IMPERFECT IS PERFECT: LEARNED COLUMN PREFILLING FOR BRANCH-PRICE-AND-CUT

Anonymous authors
Paper under double-blind review

ABSTRACT

Column generation folklore warns that initial columns hurt: a warm start may bias the dual and slow the root of branch-price-and-cut. We show the opposite once the columns are imperfect in the right way. Preloading the support of the optimal root LP, the best columns one could know in advance, fails to finish on half of our hardest instances; a pool of ordinary routes that need not belong to any optimal solution, held outside the master and priced at the current smoothed dual, speeds the root up. Because the columns need not be optimal, producing them becomes a learning task: a neural edge scorer guides a constructive generator that builds the pool in seconds. A second network predicts the initial center of Wentges smoothing from the instance file, and with it the direction of the smoothed pricing queries; the solver's own updates run unchanged. With prefill and center in place, exact pricing dominates root pricing time, so we rebuild bucket-graph labeling around its dominance test for the GPU and certify on the CPU. Inside RouteOpt, the state-of-the-art exact solver for capacitated vehicle routing, the pipeline reaches the same certified root bound in 13.2 % of vanilla RouteOpt's root pricing time over 95 instances with 300 to 1000 customers, prefill alone in 26.1 %, and 17.5 times faster in the best case. In an exact branch-price-and-cut solver for unrelated parallel machine scheduling, prefill with smoothing and the learned center certifies all 25 benchmark instances, 12 times faster than the solver's own column generation, which certifies 20 of them.

1 INTRODUCTION

Branch-price-and-cut (BPC) is the leading exact approach for large-scale NP-hard optimization problems, such as vehicle routing, crew and machine scheduling, and cutting stock (Barnhart et al., 1998; Lübbecke & Desrosiers, 2005; van den Akker et al., 1999; Costa et al., 2019). Within branch-and-bound, column generation (CG) solves the Dantzig–Wolfe relaxation by generating necessary variables on the fly through solving a pricing subproblem, and cutting planes are typically applied to strengthen the bound. Work on accelerating BPC has therefore concentrated on branching, pricing, and cutting (Fukasawa et al., 2006; Jepsen et al., 2008; Baldacci et al., 2011; Pecin et al., 2017a; Pessoa et al., 2020; Sadykov et al., 2021), with arc elimination (Irnich et al., 2010), route enumeration (Baldacci et al., 2008) and dual stabilization (du Merle et al., 1999; Wentges, 1997; Pessoa et al., 2018) as complementary accelerators. All of these techniques apply across problems, but pricing is where the problem's structure enters. Our main test bed is RouteOpt (You & Yang, 2026), an exact solver for the capacitated vehicle routing problem (CVRP) with a state-of-the-art integrated implementation of all of the above.

With the CPU-side toolbox this mature, learning and GPU computing offer two further directions. Learning has been applied to branching (Khalil et al., 2016; Gasse et al., 2019; Bengio et al., 2021; You et al., 2026), to cutting plane generation and selection (Tang et al., 2020; Paulus et al., 2022; Wang et al., 2023; Deza & Khalil, 2023), to column generation and selection (Morabit et al., 2021; Chi et al., 2022; Morabit et al., 2023; Shen et al., 2022; Xu et al., 2025; Koutecká et al., 2025; Václavík et al., 2018) and to dual stabilization (Kraul et al., 2023; Babaki et al., 2022; Shen et al., 2024; Sarkar et al., 2025; Fang et al., 2025; Wang & Khir, 2026). On the GPU side, first-order methods have moved the linear-programming solve onto the GPU (Lu & Yang, 2025), and GPU labeling exists for constrained shortest paths outside BPC (Lu et al., 2020), but the exact pricer of a state-of-the-art BPC solver has stayed on the CPU. It is worth noting that gains beyond a modest

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.

research baseline can shrink or reverse inside a state-of-the-art solver, so a method of practical value has to be demonstrated against an efficient implementation.

Our target is *root pricing*: the column generation that solves the root relaxation before any cutting plane is added. Its wall-clock time is our outcome, and every configuration we compare that completes ends at the same certified root bound, so root pricing is a clean test bed and delivers a certified lower bound of practical value on its own. Our method is a combination of three moves: a prefilled archive supplies candidate columns, a learned initial center shapes the pricing queries, and a GPU accelerates the exact pricing level.

We started from a piece of folklore. Textbooks warn that a warm start, even from an excellent primal solution, can hinder column generation (Lübbecke & Desrosiers, 2005; Vanderbeck, 2005): the initial columns pull the dual toward the solution they encode, and in the root of a modern BPC, heuristic routes are elementary and need not suffice to attain or certify the ng-route bound that pricing works over (Baldacci et al., 2011), while a large pool adds to the complexity of solving the restricted master problem (RMP). Our measurements confirmed the warning on deep instances: preloading a heuristic pool into the master slowed the root down there, and preloading the support of the optimal root LP, the best columns one could know in advance, was the worst choice of all (Section 3.1, Appendix D.1). What did help was the opposite of what the folklore protects against. A diverse pool of ordinary, non-optimal routes, held *outside* the master as a lazy archive and queried at the current smoothed dual, shortens root pricing by a large factor on every benchmark set and route-size class we test (Section 4.2). Learning for column generation has so far concentrated on the pricing loop itself, on selecting, ranking, or generating columns during the iterations. We take a different perspective: leave the pricer untouched and learn the prefilling, with a constructive generator guided by a neural edge scorer, whose pool need not be optimal.

Prefilling alone is not the whole story. In an earlier development experiment, dumping a metaheuristic pool into the initial master slowed the root down, as the folklore predicts (Appendix D.1): the pricer spent many rounds searching for the direction of the dual and accumulated a bloated LP. The classical remedy is dual smoothing (du Merle et al., 1999; Wentges, 1997; Pessoa et al., 2018), and the learned variants predict the optimal dual and install it as the reference point (Kraul et al., 2023; Shen et al., 2024). Where those methods learn the values of the dual, we learn its direction: the RMP has many optimal duals, so what matters is the direction from which the query approaches the optimum, not its distance to one optimal point. We therefore learn the *initial* center of the standard Wentges smoothing from the instance file, calibrate its scale, and let the solver's own rule move the center and adapt the weight from there. On the hardest instances in particular, the learned center becomes decisive, especially in combination with prefill (Section 4.2, Appendix D.2).

Prefill and center cut the run time of heuristic pricing sharply, so exact pricing, the level that certifies optimality, becomes the dominant share of root time. Bucket-graph labeling is the exact pricer of modern routing solvers (Sadykov et al., 2021), and its dominance test is the cost that structure exists to contain; parallel CPU labeling has been developed for it (Petersen & Spoorendonk, 2025), while GPU work on BPC has addressed the linear-programming solve (Lu & Yang, 2025). We rebuild exact bucket-graph labeling around the dominance test for the GPU, with a final CPU certificate (Sections 3.3 and 4.3). The construction applies to any bucket-graph-based pricer once the resource extension, the dominance rule, and the join are adapted to the problem. Our **contributions** are summarized as follows.

- A general learn-to-prefill framework: a neural edge scorer and a constructive generator whose pool is served to root pricing as a lazy archive. This opens a new direction for learning in column generation, one that is both highly effective and broadly applicable (Section 3.1; Appendix D.1).
- A learned dual direction under prefilled columns: the initial center of Wentges smoothing, predicted by a neural network, together with a calibration that emphasizes the direction and relaxes its absolute magnitude to better steer the prefilled columns (Section 3.2, Appendix D.2).
- A GPU backend for exact bucket-graph labeling with CPU certification, which applies to any BPC solver with bucket-graph pricing (Sections 3.3 and 4.3).
- A general plug-in pipeline for problems solved by BPC, which delivers strong performance inside optimized exact BPC solvers: the state-of-the-art CVRP solver RouteOpt and an exact solver for unrelated parallel machine scheduling (Sections 4.2 to 4.4).

2 BACKGROUND

Set partitioning, its dual, and pricing. Let $C = \{1, \dots, n\}$ be the customers, 0 the depot, q_i the demand of customer i, Q the capacity, and d_{0i} the depot distance. Over a route set $\mathcal{R}$, the master problem and its dual are

$$\min_{\lambda \geq 0} \Big\{ \sum_{r \in \mathcal{R}} c_r \lambda_r : \sum_{r \in \mathcal{R}} a_{ir} \lambda_r = 1,\ i \in C \Big\} \;=\; \max_{u \in \mathbb{R}^n} \Big\{ \sum_{i \in C} u_i : \ rc_r(u) \geq 0,\ r \in \mathcal{R} \Big\}, \qquad (1)$$

with c_r the cost of route r, a_{ir} its number of visits to customer i and $rc_r(u) = c_r - \sum_{i \in C} a_{ir} u_i$ its reduced cost. A dual point u is feasible exactly when $\min_r rc_r(u) \geq 0$, so *pricing* is the separation problem of the dual. Column generation solves the RMP over a subset of $\mathcal{R}$, starting from the singleton routes $(0, i, 0)$, prices at its dual u and adds the routes found (Barnhart et al., 1998; Lübbecke & Desrosiers, 2005); when pricing certifies u, the RMP value is the root bound. Elementary pricing is intractable at these sizes; the ng-route relaxation (Baldacci et al., 2011), which lets a route revisit a customer only after visiting a customer whose neighborhood does not contain it, made the exact solution of large instances practical, and $\mathcal{R}$ is the set of ng-feasible routes in RouteOpt as in other modern solvers (Appendix A.4). This paper calls *prefilling* any supply of candidate columns prepared before column generation starts, whether preloaded into the initial RMP or held outside it and served on demand.

Heuristic and exact pricing. Pricing is a shortest-path problem with resource constraints, solved by bucket-graph labeling (Sadykov et al., 2021) at three levels, as in generic BPC solvers (Pessoa et al., 2020): two *heuristic* levels (LIGHT and HEAVY) drop part of the label state, are cheap, may miss columns and account for most of the iterations; the *exact* level keeps every state the dominance rule needs, is expensive, and alone certifies that no column prices out. Root pricing stops when an exact pricing call at the raw dual u that explores its whole label space, a *complete* call, returns no route with $rc_r(u) < -\epsilon$, where $\epsilon > 0$ is the solver's pricing tolerance; its wall clock is what we measure (Section 4.1). This pricer is highly efficient, a bar that learned column generators rarely clear, which is why our two learned components leave its algorithm untouched; the GPU component of Section 3.3 reimplements its exact level without changing what it returns.

Dual stabilization. The master (1) is highly degenerate, so the RMP dual jumps between optimal solutions from one iteration to the next (du Merle et al., 1999; Pessoa et al., 2018). Wentges smoothing (Wentges, 1997) prices at a convex combination of the RMP dual u_t and a stabilization center c_t,

$$q_t \;=\; (1 - \alpha_t)\, u_t + \alpha_t\, c_t, \qquad \alpha_t \in [0, 1), \qquad (2)$$

where c_t is the dual point with the best certified Lagrangian bound so far and α is adapted from $\alpha_0 = 0.5$ by the solver's weight rule, (10) below; candidates are re-priced at u_t and admitted only if $rc_r(u_t) < -\epsilon$, so the termination test is unchanged.

3 METHOD

Our method combines three moves, arranged as in Figure 1. Before the run, two neural models read the instance file. An *edge scorer* drives a constructive generator that builds a large pool of candidate routes, and a *set transformer* predicts the initial center of dual smoothing. Root pricing then starts from the singleton columns and alternates two loops. In the first, the smoothed query q_t of (2) is served first from the pool, which acts as a lazy archive, and then by heuristic pricing, with a primal simplex re-solve of the RMP after each batch. When the heuristic levels return nothing, exact pricing takes over on the GPU, again alternating with primal simplex, until a complete exact call at the raw dual, re-proved on the CPU, returns nothing and the root bound is certified.

3.1 LEARNING TO PREFILL

Columns act on the dual. A set T of columns in the master restricts the dual of (1) to the polyhedron

$$D_T = \{u : \ a_r^\top u \leq c_r,\ r \in T\} \ \supseteq\ D_{\mathcal{R}}, \qquad F_T = \arg\max\{\textstyle\sum_i u_i : \ u \in D_T\}, \qquad (3)$$

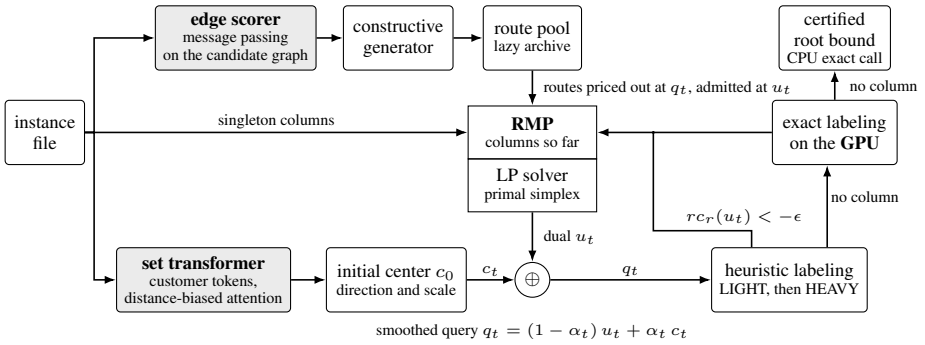

Figure 1: The pipeline. Two neural models act on the primal and the dual side of the RMP, the LP solver connects the two, and exact pricing runs on the GPU with CPU certification.

where a_r is the visit vector of route r, for an elementary route the incidence vector of its customer set; RouteOpt's vehicle-count row is suppressed here and restored in (6) (Appendix A.2 states D_T and F_T with it). The singleton columns make the RMP feasible and bound its dual objective above, and primal simplex returns a basic optimal dual $u_T \in F_T$, a vertex of D_T. In exact arithmetic root pricing terminates exactly when $u_T \in D_{\mathcal{R}}$, that is, when $m(u_T) = \min_r rc_r(u_T) \geq 0$; the implementation uses the pricing tolerance of Section 2. This is why preloading the support of an optimal solution failed in our hands (Appendix D.1). If T contains the support of a full-master optimum, the RMP already attains the optimal value z^*, yet F_T only meets $D_{\mathcal{R}}$: recovering the optimal primal value says nothing about whether the vertex the simplex returns lies in $D_{\mathcal{R}}$ (Proposition A.2), so preloading a converged support leaves termination to that choice. A column restricts the dual only through its visit vector and its cost, so our design hypothesis is that what shortens the run is *coverage*: many distinct customer sets, each customer in many of them, so that (3) constrains u_T from many directions along the whole trajectory; Appendix D.1 evaluates this design against preloading. A seed-based population heuristic produces exactly such coverage, its seeds exploring different regions of the route space; hybrid genetic search (Vidal, 2022) is the strongest known metaheuristic for the CVRP, and we learn to reproduce the routes it samples.

Learning task. Let $E(I)$ be the candidate edge set of instance I, a nearest-neighbor graph plus the depot edges. Among the reference-pool routes that price out at the reference center, the tree-predicted dual of Appendix B.1.1, let $\nu_e \in [0, 1]$ be the fraction of those visiting i or j that traverse the edge $e = (i, j)$; these are the *heated* edges, which measure edge usage in the routes the reference dual asks for. The scorer f_θ maps the instance-only node and edge features $\Phi(I)$ (Appendix B.1.1) to a heat map on $E(I)$,

$$\hat{z} = f_\theta(\Phi(I)), \qquad \hat{\nu}_e = \big[\max\{\hat{z}_e, 0\}\big]^2, \qquad \mathcal{L}_{\text{edge}}(\theta) = \sum_I \sum_{e \in H(I)} \big(\hat{z}_e - \sqrt{\nu_e}\big)^2, \tag{4}$$

where the supervised set $H(I)$ adds the reference-pool edges to $E(I)$; message passing runs on $E(I)$ and the edge head is evaluated on $H(I)$ in training and on $E(I)$ at generation, and the model is selected on validation rank correlation with ν. Heat is contextual, decided by an edge's competitors at both endpoints, so f_θ is a message-passing network with messages symmetric in the endpoints. Its graph is constructed from the instance alone and is identical during training and generation.

From heat map to columns. Algorithm 1 turns the heat map into a pool P of routes; its constants are listed in Appendix B.1.2. The pool is kept outside the initial RMP and supplies columns lazily. At pricing round t, before any labeling, the archive offers its B most negative eligible routes (the k_t largest values of $-rc_r(q_t)$) and re-prices them at the raw dual,

$$\begin{aligned} A_t &= \{r \in P_t :\ rc_r(q_t) < 0\}, \qquad O_t = \mathrm{Top}_{k_t}\big(-rc_r(q_t);\, A_t\big), \quad k_t = \min\{B, |A_t|\}, \\ C_t &= \{r \in O_t :\ rc_r(u_t) < -\epsilon\}, \qquad P_{t+1} = P_t \setminus O_t, \end{aligned} \tag{5}$$

so each route is offered once, and C_t is admitted by the same test as a generated column, so the archive cannot change the value CG converges to. The labeling call is skipped when $C_t \neq \emptyset$. The recipe carries over to any problem with a seed-based heuristic that samples diverse columns; the scheduling case study of Section 4.4 uses randomized list scheduling.

Algorithm 1 Learned constructive prefill. $P \uplus S$ inserts the routes of S into P by customer set, keeping the cheaper ordering; $\kappa_i(P)$ counts the routes of P through i; $\mathcal{N}(i)$ is the heat-filtered neighborhood of i; Polish is intra-route 2-opt; τ is the sampling exponent, δ, δ' probability floors, γ the coverage exponent, and LS relocate, swap, and 2-opt* (constants in Appendix B.1.2; split and local search run twice per tour).

Require: heat map $\hat{\nu}$ from (4), seeds $w = 1, \dots, W$, time budget T

for $w = 1, \dots, W$ in parallel **do**
  $P^{(w)} \leftarrow \emptyset$
  repeat
    draw τ and a start i_1 with $\Pr[i_1 = i] \propto \max\{\hat{\nu}_{0i}, \delta\} \left(1 + \kappa_i(P^{(w)})\right)^{-\gamma}$
    grow $\omega = (i_1, \dots, i_n)$ with $\Pr[i_{k+1} = j \mid i_1, \dots, i_k] \propto \left[\max\{\hat{\nu}_{i_k j}, \delta'\}\, e^{\xi_{kj}}\right]^{\tau}$ over unvisited $j \in \mathcal{N}(i_k)$, ξ independent noise, else the nearest unvisited customer
    $S \leftarrow \mathrm{Polish}\big(\mathrm{Split}(2\mathrm{opt}(\omega))\big)$; $P^{(w)} \leftarrow P^{(w)} \uplus S$ ▷ optimal split (Prins, 2004)
    for each local-search pass: $S \leftarrow \mathrm{Polish}(\mathrm{LS}(S))$; $P^{(w)} \leftarrow P^{(w)} \uplus S$
  until the time budget T is spent, or consecutive tours add no new customer set
end for
return $P \leftarrow \biguplus_w P^{(w)}$

3.2 Learning the dual center

Not the optimal-dual task. Learned dual stabilization predicts the optimal dual and installs it as the reference point. We train on the same certified labels, the optimal root dual $\pi^* = (\pi^*_C, \pi^*_{\mathrm{veh}})$ of each label instance (Appendix B.3), but use the prediction to initialize the Wentges center, and we judge it by the pricing trajectory that follows. The RMP admits many optimal duals, and the smoothed query of (2) needs a *direction* to price from and a *scale* at which pricing stays productive, not proximity to one optimal point. By (1), a center c is dual feasible exactly when $m(c) \geq 0$. The solver certifies an installed center with one exact pricing at c, whose outcome is the Lagrangian bound

$$L(c) = \sum_{i \in C} c_i + n \, \min\{0, \, m(c)\}, \tag{6}$$

valid because every route covers at least one customer, so $\sum_r \lambda_r \leq n$ for every feasible λ (Proposition A.3). With RouteOpt's row $\sum_r \lambda_r \geq K_{\min}$, $K_{\min}$ a lower bound on the fleet size, the center carries a vehicle dual $c_{\mathrm{veh}} \geq 0$, every reduced cost becomes $c_r - a_r^\top c_C - c_{\mathrm{veh}}$, and the first term of (6) becomes $\sum_i c_i + K_{\min} c_{\mathrm{veh}}$. The same call reads off how aggressive the center is: at $m(c) \geq 0$ it returns no column, and a center at or near feasibility returns no column at installation, so the columns must come from the calls that follow. The certified optimal dual is the extreme case, feasible and pricing out nothing. We therefore calibrate the center to keep pricing productive.

Learning task. The dual lives on the customers, so the model assigns one output to each customer: a set transformer g_φ (Lee et al., 2019) over customer tokens, with distance-biased attention to represent the interactions among customers, maps the instance-only customer and context features $Z(I)$ (Appendix B.2.1) to singleton-normalized coordinates $\hat{y}$ and a log magnitude $\hat{\ell}$. With $D_0 = \mathrm{diag}(2d_0)$, the direction is scored in the original dual units and the magnitude as the ratio of the customer-dual norm to the singleton-cost norm,

$$(\hat{y}, \hat{\ell}) = g_\varphi(Z(I)), \qquad \rho^* = \|\pi^*_C\|_2 \,/\, \|2d_0\|_2,$$
$$\mathcal{L}_{\mathrm{center}}(\varphi) = \sum_I \Big[1 - \cos\left(D_0 \hat{y}, \pi^*_C\right) + \beta \left(\hat{\ell} - \log \rho^*\right)^2 \Big]. \tag{7}$$

Direction and scale are produced separately: the installed center is $c_0 = (c_{0,C}, c_{\mathrm{veh}})$ with

$$c_{0,C} = \Pi_{\mathcal{B}}\big(s \, D_0 \hat{y}\big), \qquad \mathcal{B} = \{c : \; 0 \leq c_i \leq \max(2d_{0i} - c_{\mathrm{veh}}, 0)\}, \qquad s : \; \big\|\Pi_{\mathcal{B}}(s \, D_0 \hat{y})\big\|_2 = T, \tag{8}$$

where $\Pi_{\mathcal{B}}$ clips to a singleton-based box, the multiplier s is found by bisection so that the customer-block norm after clipping equals the target T, the box corner being installed when T exceeds its norm, and T is a calibrated geometric mean of auxiliary neural magnitude estimates computed from customer features and instance context, independently of the deployed direction. The vehicle dual c_{veh} comes from gradient-boosted trees.

Smoothing from a learned start. The predictor is evaluated once. At iteration 0 the solver installs c_0 and certifies $L(c_0)$ by one exact call at c_0 whose columns are not added; this replaces the seeding exact call that the stabilized baseline pays for its first center. From then on the rule of Section 2 runs unchanged, organized in exact-pricing sequences: a *sequence* is a maximal run of consecutive exact calls at one raw dual u_s, the mis-price ladder below together with the call that closes it. Sequence s freezes the raw dual u_s, the center c^s, the incumbent bound L_s and the starting weight α^s; let $\mathcal{Q}_s$ be its complete exact queries, $q_s^{\text{best}} \in \arg\max_{q \in \mathcal{Q}_s} L(q)$, and α_{last} the weight of the last complete query q_{last}. When the sequence closes,

$$(c^{s+1}, L_{s+1}) = \begin{cases} (q_s^{\text{best}}, L(q_s^{\text{best}})) & L(q_s^{\text{best}}) > L_s + \epsilon_L, \\ (c^s, L_s) & \text{otherwise}, \end{cases} \tag{9}$$

$$\alpha^{s+1} = \begin{cases} \max\{0,\ \alpha_{\text{last}} - \eta\} & h_s > \epsilon_g, \\ \min\{\bar{\alpha},\ \alpha_{\text{last}} + \eta(1 - \alpha_{\text{last}})\} & h_s < -\epsilon_g, \\ \alpha_{\text{last}} & \text{otherwise}, \end{cases} \qquad h_s = g(q_{\text{last}})^\top (u_s - c^s), \tag{10}$$

where g is a supergradient of L at the last complete query and the constants η, $\bar{\alpha}$, α_0 and the tolerances are RouteOpt's defaults, which we do not change (Table 4). The learned prediction initializes the center; every later center is selected by certified Lagrangian bounds. Two fallback rules keep a miscalibrated center from stalling the queries. An exact call at q_t with no column admissible at u_t is a mis-price rather than a certificate, and while that remains so the query is retried along the ladder $\alpha_j = \max\{0,\ 1 - (j+1)(1 - \alpha^s)\}$ until it reaches u_t (Appendix A.2). An empty smoothed LIGHT call is retried once at u_t; if the retry admits columns the rest of that LIGHT level prices at the raw dual, otherwise the solver advances to HEAVY. Once the deployed endgame switch fires (Appendix B.2.2), α is set to 0 and adaptation stops.

3.3 GPU EXACT PRICING

The first two components change where root pricing spends its time. Prefill and a learned center reduce the work at the heuristic levels, and in our development runs this left exact pricing as the largest stage of the run and the next target for acceleration. The CPU profile of Appendix C.2 identifies the dominance test as the hotspot, and its per-level counters show that an exact call evaluates orders of magnitude more dominance predicates than a heuristic call. Appendix C.1 reviews bucket-graph pricing: labels are extended bucket by bucket, each arrival is checked against the current contents of its target bucket, and the resulting insertions and deletions create a sequential dependence between the arrivals to a bucket. Our GPU exact pricing removes this dependence in three steps.

1. *Batched dominance.* Labels enter a bucket in waves. Each wave passes three dominance stages: stored labels against candidates, candidates against earlier candidates in a canonical order, and the surviving candidates against the stored labels. The stages run in sequence and the comparisons within a stage in parallel; every deletion keeps a direct dominator that survives the wave, with conservative floating-point guards and a repair pass that restores any candidate left without a surviving direct dominator.
2. *Labeling redesigned around the test.* RouteOpt's push of one label at a time into the next bucket becomes a level-synchronous, wave-based expansion in which the arrivals to a bucket are batched instead of chained, the form parallel CPU labeling also takes (Petersen & Spoorendonk, 2025): resource levels are processed in order, waves repeat within a level until its frontier is empty, and all label state is in structure-of-arrays form. Label extension, dominance testing, and the join of half-routes run on the GPU; route reconstruction, deduplication, and the final column selection run on the CPU, the heuristic levels stay on the CPU, and dispatch sits inside the exact level, so the CG loop is unchanged.
3. *Memory and filtering.* Shared-memory tiles reuse candidate prefixes across comparisons, and the cross-bucket filter runs one warp per frontier label and stops at a reduced-cost prefix bound.

Bucket-graph labeling is the exact pricer of state-of-the-art vehicle-routing solvers, among them VRPSolver and RouteOpt (Sadykov et al., 2021; Pessoa et al., 2020; You & Yang, 2026). They share the label-comparison structure the three steps act on, whose dominance test is the cost the bucket graph exists to contain (Sadykov et al., 2021), so the steps carry over once the resource extension,

the dominance rule, and the join are adapted. Appendices C.3 and C.4 give the implementation details and the safeguards behind the optimality guarantee.

4 EXPERIMENTS

We organize the empirical evaluation around three questions:

(i) **How the pipeline works.** Which component carries the gain, how the components interact, and where the prefilled columns go during column generation.
(ii) **What GPU pricing can add.** How much a batched exact pricer buys per call, on which GPU, and how that converts into root time once the cost has shifted to exact pricing.
(iii) **How general the recipe is.** Whether prefill and the learned center carry over to a different problem inside a different solver, as a case study on parallel-machine scheduling.

Training, model selection and the constants of the pipeline are in Appendix B, the scheduling variants in Appendix A.5; the ablations that isolate the two learned components are in Appendix D.

4.1 EXPERIMENTAL SETUP

All experiments ran on one workstation with an NVIDIA RTX PRO 6000 Blackwell GPU and an AMD EPYC 9555 processor; only the per-call device comparison of Table 2 and Appendix C.5 adds an L4 and a B200. For CVRP the baseline is RouteOpt (You & Yang, 2026), the state-of-the-art exact solver, whose reach marks the frontier of instances solved to optimality; for scheduling it is the BPC algorithm of Hijazi et al. (2026), which accelerates pricing with a pointer transformer. On CVRP we focus on the region exact solvers have not yet conquered, instances with 300 customers and more. Route size and instance size are the two factors that decide difficulty, so the panel is stratified by both: on Vrp-Set-X (Uchoa et al., 2017) we group instances by route size with the XML classification (Table 13) and sample five instances with 300 to 1000 customers from each of the short, medium, long and very-long classes; on XML (Queiroga et al., 2022) we sample five instances per class from short to ultra-long at 300, 400 and 500 customers; 95 instances in all. For scheduling we use the standard benchmark distribution of Hijazi et al. (2026). The measured quantity is the wall clock of root pricing; every completed arm reaches the same certified root bound, each configuration is normalized to the mean of two runs of RouteOpt (*vanilla*) on the same instance, and ratios are aggregated as instance-equal geometric means.

4.2 THE PIPELINE TAKEN APART

Table 1 compares five configurations with vanilla RouteOpt on the panel: the learned archive alone (*Prefill*), the learned center alone (*Center*), the GPU backend alone on the vanilla solver (*GPU*), archive and center together, and the full pipeline. Prefill is the central idea and the largest lever. On its own it brings root pricing to 26.1 % of vanilla's, faster on all instances, and does so on every benchmark set and every route-size class of the panel. In instrumented runs the archive supplied columns in 24 to 52 % of the LIGHT rounds, bypassing labeling in those rounds, so it acts as a fast alternative to the LIGHT level; when it supplies no admissible column, the solver resumes labeling (Appendix B.1.2). Appendix D.1 compares the lazy archive with preloading, with a heuristic pool in place of the learned one, and with two oracles that preload the converged column set or its support. The center is designed to correct what prefill can do to the early dual trajectory; alone it is a smaller lever, bringing root pricing to 61.6 % of vanilla's. Prefill+Center matches Prefill on the panel average (0.264 against 0.261) because the center acts mainly on the hard instances, the large ones with long routes: on the ultra-long rows of every XML set it is clearly faster than Prefill alone, on long and very-long routes the two are even, and on short and medium routes, where prefill leaves little to correct, it costs a little. The controls for the center, standard smoothing and a permuted direction, are in Appendix D.2. Figure 2 shows the mechanism on the largest instance. The archive's columns take a sizable share of the master's pool, of the LP solution weight $\sum x$, and of the basis, so they do real work, and the pricer replaces them only gradually. With the learned center the HEAVY iterations fall while the demand for exact pricing does not, so its share of the run rises, which is what GPU pricing, the third component, addresses in the next subsection.

Table 1: Component ablation on the 95-instance panel: geometric-mean root-time ratio to vanilla (lower is better), with the number of instances faster than vanilla, out of the cell's instances, in parentheses.

Set	Class	Inst.	Prefill	Center	GPU	Prefill+Center	Full pipeline
X	short	5	0.149 (5/5)	0.686 (5/5)	1.002 (3/5)	0.193 (5/5)	0.125 (5/5)
X	medium	5	0.154 (5/5)	0.671 (5/5)	0.925 (5/5)	0.173 (5/5)	0.100 (5/5)
X	long	5	0.269 (5/5)	0.587 (5/5)	0.886 (5/5)	0.258 (5/5)	0.101 (5/5)
X	very long	5	0.318 (5/5)	0.689 (5/5)	0.870 (5/5)	0.292 (5/5)	0.133 (5/5)
X	all	20	0.210 (20/20)	0.657 (20/20)	0.919 (18/20)	0.224 (20/20)	0.114 (20/20)
XML300	short	5	0.203 (5/5)	0.710 (5/5)	0.969 (5/5)	0.234 (5/5)	0.136 (5/5)
XML300	medium	5	0.282 (5/5)	0.677 (5/5)	0.939 (5/5)	0.310 (5/5)	0.141 (5/5)
XML300	long	5	0.270 (5/5)	0.690 (5/5)	0.954 (4/5)	0.274 (5/5)	0.179 (5/5)
XML300	very long	5	0.452 (5/5)	0.642 (5/5)	0.865 (5/5)	0.421 (5/5)	0.208 (5/5)
XML300	ultra long	5	0.392 (5/5)	0.329 (5/5)	0.742 (4/5)	0.296 (5/5)	0.150 (5/5)
XML300	all	25	0.307 (25/25)	0.588 (25/25)	0.890 (23/25)	0.301 (25/25)	0.161 (25/25)
XML400	short	5	0.179 (5/5)	0.717 (5/5)	0.955 (5/5)	0.209 (5/5)	0.117 (5/5)
XML400	medium	5	0.237 (5/5)	0.660 (5/5)	0.931 (4/5)	0.238 (5/5)	0.115 (5/5)
XML400	long	5	0.300 (5/5)	0.668 (5/5)	0.932 (4/5)	0.275 (5/5)	0.131 (5/5)
XML400	very long	5	0.369 (5/5)	0.660 (4/5)	0.848 (5/5)	0.373 (4/5)	0.170 (5/5)
XML400	ultra long	5	0.465 (5/5)	0.493 (5/5)	0.734 (5/5)	0.391 (5/5)	0.169 (5/5)
XML400	all	25	0.293 (25/25)	0.634 (24/25)	0.876 (23/25)	0.288 (24/25)	0.138 (25/25)
XML500	short	5	0.153 (5/5)	0.739 (5/5)	0.941 (5/5)	0.182 (5/5)	0.082 (5/5)
XML500	medium	5	0.165 (5/5)	0.648 (5/5)	0.938 (4/5)	0.168 (5/5)	0.103 (5/5)
XML500	long	5	0.231 (5/5)	0.668 (5/5)	0.955 (5/5)	0.236 (5/5)	0.104 (5/5)
XML500	very long	5	0.336 (5/5)	0.637 (5/5)	0.902 (5/5)	0.395 (5/5)	0.159 (5/5)
XML500	ultra long	5	0.368 (5/5)	0.371 (5/5)	0.616 (5/5)	0.283 (5/5)	0.148 (5/5)
XML500	all	25	0.235 (25/25)	0.597 (25/25)	0.859 (24/25)	0.241 (25/25)	0.116 (25/25)
all		95	0.261 (95/95)	0.616 (94/95)	0.884 (88/95)	0.264 (94/95)	0.132 (95/95)

Table 2: Median seconds per steady-state exact call on X-n1001-k43 (learned stack), one CPU thread against three GPUs; same bound and call count on every card.

	CPU (one thread)	L4	RTX PRO 6000	B200
Seconds per exact call	15.84	4.02	1.30	1.21

Table 3: Parallel-machine scheduling inside the code of Hijazi et al. (2026): geometric-mean wall time in seconds with the number of instances certified within $1800\,\mathrm{s}$ in parentheses; m machines, n jobs, five instances per cell; a star marks a cell with a timed-out run, which enters the mean at $1800\,\mathrm{s}$. *Wentges* is smoothing alone.

	$m4\,n40$	$m4\,n100$	$m8\,n40$	$m12\,n100$	$m20\,n100$	all 25
Their CG	3.1 (5)	1800* (0)	1.4 (5)	403.4 (5)	197.8 (5)	57.1 (20)
Their transformer pricing	17.7 (5)	1800* (0)	12.2 (5)	866.4 (5)	434.6 (5)	171.2 (20)
Prefill	1.3 (5)	1800* (0)	0.2 (5)	6.6 (5)	1.8 (5)	5.4 (20)
Wentges	1.6 (5)	1489* (2)	1.1 (5)	275.7 (5)	154.1 (5)	40.9 (22)
Wentges + Prefill	0.6 (5)	1047* (4)	0.2 (5)	7.6 (5)	2.9 (5)	5.1 (24)
Wentges + Prefill + Center	0.7 (5)	628 (5)	0.2 (5)	7.6 (5)	3.3 (5)	4.8 (25)

4.3 GPU EXACT PRICING

Prefill and the center change where root pricing spends its time: exact pricing becomes the largest stage of the run once archive and center are in place, and it is where a further lever pays. Table 2 shows what this buys per exact call on X-n1001-k43, the largest instance, measured as pricing time alone without the LP solves. Over the ten X instances with the longest exact pricing time (Appendix C.5), the median ratio of CPU to GPU median call time is 4.70 on a low-power L4 and 8.90 on the Blackwell card we deploy, with identical bounds on every GPU and identical exact-call counts across the three cards. Because exact pricing dominates the profile once prefill and center are in place, the per-call gain converts into the full-stack result: the backend alone on vanilla RouteOpt gives a root-time ratio of 0.884; within the learned stack the GPU-to-CPU ratio is 0.500 on the same 95 instances; and the full pipeline is 7.6 times faster than vanilla RouteOpt on average and 17.5 times faster in the best case.

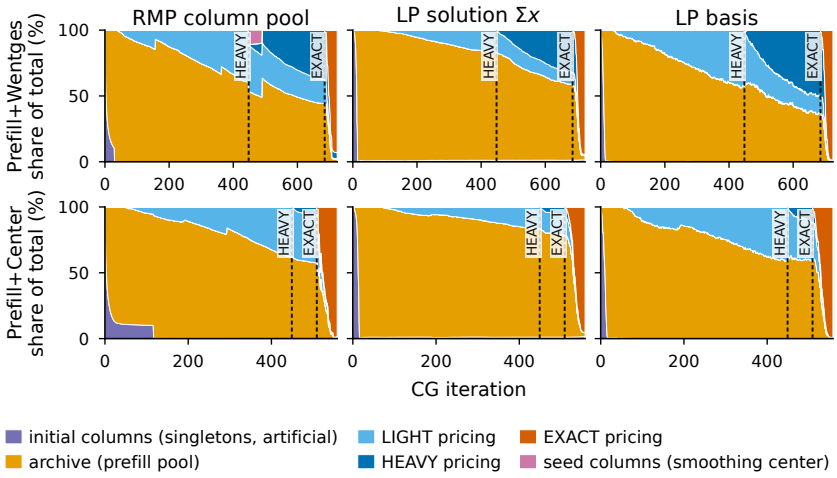

Figure 2: Share of columns by column origin over the root's iterations on X-n1001-k43, for Prefill+Wentges (top) and Prefill+Center (bottom); dashed lines mark the pricer changes, LIGHT to HEAVY to EXACT.

4.4 Case study: parallel-machine scheduling

The scheduling oracle is a dense dynamic program, so this case study evaluates prefill and dual smoothing without porting the routing GPU backend. We run inside the BPC solver of Hijazi et al. (2026), on their instances and seeds, with a vectorized exact pricing oracle shared by every arm so that the comparison is about column generation and not about the speed of one implementation (Appendix A.5); prefill here preloads the RMP with schedules from randomized list scheduling, without the neural scorer, since the pricing subproblem of this problem is not hard. Table 3 reports geometric-mean wall time with the number of instances certified within the time limit. Prefill alone is already strong on this problem: it certifies the same instances as the baseline column generation in a small fraction of the time. Wentges smoothing helps on its own, smoothing with prefill certifies all but one instance, and adding the learned center certifies all 25 instances, where the learned pricer certifies 20; the whole configuration is 12 times faster in geometric mean than the baseline column generation. Changing how column generation is started and steered is therefore a lever of great power on this problem, one that acts on the initialization and the duals rather than on the pricer.

5 Conclusion

This paper breaks with the folklore that initial columns harm column generation and turns prefilling into a general plug-in accelerator for BPC: learning to prefill the columns. The idea at its center is that imperfect columns, routes that need not belong to any optimal solution but together cover the customers well, are what speeds the root up, provided they are held outside the master as a lazy archive and served at the current smoothed dual. We built a learn-to-prefill pipeline on this idea and obtained large speed-ups on two problems inside two exact solvers, CVRP in RouteOpt and parallel-machine scheduling in the code of Hijazi et al. (2026). On the hardest instances prefill also showed its weakness: the dual it produces is close to the optimal dual in value but deviates from it in direction, so the query needs a correction of direction rather than of distance, and we designed a learned dual center for that purpose. Once archive and center are in place, the remaining bottleneck of the pipeline is exact pricing, and we rebuilt it around its dominance test for the GPU. On the state-of-the-art CVRP solver the complete pipeline reaches the certified root bound with a 7.6-fold geometric-mean speed-up of root pricing over vanilla RouteOpt, and in the best case in 5.7 % of vanilla RouteOpt's time.

The main limitation is that the effect on the full BPC tree is unknown: at the scale we test, the instances have no known exact solution time, and root pricing is the part we can measure and certify; extending the measurement to the tree is one direction of future work. A second direction is the co-design of the two learned components, which we designed independently, either as a joint configuration or as a single model that predicts pool and center together.

AI USE STATEMENT

Generative AI tools were used to help draft and edit the text of this paper. All other parts of this work, including the ideas, the methods, the implementation and the experiments, are the authors' own. The authors reviewed the drafted text and take responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

The arms and the measurement rules are stated in Section 4.1; the hyper-parameters and constants of the learned components are listed in Appendix B, the GPU backend is described in Appendix C, the ablation protocol in Appendix D, and the software versions and seeds in Appendix E. The code, the trained models and the measurement scripts will be released upon acceptance.

REFERENCES

Behrouz Babaki, Laurent Charlin, and Sanjay Dominik Jena. COIL: A deep architecture for column generation. Technical Report CIRRELT-2022-25, CIRRELT, September 2022.

Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008.

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011.

Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

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.

Cheng Chi, Amine Mohamed Aboussalah, Elias B. Khalil, Juyoung Wang, and Zoha Sherkat-Masoumi. A deep reinforcement learning framework for column generation. In *Advances in Neural Information Processing Systems*, volume 35, pp. 9633–9644, 2022.

Luciano Costa, Claudio Contardo, and Guy Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4):946–985, 2019.

George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

Arnaud Deza and Elias B. Khalil. Machine learning for cutting planes in integer programming: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 6592–6600, 2023.

Olivier du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1–3):229–237, 1999.

Lichang Fang, Haofeng Yuan, Shiji Song, and Bokui Chen. Learning to stabilize column generation. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2025.

Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106:491–511, 2006.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Amira Hijazi, Osman Y. Özaltın, and Reha Uzsoy. An improving column is all you need: Enhancing column generation for parallel machine scheduling via transformers. *INFORMS Journal on Computing*, 2026.

Stefan Irnich, Guy Desaulniers, Jacques Desrosiers, and Ahmed Hadjar. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, 22(2):297–313, 2010.

Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 724–731, 2016.

Pavlína Koutecká, Přemysl Šůcha, Jan Hůla, and Broos Maenhout. A machine learning approach to rank pricing problems in branch-and-price. *European Journal of Operational Research*, 320(2): 328–342, 2025.

Sebastian Kraul, Markus Seizinger, and Jens O. Brunner. Machine learning-supported prediction of dual variables for the cutting stock problem with an application in stabilized column generation. *INFORMS Journal on Computing*, 35(3):692–709, 2023.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

Haihao Lu and Jinwen Yang. cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia. *Operations Research*, 73(6):3440–3452, 2025.

Shengliang Lu, Bingsheng He, Yuchen Li, and Hao Fu. Accelerating exact constrained shortest paths on GPUs. *Proceedings of the VLDB Endowment*, 14(4):547–559, 2020.

Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Machine-learning-based column selection for column generation. *Transportation Science*, 55(4):815–831, 2021.

Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Machine-learning-based arc selection for constrained shortest path problems in column generation. *INFORMS Journal on Optimization*, 5(2): 191–210, 2023.

Max B. Paulus, Giulia Zarpellon, Andreas Krause, Laurent Charlin, and Chris J. Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17584–17600, 2022.

Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017a.

Diego Pecin, Artur Pessoa, Marcus Poggi, Eduardo Uchoa, and Haroldo Santos. Limited memory rank-1 cuts for vehicle routing problems. *Operations Research Letters*, 45(3):206–209, 2017b.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360, 2018.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183(1–2):483–523, 2020.

Bjørn Petersen and Simon Spoorendonk. A parallel pull labelling algorithm for the resource constrained shortest path problem. *arXiv preprint arXiv:2511.01397*, 2025.

Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

Eduardo Queiroga, Ruslan Sadykov, Eduardo Uchoa, and Thibaut Vidal. 10,000 optimal CVRP solutions for testing machine learning based heuristics. In *AAAI-22 Workshop on Machine Learning for Operations Research (ML4OR)*, 2022.

Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa. A bucket graph-based labeling algorithm with application to vehicle routing. *Transportation Science*, 55(1):4–28, 2021.

Puja Sarkar, Vivekanand B. Khanapuri, and Manoj Kumar Tiwari. Accelerating the stabilized column generation using machine learning. *Computers & Industrial Engineering*, 200:110837, 2025.

Yunzhuang Shen, Yuan Sun, Xiaodong Li, Andrew Eberhard, and Andreas Ernst. Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 9926–9934, 2022.

Yunzhuang Shen, Yuan Sun, Xiaodong Li, Zhiguang Cao, Andrew Eberhard, and Guangquan Zhang. Adaptive stabilization based on machine learning for column generation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 44741–44758, 2024.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9367–9376, 2020.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

Roman Václavík, Antonín Novák, Přemysl Šůcha, and Zdeněk Hanzálek. Accelerating the branch-and-price algorithm using machine learning. *European Journal of Operational Research*, 271(3): 1055–1069, 2018.

J. M. van den Akker, J. A. Hoogeveen, and S. L. van de Velde. Parallel machine scheduling by column generation. *Operations Research*, 47(6):862–872, 1999.

François Vanderbeck. Implementing mixed integer column generation. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon (eds.), *Column Generation*, pp. 331–358. Springer, 2005.

Thibaut Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

Olivia Wang and Reem Khir. Learning to control stabilization in column generation, 2026. arXiv preprint arXiv:2604.23889.

Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *International Conference on Learning Representations (ICLR)*, 2023.

Paul Wentges. Weighted Dantzig–Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.

Kuan Xu, Li Shen, and Lindong Liu. Enhancing column generation by reinforcement learning-based hyper-heuristic for vehicle routing and scheduling problems. *Computers & Industrial Engineering*, 206:111138, 2025.

Zhengzhong You and Yu Yang. RouteOpt: An open-source modular exact solver for vehicle routing problems. *INFORMS Journal on Computing*, 2026. Articles in advance.

Zhengzhong You, Yu Yang, Xinshang Wang, and Wotao Yin. Two-stage learning to branch in branch-price-and-cut algorithms for solving vehicle routing problems exactly. *Operations Research*, 74(4):1811–1833, 2026.

Yang Yu, Xiaolong Li, Roberto Baldacci, Zhiqiao Wu, Wei Sun, Jiafu Tang, and Han Zhu. An exact branch-price-and-cut algorithm for the unrelated parallel machine scheduling problem. *INFORMS Journal on Computing*, 2026.

CONTENTS OF THE APPENDIX

A Branch-price-and-cut in detail **13**

A.1 Dantzig–Wolfe decomposition and the master problem 13

A.2 Column generation: the primal–dual view 14

A.3 Two typical tasks: routing and scheduling 16

A.4 Case 1: capacitated vehicle routing in RouteOpt 16

A.5 Case 2: unrelated parallel machine scheduling 17

B The learning components in detail **18**

B.1 Learning to prefill 18

B.2 Learning the dual center 20

B.3 Labels and data 21

C GPU exact pricing in detail **22**

C.1 Bucket-graph labeling and the dominance test 22

C.2 Where the time goes 22

C.3 Removing the sequential dependence 22

C.4 Certification and safeguards 23

C.5 What the backend buys per call 23

D Ablations and additional results **24**

D.1 Supplying the columns: which pool, and how 24

D.2 Choosing the center: the learned direction 25

D.3 Very short routes 25

E Reproducibility **26**

A BRANCH-PRICE-AND-CUT IN DETAIL

This appendix defines the routing and scheduling masters, their pricing problems and the associated bounds: the master problem obtained by Dantzig–Wolfe decomposition, the primal–dual relations that make pricing a separation problem and give every dual point a bound, and the two pricing problems on which the paper's components are evaluated. Throughout, $v = (u, \sigma)$ denotes the full dual vector of a master, u the block of multipliers on the covering rows and σ the multiplier of the resource-count row; where that row is absent, σ is omitted.

A.1 DANTZIG–WOLFE DECOMPOSITION AND THE MASTER PROBLEM

Let the compact model have K blocks coupled by linking constraints,

$$z_{\mathrm{IP}} = \min \Big\{ \sum_{k=1}^{K} c_k^\top x_k : \sum_{k=1}^{K} A_k x_k = b,\ \ x_k \in X_k \ (k = 1, \dots, K) \Big\}, \tag{A.1}$$

where $X_k = \{x \in \mathbb{Z}^{n_k}_{\geq 0} : \ D_k x \leq d_k\}$ is bounded for every k. Write $X_k = \{x_{k,r} : \ r \in R_k\}$ for the finite point set of block k. Every $x_k \in \mathrm{conv}(X_k)$ is a convex combination $x_k = \sum_{r \in R_k} \lambda_{k,r} x_{k,r}$ with $\sum_{r \in R_k} \lambda_{k,r} = 1$ and $\lambda_k \geq 0$ (Dantzig & Wolfe, 1960). Substituting into (A.1) gives the *master*

LP

$$z_{\mathrm{DW}} = \min_{\lambda \geq 0} \Big\{ \sum_{k} \sum_{r \in R_k} c_{k,r} \lambda_{k,r} : \sum_{k} \sum_{r \in R_k} a_{k,r} \lambda_{k,r} = b, \ \sum_{r \in R_k} \lambda_{k,r} = 1 \ (k = 1, \ldots, K) \Big\}, \tag{A.2}$$

with $c_{k,r} = c_k^\top x_{k,r}$ and $a_{k,r} = A_k x_{k,r}$. Requiring in addition $\sum_r \lambda_{k,r} x_{k,r} \in X_k$, which for the set-partitioning structures below is the integrality of λ, gives the integer master, whose optimum is z_{IP}.

Proposition A.1 (Dantzig–Wolfe bound) *Let z_{LP} be the value of* (A.1) *with each X_k replaced by $\{x \geq 0 : \ D_k x \leq d_k\}$, and let $z_{\mathrm{L}} = \max_{\mu} \big[\mu^\top b + \sum_k \min\{(c_k - A_k^\top \mu)^\top x_k : \ x_k \in X_k\} \big]$ be the Lagrangian dual of the linking constraints. Then $z_{\mathrm{LP}} \leq z_{\mathrm{DW}} = z_{\mathrm{L}} \leq z_{\mathrm{IP}}$, and $z_{\mathrm{LP}} = z_{\mathrm{DW}}$ whenever every polyhedron $\{x \geq 0 : \ D_k x \leq d_k\}$ has integral vertices.*

Proof. Problem (A.2) minimizes the objective of (A.1) over $\prod_k \mathrm{conv}(X_k)$, which is contained in $\prod_k \{x \geq 0 : \ D_k x \leq d_k\}$ and equals it when the vertices are integral; this gives the first inequality and the equality case. LP duality applied to (A.2), with μ the multipliers of the linking rows and the convexity rows priced out block by block, gives $z_{\mathrm{DW}} = z_{\mathrm{L}}$, and $z_{\mathrm{L}} \leq z_{\mathrm{IP}}$ is weak Lagrangian duality (Lübbecke & Desrosiers, 2005). □

Aggregation and the routing master. Aggregating K identical blocks, as the vehicles of one fleet are, drops the block index, makes a column the plan of one anonymous block, and turns the K convexity rows into a single equation counting the selected plans. For the CVRP with a free fleet, RouteOpt instead uses the valid vehicle-count inequality $\sum_r \lambda_r \geq K_{\min}$, with $K_{\min}$ a lower bound on the number of vehicles; adding this row to (1) gives the following master and its dual,

$$\begin{aligned} z_{\mathrm{DW}} \ &= \ \min_{\lambda \geq 0} \Big\{ \sum_{r \in \mathcal{R}} c_r \lambda_r : \ \sum_{r \in \mathcal{R}} a_{ir} \lambda_r = 1 \ (i \in C), \ \sum_{r \in \mathcal{R}} \lambda_r \geq K_{\min} \Big\} \\ &= \ \max_{u \in \mathbb{R}^n, \ \sigma \geq 0} \Big\{ \sum_{i \in C} u_i + K_{\min} \sigma : \ rc_r(v) = c_r - a_r^\top u - \sigma \ \geq \ 0, \ \ r \in \mathcal{R} \Big\}, \end{aligned} \tag{A.3}$$

where $\mathcal{R}$ is the set of ng-feasible routes of Appendix A.4, c_r the length of route r and a_{ir} its number of visits to customer i. Unrelated machines are not copies of one another, and the scheduling master of Appendix A.5 keeps one row per machine.

A.2 Column generation: the primal–dual view

Restricted master and pricing. For a route set $T \subseteq \mathcal{R}$ containing all singleton routes, let $\mathrm{RMP}(T)$ be (A.3) with $\mathcal{R}$ replaced by T, let z_T be its value, λ_T a basic optimal solution and $v_T = (u_T, \sigma_T)$ the associated basic optimal dual. Extending the notation of (3) to the vehicle row, let

$$D_T = \{(u, \sigma) \in \mathbb{R}^n \times \mathbb{R}_{\geq 0} : \ a_r^\top u + \sigma \leq c_r, \ r \in T\}, \qquad F_T = \arg \max_{(u,\sigma) \in D_T} \Big\{ \sum_{i \in C} u_i + K_{\min} \sigma \Big\}, \tag{A.4}$$

so that v_T is a vertex of F_T and $D_{\mathcal{R}}$ is the feasible set of the full dual. The *pricing problem* at a dual point v is

$$m(v) \ = \ \min_{r \in \mathcal{R}} \ rc_r(v), \qquad rc_r(v) = c_r - a_r^\top u - \sigma, \tag{A.5}$$

the separation problem of $D_{\mathcal{R}}$: a route with $rc_r(v) < 0$ is a violated dual constraint. Column generation iterates: solve $\mathrm{RMP}(T)$, solve (A.5) at v_T, stop if $m(v_T) \geq 0$, otherwise add a batch of routes with negative reduced cost to T (Barnhart et al., 1998; Lübbecke & Desrosiers, 2005).

Proposition A.2 (Optimality) (i) *If $m(v_T) \geq 0$, then λ_T extended by zeros is optimal for* (A.3) *and $z_T = z_{\mathrm{DW}}$.* (ii) *If $z_T = z_{\mathrm{DW}}$, then $F_T \cap D_{\mathcal{R}} \neq \emptyset$; the vertex v_T returned by the simplex method need not lie in $D_{\mathcal{R}}$.*

Proof. (i) v_T is feasible for the dual of (A.3) with objective z_T, the value of the feasible primal solution λ_T, so both are optimal by weak duality. (ii) An optimal dual v^* of the full master is feasible for the dual of $\mathrm{RMP}(T)$ with objective $z_{\mathrm{DW}} = z_T$, hence $v^* \in F_T \cap D_{\mathcal{R}}$; F_T may nevertheless have vertices outside $D_{\mathcal{R}}$, which is the situation discussed after (3). □

Proposition A.3 (Lagrangian bound) *Let κ bound $\sum_r \lambda_r$ over the feasible solutions of* (A.3)*. For every $u \in \mathbb{R}^n$ and $\sigma \geq 0$,*

$$z_{\mathrm{DW}} \;\geq\; L(v) \;=\; \sum_{i \in C} u_i + K_{\min}\sigma + \kappa\, \min\{0,\, m(v)\}. \tag{A.6}$$

Proof. For a feasible λ write $s = \sum_r \lambda_r$, so that $K_{\min} \leq s \leq \kappa$. Since $\sum_r a_{ir}\lambda_r = 1$ for every i and $rc_r(v) \geq m(v)$,

$$\begin{aligned}\sum_r c_r\lambda_r = \sum_i u_i \sum_r a_{ir}\lambda_r + \sigma \sum_r \lambda_r + \sum_r rc_r(v)\lambda_r \;&=\; \sum_i u_i + \sigma s + \sum_r rc_r(v)\lambda_r \\ \geq \sum_i u_i + K_{\min}\sigma + \min\{0, m(v)\}\, s \;&\geq\; \sum_i u_i + K_{\min}\sigma + \kappa\, \min\{0, m(v)\} = L(v),\end{aligned}$$

using $\sigma \geq 0$ in the first inequality and $\min\{0, m(v)\} \leq 0$ in the second. □

Every route covers at least one customer and every customer is covered once, so $\kappa = n$ and (A.6) is the bound L of (6). The bound is computed by the exact pricing call that finds the columns: the running maximum of the certified bounds is nondecreasing, z_T is nonincreasing in T, and the two meet when $m(v_T) \geq 0$. Every dual point has a bound, feasible or not, which is what allows a stabilization center to be selected by its bound.

Degeneracy and smoothing. The face F_T is in general not a singleton, and successive v_T can change substantially while z_T barely moves, so that the tail of column generation stretches (Lübbecke & Desrosiers, 2005; du Merle et al., 1999). Wentges smoothing (Wentges, 1997) prices at the query $q_t = (1 - \alpha_t)v_t + \alpha_t c_t$, with $c_t = (c_{t,C}, c_{t,\mathrm{veh}})$ the center and $\alpha_t \in [0, 1)$ the weight, which extends (2) to the vehicle coordinate, and admits a returned route r only if $rc_r(v_t) < -\epsilon$ at the raw dual. An exact smoothed query that returns no route admissible at v_t is a *mis-price*; it triggers the next rung of the ladder

$$\alpha_{(j)} = \max\{0,\; 1 - (j+1)(1 - \alpha_t)\}, \qquad j = 0, 1, \ldots, \tag{A.7}$$

Table 4: Constants of the smoothing rule, RouteOpt's defaults, used unchanged in every configuration: the initial weight, the weight step and cap of (10), the pricing tolerance of the admission test, and the tolerances of (9) and (10).

constant	α_0	η	$\bar{\alpha}$	ϵ	ϵ_L	ϵ_g
value	0.5	0.1	0.9	10^{-5}	10^{-6}	10^{-9}

and termination requires a complete exact call at $\alpha_{(j)} = 0$ that returns no route with $rc_r(v_t) < -\epsilon$, which certifies $m(v_t) \geq -\epsilon$ (Pessoa et al., 2018). At the end of each exact-pricing sequence, the best complete query determines the center update of (9), the center being replaced only when its certified bound improves by more than ϵ_L, and the last complete query determines the weight update of (10). Because admission uses the raw dual, smoothing changes the sequence T and the time, never z_{DW}.

Heuristic and exact pricing. Pricing is solved by dynamic programming over labels, and labels are discarded under a dominance relation $\preceq$ (Appendix A.4). The exact level uses a relation $\preceq_{\mathrm{E}}$ under which every completion of a discarded label is matched by a completion of its dominator with no larger reduced cost, so that it returns a minimizer of (A.5); the heuristic levels use coarser relations $\preceq_{\mathrm{E}} \subseteq \preceq_{\mathrm{H}} \subseteq \preceq_{\mathrm{L}}$ that compare fewer label components, discard more labels and may therefore miss the minimizer. A heuristic call returns a subset of the routes with $rc_r(v) < 0$ and can never certify $m(v) \geq 0$; termination requires an exact call. *Root pricing*, the quantity measured in this paper, is the first column generation from the initial columns to the exact certificate, before any cut is separated.

Cuts and branching. After the initial cut-free root column generation, root cut rounds strengthen the bound with valid inequalities and branching enforces integrality, both imposed inside the pricing problem. Cuts on the edge flow $x_e = \sum_r b_{er}\lambda_r$, such as rounded capacity cuts, are *robust*: their multipliers modify the edge costs and leave the structure of pricing unchanged (Fukasawa et al., 2006). Cuts on the columns, such as subset-row and limited-memory rank-1 cuts, are *non-robust*:

each adds a state component to the labels and modifies the dominance test (Jepsen et al., 2008; Pecin et al., 2017a). Edge branching is enforced by deleting forbidden arcs or by adding master rows whose dual multipliers modify the pricing costs; reduced-cost arc elimination (Irnich et al., 2010) and route enumeration (Baldacci et al., 2008) complete the toolbox surveyed by Costa et al. (2019) and assembled in generic form by Pessoa et al. (2020). Subsequent cut rounds and branching nodes reuse the columns generated by the initial root column generation, which is the quantity measured in this paper.

A.3 Two typical tasks: routing and scheduling

In both families a column is the plan of one resource, a vehicle's route or a machine's schedule, and the master covers a set of demands, customers or jobs. The pricing problems differ in kind: routing prices by a resource-constrained shortest path over a graph, relaxed from elementarity by ng-routes and solved by bucket-graph labeling with a dominance relation (Baldacci et al., 2011; Sadykov et al., 2021; Pessoa et al., 2020); the scheduling formulation used here prices by a subset-selection dynamic program over jobs whose state is the accumulated processing time (van den Akker et al., 1999; Yu et al., 2026). The learned components of this paper touch neither pricing problem; they generate column pools and initialize stabilization centers. The two cases below fix what a column, a dual and a pricing call are in each family.

A.4 Case 1: capacitated vehicle routing in RouteOpt

Instance and routes. Let $V = C \cup \{0\}$ with $|C| = n$, symmetric distances d_{ij}, demands q_i and capacity Q. A route $r = (0, i_1, \dots, i_k, 0)$ visits customers $i_1, \dots, i_k$, has load $q(r) = \sum_h q_{i_h} \le Q$ and cost $c_r = \sum_{(i,j)\in r} d_{ij}$. A vehicle-indexed formulation separates the route-feasibility constraints by vehicle and has the block structure of (A.1); the master (A.3) uses anonymous ng-route columns, defined next, and the valid vehicle-count inequality (Fukasawa et al., 2006; Pecin et al., 2017a).

ng-routes. Each customer i carries a neighborhood $N_i \subseteq C$ with $i \in N_i$ and $|N_i| = \Delta$ ($\Delta = 8$ in the deployed configuration). Along a route the *ng memory* is defined by

$$\Pi_1 = \{i_1\}, \qquad \Pi_{h+1} = \big(\Pi_h \cap N_{i_{h+1}}\big) \cup \{i_{h+1}\}, \tag{A.8}$$

and r is an *ng-route* if $i_{h+1} \notin \Pi_h$ for every h: a customer may be revisited only after the route has passed through a customer whose neighborhood does not contain it (Baldacci et al., 2011). Every elementary route is an ng-route, and the ng-routes coincide with the elementary routes when $N_i = C$ for all i, so enlarging the neighborhoods strengthens the relaxation; heuristic columns from a metaheuristic are elementary and hence valid columns of (A.3).

Labeling. Put $u_0 = 0$ and $\bar{d}_{ij} = d_{ij} - u_j$, so that $rc_r(v) = \sum_{(i,j)\in r} \bar{d}_{ij} - \sigma$. A *label* is a tuple $L = (i,\, q,\, \Pi,\, \bar{c})$: the last customer i of a partial route from the depot, its load q, its ng memory Π and its accumulated cost $\bar{c} = \sum \bar{d}$ along the partial route. The initial label is $(0, 0, \emptyset, 0)$; the extension of L along arc (i, j) is feasible if $j \notin \Pi$ and $q + q_j \le Q$, and produces

$$L' = \big(j,\; q + q_j,\; (\Pi \cap N_j) \cup \{j\},\; \bar{c} + \bar{d}_{ij}\big); \tag{A.9}$$

a label at i closes to the route of reduced cost $\bar{c} + d_{i0} - \sigma$, and $m(v)$ is the minimum over closed labels. Under symmetric distances RouteOpt forms routes by joining a forward partial route with the reversal of another forward partial route over an edge (i, j): with the label definitions above, the join of L_1 at i and L_2 at j is feasible if $q_1 + q_2 \le Q$ and the concatenated route satisfies the ng condition, and has reduced cost $\bar{c}_1 + \bar{c}_2 + d_{ij} - \sigma$.

Dominance. For two labels at the same customer, $L_1 \preceq_{\mathrm{E}} L_2$ if

$$\bar{c}_1 \le \bar{c}_2, \qquad q_1 \le q_2, \qquad \Pi_1 \subseteq \Pi_2. \tag{A.10}$$

Proposition A.4 (Dominance) *If $L_1 \preceq_{\mathrm{E}} L_2$, then for every feasible completion of L_2 the same completion is feasible for L_1 and yields a route with no larger reduced cost; hence L_2 can be discarded without changing $m(v)$.*

Proof. Feasibility of an extension requires $j \notin \Pi$ and $q + q_j \le Q$, both monotone in (q, Π); the memory update (A.8) preserves $\Pi_1 \subseteq \Pi_2$ and the load update preserves $q_1 \le q_2$, so the conditions hold along the whole completion, and the cost difference $\bar{c}_1 - \bar{c}_2 \le 0$ is carried unchanged. The same argument applies to the join. □

In the cut-free root comparisons, exact pricing uses (A.10). With non-robust cuts, dominance also accounts for their states and dual contributions; the heuristic levels use the coarser relations described in Appendix A.2.

Bucket graph. Fix a load step Δ_q and let bucket $B(i,k)$ hold the labels at customer i with load in $[k\Delta_q, (k+1)\Delta_q)$. Buckets are processed in increasing k; a label in $B(i,k)$ can only be dominated by labels in $B(i,k')$ with $k' \le k$, so the dominance check for a new label is restricted to the current bucket and, through a lower bound on the reduced cost of the earlier buckets of the same customer, to a cross-bucket test; the extension (A.9) moves a label along a *bucket arc* from (i,k) to (j,k') with k' the bucket of $q + q_j$, and *jump arcs* advance the resource to a later bucket of the same vertex before traversing a retained outgoing arc (Sadykov et al., 2021). In RouteOpt's own implementation each arrival is checked against the bucket as it stands and inserted, so that the arrivals to a bucket form a chain; Appendix C is about removing this dependence.

Smoothing and the root. The root loop studied here runs the Wentges rule of Appendix A.2 with the center updates, the weight adaptation and the ladder (A.7) of Section 3.2. The paper's measurements stop at the first exact certificate of the root; after it, RouteOpt continues as a branch-price-and-cut solver (You & Yang, 2026): rounded capacity cuts are separated heuristically with the CVRPSEP routines, limited-memory rank-1 cuts involving up to five customer rows by seed-set enumeration and local search (Pecin et al., 2017b), branching is on edges with three-phase strong branching whose candidates are screened by a learned model (You et al., 2026), and reduced-cost arc elimination and route enumeration are applied once the gap permits. Root pricing is the first cut-free column generation up to the exact certificate, with no cut, no branching decision and no arc elimination. When the root loop separates rounded capacity cuts, the learned center is reinstalled at the start of every round, while LIGHT-level smoothing activates only after a fixed number of iterations within the round; the solver comparisons of Section 4 and Appendix D are cut-free, so this rule is stated for completeness, and the CPU profile of Appendix C.2 is the one measurement taken on a cut-loaded root.

A.5 Case 2: unrelated parallel machine scheduling

Instance, columns and master. Jobs $J = \{1, \dots, n\}$ are assigned to machines $M = \{1, \dots, m\}$; job j takes time p_{kj} on machine k and carries a weight w_j, and the objective is $\sum_j w_j C_j$ with C_j the completion time of j. A column is a *single-machine schedule* (k, S), $S \subseteq J$, sequenced in the weighted-shortest-processing-time order of machine k (p_{kj}/w_j nondecreasing), which is optimal within a fixed job set; its cost is $c_k(S) = \sum_{j \in S} w_j\, C_j(k,S)$ with $C_j(k,S) = \sum_{l \in S,\ l \preceq_k j} p_{kl}$. Schedule columns originate in the identical-machine setting of van den Akker et al. (1999); the machine-indexed master of Yu et al. (2026), on which the experiments of Section 4.4 build with the deviations described below, is

$$\min_{\lambda \ge 0} \Big\{ \sum_{k \in M} \sum_{S \subseteq J} c_k(S)\, \lambda_{k,S} : \sum_{k \in M} \sum_{S \ni j} \lambda_{k,S} \ \ge\ 1 \ (j \in J),\ \sum_{S \subseteq J} \lambda_{k,S} \ \le\ 1 \ (k \in M) \Big\}, \quad \text{(A.11)}$$

with duals $\pi_j \ge 0$ on the jobs and $\sigma_k \le 0$ on the machines and reduced cost $rc_{k,S} = c_k(S) - \sum_{j \in S} \pi_j - \sigma_k$; the formulation of Hijazi et al. (2026) below imposes equality on the job rows, so that its job duals are unrestricted in sign. With singleton schedules as the only initial columns and at most one schedule per machine, the initial restricted master needs artificial covering variables when $n > m$.

Pricing by dynamic programming. For machine k index the jobs in its weighted-shortest-processing-time order and let $F_k(j,t)$ be the minimum of $\sum_{l \in S}(w_l C_l - \pi_l)$ over subsets S of the first j jobs with total processing time t; then

$$F_k(j,t) = \min \big\{ F_k(j-1,t),\ \ F_k(j-1, t - p_{kj}) + w_j\, t - \pi_j \big\}, \quad \text{(A.12)}$$

with $F_k(0,0) = 0$ and $+\infty$ assigned to every infeasible state, including all states with $t < 0$, because appending j to a schedule of total time $t - p_{kj}$ completes it at time t. The minimum reduced cost is $rc^{\min}(\pi, \sigma) = \min_k \left[\min_t F_k(n,t) - \sigma_k \right]$, computed exactly in $O(n \sum_j p_{kj})$ per machine for integer processing times: a pseudo-polynomial table with no dominance rule. In the implementation used here the recursion is vectorized. The reference implementation of Yu et al. (2026) shares the master (A.11) and uses Wentges smoothing with $\alpha_0 = 0.5$ and Lagrangian-bound center updates; its mis-pricing update differs from (A.7), and it disables smoothing once the weight reaches zero, whereas the harness of this paper restarts its retry sequence at each iteration.

The published learned column-generation setting. The comparison of Section 4.4 runs inside the released code of Hijazi et al. (2026), who replace the pricing oracle by a pointer transformer and keep an exact dynamic program of the form (A.12) as the fallback that preserves the optimality guarantee. Their setting differs from (A.11) in three respects that matter for reading the results: the master is a set-partitioning master without artificial variables, each job covered exactly once and each machine used at most once; the restricted master is seeded with one hundred random job subsets per machine; and no dual stabilization is used. Their contribution is on the pricing side; Section 4.4 combines their learned pricer with column initialization and dual stabilization inside one implementation.

The learned center for scheduling. The center of Section 4.4 is the scheduling analogue of the model of Appendix B.2.1: a bipartite set-attention network over jobs and machines, with cross-attention biased by the processing times, that reads the instance data alone and predicts the certified root duals in the scale-free units $\pi_j/(w_j \bar{L})$ and σ_k/C_0, where $\bar{L}$ is the mean machine load and $C_0 = \bar{L} \sum_j w_j$; it was trained on 168 instances of the benchmark generator of Yu et al. (2026), selected on a validation split, and not retrained on the distribution of Hijazi et al. (2026). The prediction is projected onto $\pi \geq 0$, $\sigma \leq 0$ and installed at iteration 0 together with its own Lagrangian value from one exact sweep, after which the Wentges rule runs unchanged.

B THE LEARNING COMPONENTS IN DETAIL

Both learned components read the instance file and nothing else, run once before column generation starts, and hand the solver an object it already uses: a pool of routes for the lazy archive of Section 3.1, and an initial center for the smoothing rule of Section 3.2. This appendix describes each component as a training problem, with its architecture, features, labels and training parameters, and as an inference step, with what the prediction becomes inside the solver. The shared label set and its splits are in Appendix B.3.

B.1 LEARNING TO PREFILL

B.1.1 TRAINING

Model architecture. The deployed scorer f_θ of (4) is a message-passing network over the candidate graph $E(I)$, the 20 nearest neighbors of each customer plus the depot edges. Node inputs are the position relative to the depot in units of the mean depot distance, the demand fraction q_i/Q, the depot distance in the same units, the candidate degree divided by 20, and a depot flag; edge inputs are the 26 instance-only edge features below. Three rounds of messages are built from $h_i + h_j$, $h_i \odot h_j$ and e_{ij}, hence symmetric in the endpoints, at width 192, followed by an edge head. The graph is constructed from the instance alone and is therefore identical during training and at generation time. Table 5 lists the architecture and the training parameters.

Features. The 26 edge features are instance-only quantities: normalized edge and depot distances, nearest-neighbor ranks and membership indicators of the two endpoints, the depot-centered angular separation of the endpoints, the normalized savings term $d_{0i} + d_{0j} - d_{ij}$, the endpoint demands, a depot-edge flag and instance context. No feature depends on a solution, a dual or a pool, so the same feature extractor serves training and deployment.

Labels. The label of an edge is its *heat* ν_e of (4): among the routes of the instance's reference pool that price out at the reference center, the fraction of those visiting i or j that traverse e. The reference

pool is the metaheuristic pool of the instance, built by hybrid genetic search with three seeds and a time and route-count budget that grows with the instance size, and the reference center is the tree-predicted center CG of the instance, the deployed center at the time the labels were computed, so that the heated edges are the edges used by the routes that a plausible iteration-0 center asks for; a route prices out when its reduced cost at that center is below -10^{-9}. The supervised edge set $H(I)$ adds the reference-pool edges to $E(I)$, so that the supervised set includes reference-pool edges outside the message-passing graph while message passing uses only the instance-derived candidate graph, and the regression target is $\sqrt{\nu_e}$, which spreads the mass of the many lightly heated edges. Training uses the training split of Appendix B.3 and excludes instances with fewer than 50 negative-reduced-cost reference routes, leaving 192 of the 200 training instances.

Training parameters. Table 5 gives the optimizer, schedule and selection rule. The checkpoint is the epoch with the best validation Spearman rank correlation with ν. Offline rank correlation is the selection statistic and nothing more; the pool is evaluated end to end in Appendix D.1.

Table 5: Deployed edge scorer: architecture and training parameters.

item	value
graph	20-NN candidate graph plus depot edges, built from the instance file; identical at training and at generation time
node inputs	relative position and depot distance in units of the mean depot distance, q_i/Q, candidate degree$/20$, depot flag
edge inputs	the 26 instance-only edge features
message passing	three rounds at width 192; messages from $h_i + h_j$, $h_i \odot h_j$, e_{ij}; edge head; no dropout
parameters	1 432 321
target and loss	$\sqrt{\nu_e}$ on $H(I)$, squared error
training set	192 of the 200 training instances (instances with fewer than 50 negative-reduced-cost reference routes excluded)
optimizer	AdamW, learning rate 8×10^{-4}, weight decay 0, cosine annealing to $0.01\times$ the initial rate, batches of 16 instances, gradient-norm clip 5, seed 0
epochs and selection	260 epochs, no early stopping; checkpoint with the best validation Spearman rank correlation (epoch 158)

B.1.2 Inference: from the heat map to the archive

The generator. Algorithm 1 turns the heat map into a pool of routes with the constants of Table 6. The deployed pools are built under a time budget of ten seconds per instance, four independent seeds running in parallel and merged at the end by customer set. Routes are deduplicated by customer set, keeping the cheapest ordering seen, so the pool holds distinct columns by construction; a costlier ordering of the same customer set is a dominated column.

How the pool is used. The pool never enters the initial restricted master; it is a lazy archive that supplies columns at pricing time by (5). At every pricing round, before any labeling, the archive offers at most $B = 300$ routes, the most negative at the smoothed query q_t among those not yet offered, and the solver admits those with $rc_r(u_t) < -\epsilon$ at the raw dual, the same test a generated column must pass; each route is offered once, and the labeling call of that round is skipped when at least one route is admitted. On instrumented development instances the archive served 24 to 52 % of LIGHT rounds this way; when it supplies no admissible column, the solver resumes labeling. Because admission uses the solver's own test, the archive changes the trajectory of column generation and not the value it converges to, and because the routes are never in the initial master, no basis is imposed on the first simplex solve; Appendix D.1 records why preloading the same kind of columns into the initial master fails.

Cost. In a dedicated timing study on an uncontended node, excluding coverage instrumentation that a deployment does not run, the learned generator's whole-command wall is 0.23 to 0.30 of the metaheuristic pool's; Appendix D.1 compares the learned pool with a heuristic pool whose generation time is included.

Table 6: Learned constructor: constants of Algorithm 1 in the deployed configuration.

item	value
time budget T	ten seconds per instance, shared by the seeds
seeds	$W = 4$ independent seeds in parallel, merged by customer set; random seed 42
neighborhood $\mathcal{N}(i)$	edges of the 20-NN graph among the 12 highest-scoring candidates at either endpoint
floors and exponents	$\delta = 10^{-9}$ on the depot heat, $\delta' = 10^{-12}$ on the walk weights, coverage exponent $\gamma = 2$
tour sampling	τ uniform on $\{2, 4, 8, 16\}$ per tour; multiplicative noise $e^{0.15Z}$, $Z \sim \mathcal{N}(0, 1)$, per step; nearest-unvisited fallback may leave $E(I)$
tour to routes	neighbor-list 2-opt on the giant tour; optimal split (Prins, 2004) under capacity and a route-length bound of $\max\{25, \lceil 3n/K_{\min} \rceil\}$ customers; intra-route 2-opt (Polish); split-and-search twice per tour
local search	relocate, swap and 2-opt* in up to three passes per route set
stopping rule	a seed stops when 400 consecutive tours add no new customer set
deduplication	by customer set, cheapest ordering kept
archive batch	at most $B = 300$ routes offered per pricing round, each route once

B.2 Learning the dual center

B.2.1 Training

Model architecture. The deployed center CN is a set transformer g_φ over customer tokens plus an instance-context token: three pre-layer-norm blocks of multi-head attention at $d = 128$ with eight heads and dropout 0.1, a geometric additive bias on the attention logits that encodes the distance between the two customers, a per-customer softplus head for the direction and a scalar head for the log magnitude of (7). The vehicle row of the center comes from a gradient-boosted tree model on instance context, and the tree center CG, gradient-boosted trees that predict each customer independently from the same features, is the tree-based alternative to the direction model. Table 7 lists the architectures and the training parameters.

Features. The 53 per-customer features are the depot distance in raw, normalized and rank form, the demand and the demand fraction q_i/Q, the distance to the customer centroid, neighborhood statistics at two neighborhood sizes, and instance context such as n, $K_{\min}$, Q, $n/K_{\min}$ and the dispersion of depot distances and demands. Features derived from the root LP value are excluded, so that the model needs no solve, and the generator-family digits of the label instances are dropped so that the model applies outside that generator. The trees use the same features; the vehicle-row model uses nine instance-context features.

Labels. The label of an instance is its certified optimal root dual π^*, customer block and vehicle row, obtained as described in Appendix B.3. The direction target is the singleton-normalized coordinate $y_i = \pi_i^*/(2d_{0i})$, so that the network predicts each customer's dual relative to the cost of serving it alone, and the magnitude target is the ratio ρ^* of the customer-dual norm to the singleton-cost norm of (7); the vehicle-row model regresses $\pi^*_{\text{veh}}/\operatorname{mean}_i d_{0i}$ on the instance context. Including the vehicle row, the median relative L_2 error of the predicted center on the test split is 0.2017, against 0.1958 for the customer block alone; the paper's claim is not that this error is small but that the direction it encodes is a good initial center, which Appendix D.2 tests against controls.

Training parameters and magnitude calibration. Table 7 gives the optimizer, schedule and selection rule; CN was selected on validation cosine after the box projection out of 87 training runs of the same family. A direction is installable only once it has a magnitude, and what the solver sees is the magnitude after the box projection of (8), so the scale must be produced without consulting the tree model at test time. The deployed calibration forms a target norm T in log space as the geometric mean of de-biased estimates available at test time, the scale heads of two auxiliary networks of the same family and the norm predicted by a network trained under a plain L_2 loss, and scales the direction by the factor for which the norm *after* the box projection equals T, found by bisection; when the target exceeds the norm of the box corner, the corner itself is installed.

Table 7: Deployed dual center: architectures and training parameters.

item	value
inputs	53 per-customer features plus an instance-context token, all computable from the instance file
set transformer (CN)	three pre-layer-norm blocks, $d = 128$, eight heads, geometric additive attention bias, dropout 0.1; per-customer softplus direction head; scalar log-magnitude head; 423 834 parameters
labels	$y_i = \pi_i^*/(2d_{0i})$ for the direction; $\rho^* = \|\pi_C^*\|_2/\|2d_0\|_2$ for the magnitude
loss	(7) with $\beta = 0.3$
optimizer	AdamW, learning rate 10^{-3}, weight decay 0.01, batch 8, ten warm-up epochs then a cosine schedule, cap 220 epochs, patience 29, seed 0; best epoch 44
selection	validation cosine after the box projection, out of 87 runs of the same family
box projection	$0 \le c_i \le \max(2d_{0i} - c_{\text{veh}}, 0)$, with c_{veh} clipped at 0
magnitude calibration	target norm T: geometric mean in log space of three de-biased estimates (scale heads of two auxiliary set-transformer networks with $d = 64$, four heads, two blocks; the norm of a network of that architecture trained under a plain L_2 loss); s by bisection so that $\|\Pi_{\mathcal{B}}(sD_0\hat{y})\|_2 = T$; corner installed when T exceeds the corner norm
vehicle-row model	histogram gradient-boosted trees, 200 trees, depth 4, learning rate 0.06, at least 5 samples per leaf, seed 0; target $\pi^*_{\text{veh}}/\operatorname{mean}_i d_{0i}$ on 9 instance-context features
tree center (CG)	histogram gradient-boosted trees, at most 400 iterations with early stopping on a 0.1 validation fraction and patience 30 (369 trees), depth 6, learning rate 0.06, at least 40 samples per leaf, L_2 regularization 1.0, 255 bins, seed 0; same 53 features and labels

B.2.2 Inference: from the prediction to the smoothing center

The predictor is evaluated once per instance, in process: feature construction takes a median of 1.10 s and network inference 0.226 s. The customer block is clipped to the singleton-based box $\mathcal{B}$ of (8) after scaling, the vehicle dual from the trees is clipped at 0, and the resulting c_0 is installed at the first pricing iteration. The solver certifies it by one exact pricing call at c_0, whose outcome is the Lagrangian bound $L(c_0)$ of (6); the columns that call prices out are dropped, so the center adds no column of its own and acts only through the queries it shapes. This replaces the seeding exact call that the stabilized baseline pays for its first center. From then on the rule of Section 2 runs unchanged: whenever a complete exact pricing improves the Lagrangian bound, the center moves to that dual point by (9), and the weight adapts by (10); the center is therefore *not* held fixed after initialization. The deployed configuration also carries the endgame switch, which sets $\alpha = 0$ and stops adaptation after two consecutive complete, non-retry exact calls that each admit between 1 and 99 columns, and the LIGHT fallback: after an empty smoothed LIGHT query the solver retries at the raw dual, and if that retry admits columns the remainder of the LIGHT phase prices at the raw dual. Under cut separation the prediction would be reinstalled every cut round, a configuration not used in this paper. The calibration above aims the center at a scale at which pricing stays productive (Section 3.2).

B.3 Labels and data

The label set consists of 320 instances of the XML generator (Queiroga et al., 2022), 80 each at 100, 200, 300 and 500 customers, twenty per route-length class from medium to ultra long, and none at 400 customers, so that the XML400 instances of the evaluation panels are unexposed to both learned components; no Vrp-Set-X instance is labeled. Each label instance was solved at the root by vanilla RouteOpt, without prefilled columns, to the exact certificate, and the converged restricted master was written out with its basis. The label π^*, customer block and vehicle row, is the basic optimal dual of that master, recovered by re-solving the written program with primal simplex from the saved basis in zero iterations, and accepted only if its objective matches the root value to a relative 10^{-7} and no reference-pool route prices out at it beyond 10^{-6}. A metaheuristic reference pool was built for every label instance and for every evaluation instance with the same recipe. The instances are

split $200/60/60$ into training, validation and test instances, stratified into sixteen groups by size and route-length class with split seed 20260913; both learned components use the same instance split and the same 60 validation instances, the center training on all 200 training instances and selecting on validation cosine, the scorer retaining 192 of them after the reference-route-count filter and selecting on validation rank correlation. None of the 95 evaluation instances of Section 4 is a training or validation instance of either model; its XML400 and Vrp-Set-X instances are not labeled at all.

C GPU EXACT PRICING IN DETAIL

C.1 BUCKET-GRAPH LABELING AND THE DOMINANCE TEST

RouteOpt prices by bucket-graph labeling (Sadykov et al., 2021). A label is a partial route leaving the depot together with its reduced cost, its resource consumption, its ng memory and, in the tree, its cut states. The labels at a customer are kept in buckets indexed by a resource interval, and the algorithm processes buckets in resource order, extending each label along the bucket arcs and the jump arcs into the buckets of its successors. In RouteOpt's own implementation this is a *push*: an extended label arrives at its target bucket and is checked there against the bucket's move-to-front list of stored labels, with early exit once a dominator is found; before a stored label is extended, a cross-bucket check compares it with the earlier buckets of the same customer through reduced-cost prefix bounds; buckets are sorted by reduced cost when their level is done. The predicate compares resources, tests ng-set inclusion and, with cuts, checks a reduced-cost gap corrected by the cut duals; the LIGHT and HEAVY levels use weaker predicates, which is why they keep few labels and run quickly. Because every arrival is checked against the bucket as it stands at that moment and changes it, the arrivals to a bucket form a chain; this is the sequential dependence the backend removes. Under symmetry RouteOpt labels forward to the meet point and joins two forward half-routes.

C.2 WHERE THE TIME GOES

Table 8 is the CPU profile that motivated the backend design of Section 3.3, measured on vanilla RouteOpt at the root of Vrp-Set-X instances: the four dominance components together account for close to half of the profiled root-execution self cycles, of which 9.4 points are cut-state dominance and absent from the cut-free root measured in this paper, label extension for a small part, and one EXACT call evaluates orders of magnitude more dominance predicates than a LIGHT or a HEAVY call while taking two orders of magnitude longer. On the vanilla solver this concentration converts into little end to end, because exact pricing is a minor share of the cut-free root there; prefill and the learned center change that share, as Section 4.3 and Figure 2 show, and the exact level becomes the stage where a further lever pays.

Table 8: Where root labeling spends its time on the CPU: vanilla RouteOpt, single thread, cut-loaded root. Left: median self-cycle shares of profiled root execution over 35 long-route runs; the dominance total sums the four components. Right: per-level counters on the root of X-n228-k23.

kernel	self %	level	calls	s / call	predicates / call
in-bucket dominance loop	18.4	LIGHT	862	0.05	4.5×10^5
dominance core (load, ng subset)	12.5	HEAVY	201	0.10	9.2×10^5
cut-state dominance	9.4	EXACT	125	12.0	2.1×10^8
cross-bucket dominance	5.3				
dominance, total	*45.6*				
label extension	5.0				
cut-state update	8.3				
concatenation	8.4				
LP solver	8.2				

C.3 REMOVING THE SEQUENTIAL DEPENDENCE

The backend keeps the contract of the CPU pricer and changes the shape of the computation. At each resource level it expands the current frontier and groups the candidates by target bucket; same-level candidates form the next wave of that level, and candidates for later levels wait for theirs. The

candidates of one bucket in a wave form a segment, tested in three stages. Stored labels against candidates: each candidate independently, stopping at its first dominator, whose stored label is marked as a witness. Candidates against candidates in the canonical order (reduced cost, resources, ng cardinality, generation index): the serial reference accepts candidates in that order, and the exact GPU mode compares each candidate in parallel against all earlier candidates that survived the first stage. Surviving candidates against the stored labels, with marked witnesses exempt from deletion. A repair pass then restores any deleted candidate without a surviving direct dominator. Every deleted label therefore has a dominator alive at the end of the wave, an invariant maintained by retaining the stored labels that serve as deletion witnesses and stronger than what RouteOpt's own implementation offers under tolerance. Label state is held in structure-of-arrays form; shared-memory tiles reuse candidate prefixes across the comparisons of a block; an optional warp-per-segment path for small segments exists and is disabled in the deployed configuration; the cross-bucket filter runs one warp per frontier label and uses the same reduced-cost prefix bound as the CPU. Label extension, dominance testing and the forward–forward join run on the GPU; route reconstruction, deduplication by canonical route identity and the final selection of returned columns run on the CPU, and with stable numbering, canonical sorting and fixed scan rules the output does not depend on scheduling. The heuristic levels, the master and the cut loop stay on the CPU, and dispatch sits inside the exact level, so the column-generation loop is unchanged.

C.4 Certification and safeguards

The backend runs only where its preconditions hold, namely symmetric instances and an ng bit-set width within the compiled bound, with automatic CPU fallback otherwise. Every returned column is re-priced on the CPU at the query q_t actually used, since the solver prices at q_t and admits at u_t, and mismatches are counted rather than silently accepted. An empty backend answer is not accepted as a certificate: before column generation terminates, one CPU EXACT call re-proves the same dual, so the root bound of a GPU run is certified by the CPU pricer. The backend's output policy is aligned with the CPU truncation rule, since a different rule changes how many columns a call returns and hence how many calls the loop makes. Correctness is checked by comparing final root bounds across arms and, in a diagnostic configuration, the returned column sets call by call, and the completed arms of every comparison in this paper must report the same root bound.

C.5 What the backend buys per call

Table 9 reconciles the same EXACT calls on the CPU and on the GPU: each call is served by the backend and, in the same process on the same input, by RouteOpt's own labeling, so machine and contention cancel. Table 10 extends the comparison to three cards on the learned stack, for the 10 Vrp-Set-X instances with the longest exact pricing time in a preliminary sweep, two of which lie below the 300-customer floor of the main panel: per instance, the median per-call seconds of steady-state exact calls on the single CPU thread and on each card, with the first call, which includes GPU initialization, and the last call, which is the CPU re-proof by design, excluded. Bounds agree on 10/10 instances and the three cards' exact-call counts on 10/10; only the exact-pricing clock is compared, which leaves out the LP solves, run on the CPU on every card. Section 4.3 quotes the medians over instances of these ratios.

Table 9: The same EXACT calls served by RouteOpt's labeling on the CPU and by the backend on an L4, first cut-free CG of five Vrp-Set-X instances. The last column estimates the end-to-end gain by subtracting the measured pricing time, a speed-up factor with higher better; it is not a separate vanilla run.

instance	EXACT calls	CPU (s)	L4 (s)	CPU / L4	end-to-end (est.)
X-n303-k21	9	11.8	3.0	3.9	1.12
X-n327-k20	13	11.4	2.6	4.4	1.14
X-n449-k29	11	63.6	15.9	4.0	1.11
X-n459-k26	25	107.3	30.6	3.5	1.18
X-n561-k42	14	25.3	6.0	4.2	1.05

Table 10: Per-instance exact pricing on three GPUs, learned stack, the 10 X instances with the longest exact pricing time: steady-state exact calls, median seconds per call on one CPU thread and on each card, and their ratios. Bounds agree on 10/10 instances and call counts on 10/10.

instance	calls	CPU (s)	L4 (s)	B200 (s)	RTX (s)	B200/L4	RTX/L4	L4/CPU
X-n1001-k43	41	15.838	4.015	1.205	1.298	0.30	0.32	0.25
X-n367-k17	74	3.321	0.691	0.358	0.357	0.52	0.52	0.21
X-n936-k151	37	5.233	1.239	0.594	0.575	0.48	0.46	0.24
X-n261-k13	18	2.252	0.491	0.272	0.259	0.55	0.53	0.22
X-n561-k42	14	2.036	0.422	0.231	0.222	0.55	0.53	0.21
X-n957-k87	20	1.101	0.324	0.218	0.176	0.67	0.54	0.29
X-n233-k16	17	0.954	0.214	0.158	0.148	0.74	0.69	0.22
X-n327-k20	15	0.920	0.181	0.110	0.098	0.61	0.54	0.20
X-n351-k40	10	1.205	0.224	0.171	0.152	0.76	0.68	0.19
X-n536-k96	9	1.104	0.210	0.178	0.157	0.85	0.75	0.19

D Ablations and additional results

Two ablations isolate the two learned components of Section 3: how the columns of a pool should be supplied to column generation, and what the initial center of the smoothing rule should be. Both use 10 real instances of the main panel—X-set instances of the two deepest route-size classes, long and very long. Every arm of an instance runs inside one job on one node, bracketed by two vanilla solves; the vanilla column is the mean of the four vanilla solves of the two brackets and is the denominator of every ratio in both tables; a run that times out, the limit being 1.5 times the instance's first vanilla time plus $300\,\mathrm{s}$, is reported as a lower bound ($\geq$); every completed arm reaches the same certified root bound as vanilla. Tables 11 and 12 give the root times in seconds instance by instance, so that the reader sees the raw measurements, and close with the instance-equal geometric mean of the ratio to vanilla; intervals are two-sided $90\,\%$ t intervals on the log ratios.

D.1 Supplying the columns: which pool, and how

The folklore and its best evidence. Practitioners of column generation know the advice that prefilling the master with heuristic columns "does not help": the LP starts wide, the first solve is slow, and the columns are thrown away again. The two oracle arms of Table 11 show where this experience comes from, and why it is only half right. PSUPP preloads exactly the support of the converged root LP—the routes with positive value in an optimal root solution, the best columns anyone could know in advance—and it is the worst arm of the table: 5 of the 10 runs time out, and the five that finish are $0.732\times$ vanilla but still $2.478\times$ slower than the deployed prefill on those instances (PSUPP/PN, [1.620, 3.790]). A few perfect columns give the master a degenerate vertex to stand on and nothing to move with. PCONV, which preloads the *complete* column set of the converged master, is the best arm at $0.183\times$: what a pool has to supply is not the optimum but a rich neighborhood of near-optimal routes. This is the sense in which imperfect is perfect. Neither oracle is a method—both need a completed solve—but they mark two contrasting extremes of what a pool can supply, and on the deep XML instances of a preliminary experiment preloading a heuristic pool ($1.209\times$, $n = 16$) or the converged basis ($1.223\times$) confirmed the folklore rather than beating it (Section 1).

Learning is necessary. PN is the deployed prefill: the learned pool (Appendix B) admitted lazily, the mechanism of the next paragraph. PH keeps that mechanism and replaces the learned pool by an ordinary heuristic pool, one HGS run with one seed.

On the same instances and the same node the learned pool is $1.233\times$ faster (PH/PN, $90\,\%$ interval [1.121, 1.356]), and is the faster of the two on 9 of the 10 instances. The scorer lets the constructor propose an order of magnitude more feasible routes than one heuristic run (326470 against a few thousand on X-n1001-k43), and the archive is selective enough to take only the ones that matter.

The steps we take are necessary. The deployed archive keeps the pool *outside* the model: the master starts from the singleton initial columns; after every LP re-solve the pool is priced under the current smoothed dual, the 300 most negative routes are handed to the same admission check the pricers use, the labeling call of that iteration is skipped when at least one route is admitted, and the LP is re-optimized from the previous basis; a route is offered at most once. One scan of the 326470-

route pool of X-n1001-k43 costs 18 ms; the pool served the first 29 iterations exclusively and 2.7% of it ever entered the model. PDUMP drops this discipline and places the whole learned pool in the initial master. On these instances no difference is detected between the lazy archive and the dump (PDUMP/PN 1.044×, interval [0.960, 1.135]). The gap is modest on these instances because the solver's column cleaner caps the live master at ten thousand columns and repairs a wide start within a few iterations; it is the deep XML instances, where the first barrier solve lands on a bad vertex, on which preloading costs 1.209×. The archive costs nothing when preloading would have been harmless and saves the run when it would not, which is why it is the deployed choice.

Table 11: Supplying the columns: root time in seconds on the long and very-long X instances of the main panel, same job and node per row; n customers, K vehicles; PN learned pool via the lazy archive (deployed), PH HGS pool via the archive, PDUMP the same learned pool preloaded, PCONV/PSUPP oracles preloading the converged column set or its support; "≥" marks a timed-out run.

instance	n	K	vanilla	PN	PH	PDUMP	PCONV	PSUPP
X-n327-k20	326	20	112.6	31.5	48.8	34.9	12.6	77.9
X-n367-k17	366	17	453.2	184.3	201.3	167.6	108.2	≥ 955.4
X-n449-k29	448	29	611.3	200.2	253.0	200.7	85.2	317.4
X-n513-k21	512	21	955.8	325.2	368.3	461.4	308.2	≥ 1672
X-n561-k42	560	42	438.7	74.5	100.1	80.5	61.7	347.1
X-n641-k35	640	35	1251	524.5	661.9	579.5	187.2	879.0
X-n783-k48	782	48	4076	1222	1144	1210	634.8	≥ 6478
X-n801-k40	800	40	1024	309.7	498.7	312.4	156.1	≥ 1841
X-n979-k58	978	58	13017	4490	4955	4931	1954	13695
X-n1001-k43	1000	43	7287	1825	2171	1498	3461	≥ 11399
GM of ratio to vanilla			1	0.305	0.377	0.319	0.183	0.732 (5 censored)
wins (< vanilla)				10/10	10/10	10/10	10/10	4/10

D.2 CHOOSING THE CENTER: THE LEARNED DIRECTION

The center model predicts the *direction* of the converged root dual; the vector is then given a magnitude and projected onto the structural box $\mathcal{B} = \{c : 0 \le c_i \le \max(2d_{0i} - c_{\text{veh}}, 0)\}$ of (8) (Appendix B), and installed as the iteration-0 center of Wentges smoothing. Table 12 tests this on the same instances with the center alone, no pool, as in the *Center* column of Table 1.

Learning is necessary. S is the standard Wentges rule: no learned center, the solver seeds its own center at the end of the first heuristic phase. The learned center CN, the deployed model of Appendix B.2.1, is 1.369× faster (S/CN, interval [1.257, 1.492]), consistent with the panel-wide Center column of Table 1; standard smoothing alone recovers only 0.837× of vanilla here.

The steps we take are necessary. CRAND keeps the deployed magnitude, vehicle row and box projection but randomly permutes the customer entries of the learned center—the direction is destroyed and nothing else. It costs 1.176× (CRAND/CN, [1.094, 1.264]): the direction is what the smoothed pricing queries inherit, and it is what the network learns. CRAND still beats standard smoothing (0.719× against 0.837× of vanilla), so a calibrated center installed at iteration 0 is worth part of the gain by itself, and the learned direction adds the rest on top.

D.3 VERY SHORT ROUTES

Route-size classes. The XML generator (Queiroga et al., 2022) controls the average route size r, the expected number of customers per route: r is drawn uniformly from the range of the chosen class (Table 13), the demands q_i from the chosen demand distribution, and the capacity is set to $Q = \max\{\max_i q_i,\ \lceil r \sum_i q_i/n \rceil\}$ ($Q = \lfloor r \rfloor$ for unitary demands), so that the minimum fleet $K_{\min} = \lceil \sum_i q_i/Q \rceil$ has about n/r vehicles; the class is the fourth digit of the instance code. A Vrp-Set-X instance is assigned to the class whose center, the mean of $n/K_{\min}$ over the XML300 and XML500 instances of that class, is nearest to its own $n/K_{\min}$.

Table 12: Choosing the center: root time in seconds on the instances and protocol of Table 11, center-only arms: S standard smoothing, CRAND the learned center with its customer entries permuted, CN the deployed center.

instance	n	K	vanilla	S	CRAND	CN
X-n327-k20	326	20	112.6	115.1	93.8	73.2
X-n367-k17	366	17	453.2	343.2	312.8	368.9
X-n449-k29	448	29	611.3	487.7	430.0	340.1
X-n513-k21	512	21	955.8	817.8	686.2	542.7
X-n561-k42	560	42	438.7	397.0	333.1	260.2
X-n641-k35	640	35	1251	1119	856.5	774.3
X-n783-k48	782	48	4076	3515	2988	2487
X-n801-k40	800	40	1024	966.3	843.5	732.6
X-n979-k58	978	58	13017	7623	7048	5667
X-n1001-k43	1000	43	7287	6015	5421	4534
GM of ratio to vanilla			1	0.837	0.719	0.611
wins ($<$ vanilla)				9/10	10/10	10/10

Table 13: Route-size classes of the XML generator: the range of the average route size r and the class name used in this paper.

class	1	2	3	4	5	6
r	$[3, 5]$	$[5, 8]$	$[8, 12]$	$[12, 16]$	$[16, 25]$	$[25, 50]$
name	very short	short	medium	long	very long	ultra long

The very-short class. The main panel of Section 4 starts at the short class. The very-short class was measured under the same protocol, five instances per set, and Table 14 reports it separately. Pricing is comparatively easy on these instances: under the ng-route relaxation with memories of size 8 (Appendix A.4) the pricer already returns many elementary routes, and since every prefilled column is an elementary route, prefill has more to offer here than on the harder classes, bringing root pricing to 0.126$\times$ vanilla on all 20 instances (best 0.075$\times$). The dual direction the RMP produces is adequate on its own, so the learned center adds no guidance and costs a little (Prefill+Center against Prefill 1.376$\times$, interval [1.148, 1.649]), and exact pricing is too small a share of the run for the GPU backend to matter (1.003$\times$). Giving this class weight in the main panel would overstate the effect of prefill and hide the bottleneck of the harder instances, which is why it is shown here as an outlier class rather than pooled into Table 1.

Table 14: Very short routes: geometric-mean root-time ratio to vanilla under the protocol of Table 1, with the number of instances faster than vanilla in parentheses.

Set	Inst.	Prefill	Center	GPU	Prefill+Center
X	5	0.108 (5/5)	0.764 (5/5)	0.985 (3/5)	0.117 (5/5)
XML300	5	0.159 (5/5)	0.793 (5/5)	1.019 (1/5)	0.209 (5/5)
XML400	5	0.127 (5/5)	0.785 (5/5)	1.001 (1/5)	0.162 (5/5)
XML500	5	0.115 (5/5)	0.781 (5/5)	1.010 (3/5)	0.229 (4/5)
all	20	0.126 (20/20)	0.781 (20/20)	1.003 (8/20)	0.173 (19/20)

E REPRODUCIBILITY

The routing experiments build on RouteOpt (You & Yang, 2026), from whose released source the vanilla solver and our fork are compiled with identical settings, and the scheduling case study builds on the released code, instances and seeds of Hijazi et al. (2026). Linear programs are solved with Gurobi 13.0.1, and the GPU pricing backend is built with CUDA 12.9.1. Every source of randomness is seeded: the label set of Appendix B.3 is split with seed 20260913, and the pool generators, the smoothing rule, and the training runs use fixed seeds, those of the generators and the training runs being listed in Appendix B.