean over episode ranges) and validation makespan (32 instances, 100 steps, the descent used in the run) at selected episodes.

	normalized entropy, episodes						validation makespan, episode				
Run	1–5	6–10	11–20	21–30	31–50	51–80	0	10	20	30	80
fine-tuned, seed 1	1.000	0.998	0.752	0.217	0.036	0.036	369.9	**367.7**	379.7	395.1	393.1
fine-tuned, seed 2	1.000	0.999	0.953	0.041	0.016	0.041	369.9	**368.5**	388.6	396.1	391.8
control, seed 1	1.000	1.000	1.000	0.725	0.099	0.034	370.5	**369.8**	370.2	403.0	401.7
control, seed 2	1.000	0.942	0.331	0.042	0.036	0.029	**370.5**	388.9	401.7	401.8	402.6

Table 59: DACT policy entropy on the training rollouts, divided by $\log N^2$: mean over the first epoch, maximum over epochs (epoch in parentheses), and last epoch. Validation gap (%, 500 steps, the descent used in the run): published policy, best fine-tuned epoch (epoch), last epoch.

		normalized entropy			validation gap		
Run		first	max	last	published	best	last
TSP-50, seed 1	fine-tuned	0.32	0.70 (5)	0.41	0.089	0.112 (3)	0.179
	control	0.25	0.44 (17)	0.38	0.278	0.480 (5)	0.844
TSP-50, seed 2	fine-tuned	0.25	0.99 (6)	0.67	0.077	0.118 (1)	0.232
	control	0.25	0.41 (19)	0.40	0.274	0.485 (12)	2.009
TSP-100, seed 1	fine-tuned	0.27	0.92 (9)	0.76	0.855	1.084 (1)	1.465
	control	0.22	0.22	0.22	2.304	2.723 (10)	2.723
TSP-100, seed 2	fine-tuned	0.30	0.34 (4)	0.29	0.794	0.553 (5)	0.655
	control	0.22	0.23	0.22	2.312	2.621 (9)	2.814