BEYOND DECOUPLED DECISIONS: JOINTLY LEARNING LABEL EXTENSION AND COLUMN SELECTION IN COLUMN GENERATION FOR VEHICLE ROUTING PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Column generation (CG) is a powerful framework for large-scale routing optimization that avoids exhaustive enumeration by iteratively solving master and pricing problems. However, its efficiency is constrained by two coupled bottlenecks: the labeling-based pricing problem spends substantial computation exploring labels and uncovering many redundant candidate columns, while only a small, complementary subset materially improves the master problem. We study CG with routing pricing subproblems solved by labeling algorithms and formulate label-extension ordering and column selection as two sequential decisions within a shared Markov decision process. We consider a practical setting that learns from rich pre-collected CG trajectories without solving CG online during training, using data generated by existing CG solvers. We propose COUPLED-CG, a two-stage offline Q-learning framework with stage-specific conservative regularization that combines support-aware value learning for extension ordering, relation-conditioned marginal column selection, and coupled Bellman updates that propagate value across the two stages and successive CG iterations. Experiments on public CVRPTW benchmarks demonstrate substantial reductions in labeling effort and end-to-end CG runtime. Ablation studies quantify the additional gains from jointly learning the two decisions, while instance-level analyses identify the structural characteristics of instances that benefit most from Coupled CG.

1 INTRODUCTION

Column generation (CG) solves large-scale linear programs by generating decision variables (columns) on demand (Desaulniers et al., 2006). At each iteration, a restricted master problem (RMP) optimizes over the current columns, while a pricing subproblem searches for new columns that can improve the solution. In routing problems, a column typically represents a feasible route, and the pricing problem becomes seeking negative-reduced-cost routes under operational constraints such as vehicle capacity and customer time windows (Irnich & Desaulniers, 2005; Costa et al., 2019). This decomposition has substantially advanced exact routing optimization, enabling, for example, capacitated vehicle routing instances with up to 360 customers that had previously been tackled only heuristically (Pecin et al., 2017). We focus on this important class of CG methods in which the pricing subproblem is a constrained routing problem (Righini & Salani, 2008; Baldacci et al., 2011; Dabia et al., 2013; He et al., 2019; Li et al., 2022; 2026; Mhamedi et al., 2026).

Within this class, we consider pricing methods based on labeling algorithms, which construct routes incrementally while tracking resources such as capacity and time and pruning infeasible or dominated partial routes (Feillet et al., 2004; Irnich & Desaulniers, 2005). Despite their effectiveness, these CG methods face two interacting computational bottlenecks. First, pricing remains a generally NP-hard constrained routing problem. Because the number of partial routes (labels) can grow exponentially, the labeling algorithm needs to explore and compare a large number of labels at every CG iteration. Second, after candidate columns are generated, selecting which ones to add to the RMP remains challenging. Reduced cost provides a natural measure of immediate improvement, but a column with a more negative reduced cost does not necessarily yield faster overall convergence to

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.

CG termination (Chi et al., 2022). Adding many columns can reduce the number of CG iterations, yet it also enlarges the RMP and increases its solution cost (Morabit et al., 2021).

These two bottlenecks are coupled across CG iterations. Label-extension decisions determine both pricing effort and the columns available to the RMP, while the column-selection decisions alter the RMP solution and the dual information that guides subsequent pricing. Together, these challenges limit the scale of routing problems that can be solved exactly (Desaulniers et al., 2019; Cabrera et al., 2023).

Existing CG acceleration studies address pricing and column selection separately. On the pricing side, classical methods reduce labeling effort through stronger dominance and bounding rules, bidirectional search, specialized graph representations, route relaxations, and selective pricing (Righini & Salani, 2006; Tilk et al., 2017; Desaulniers et al., 2019; Sadykov et al., 2021); more recent learning-based approaches further reduce the pricing search space through learned arc or heuristic selection (Xu et al., 2023; Morabit et al., 2023; Xia et al., 2024). For column selection, classical and learning-based strategies selectively admit generated columns to control RMP growth and accelerate CG termination (Touati et al., 2010; Morabit et al., 2021; Chi et al., 2022; Yuan et al., 2024). These approaches can yield substantial computational gains, including reported runtime reductions of up to 40% from learned pricing guidance (Morabit et al., 2023) and 30% from learned column selection (Morabit et al., 2021). However, existing methods typically treat the two decisions separately, without explicitly coordinating pricing effort with downstream column selection. This separation leaves open the question of *how pricing and column selection should be coordinated and what additional computational gains such coordination can deliver*.

In this study, we address this gap by formulating CG as a Markov decision process (MDP) with two sequentially coupled policies: one selects a label-extension ordering rule for pricing, while the other selects a subset of the resulting columns for the RMP. We design stage-specific rewards that penalize labeling effort and measure RMP objective improvement; the latter telescopes over an episode, providing intermediate feedback without altering our surrogate CG acceleration objective of minimizing cumulative labeling effort. To avoid repeatedly interacting with the costly CG solver during policy training, we generate a fixed set of trajectories from CG runs on training instances and learn from them offline. We develop COUPLED-CG, a two-stage offline conservative Q-learning framework. It uses support-aware conservative learning to address limited action coverage and relation-aware marginal column selection to capture interactions among candidate columns. The two stages are further linked through coupled Bellman updates: the pricing-stage target incorporates downstream column-selection value, while the column-selection target incorporates the next iteration's pricing value. This enables both policies to account for their longer-term effects across CG iterations. We evaluate COUPLED-CG on public CVRPTW benchmarks (Solomon, 1987) to assess its impact on labeling effort and end-to-end CG runtime, and analyze how problem characteristics influence the benefits of coupling. Our contributions are:

1. **COUPLED-CG formulation**. We formulate label-extension ordering and column selection as coupled policies in an MDP with reward shaping that preserves the cumulative labeling-effort objective.
2. **Offline RL framework**. We develop COUPLED-CG, which combines support-aware conservative learning and relation-aware column selection with cross-stage Bellman updates that propagate value between pricing and column selection across CG iterations.
3. **Numerical evaluation**. We demonstrate the effectiveness of COUPLED-CG on public CVRPTW benchmarks and analyze its computational gains, generalization to larger instances, and the structural settings in which coupling is most beneficial.

2 RELATED WORK

Pricing and Labeling Acceleration Classical methods accelerate routing pricing through route relaxations, bidirectional search, stronger dominance and bounding rules, specialized graph representations, and selective pricing (Baldacci et al., 2011; Righini & Salani, 2006; Tilk et al., 2017; Sadykov et al., 2021; Enerbäck et al., 2024; Desaulniers et al., 2019). Learning-based approaches further guide pricing by reducing the search network through arc selection (Morabit et al., 2023), guiding or selecting pricing heuristics (Shen et al., 2022; Xu et al., 2023), and identifying or gen-

erating promising candidate columns (Xia et al., 2024; Abouelrous et al., 2025). These approaches improve route generation, but pricing efficiency alone does not capture its impact on overall CG performance. We instead learn extension ordering jointly with downstream column selection to account for their interaction across CG iterations.

Column Selection and Coordination with Pricing Classical column-selection methods use reduced-cost criteria and diversification to control RMP growth (Touati et al., 2010), while learning-based approaches predict column usefulness (Morabit et al., 2021), model selection sequentially (Chi et al., 2022), or select multiple columns per iteration (Yuan et al., 2024). FFCG (Hu et al., 2025) further performs sequential marginal-value selection without enumerating subsets. Our contribution is not sequential selection itself, but its integration with learned pricing-order control in a coupled offline framework, using relation-conditioned set values and cross-stage credit assignment. Existing pricing and column-selection components have also been combined (Morabit et al., 2021; 2023), but are trained separately; we instead learn the two decisions jointly as sequentially coupled policies.

Offline Reinforcement Learning (RL) Offline RL learns policies from fixed datasets without further environment interaction during training (Prudencio et al., 2023), while conservative value learning mitigates overestimation under limited action coverage (Fujimoto et al., 2019; Kumar et al., 2020). We adapt these ideas to the coupled structure of CG, where pricing and column selection have distinct action spaces but sequentially affect one another. COUPLED-CG combines stage-specific conservative learning with relation-aware column selection and cross-stage Bellman targets that propagate value between the two decisions across CG iterations.

3 PROBLEM FORMULATION

3.1 PRELIMINARY

We use the CVRPTW as a representative setting to introduce the CG framework considered in this work. Let $\mathcal{N}$ be the customer set and $\mathcal{R}$ the set of feasible routes, where each route starts and ends at the depot, visits each customer at most once, and satisfies capacity and time-window constraints. Route $r \in \mathcal{R}$ has cost c_r, and $\alpha_{i,r} \in \{0,1\}$ indicates whether it serves customer i. With fleet limit V and route-selection variable X_r, the route-based formulation is

$$\mathcal{M}_{\text{IP}} = \min \sum_{r\in\mathcal{R}} c_r X_r \quad \text{s.t.} \quad \sum_{r\in\mathcal{R}} \alpha_{i,r} X_r = 1,\ \forall i \in \mathcal{N}, \quad \sum_{r\in\mathcal{R}} X_r \le V, \quad X_r \in \{0,1\},\ \forall r \in \mathcal{R}, \tag{1}$$

Because $|\mathcal{R}|$ can grow exponentially with $|\mathcal{N}|$, CG solves the LP relaxation using only a restricted route set $\mathcal{R}' \subseteq \mathcal{R}$. Solving the resulting restricted master problem (RMP) yields dual variables d_i for the customer-coverage constraints and q for the fleet constraint, defining the reduced cost of route r as $\bar{c}_r = c_r - \sum_{i\in\mathcal{N}} d_i \alpha_{i,r} - q$. The pricing problem then searches $\mathcal{R}$ for negative-reduced-cost routes. Selected routes are added to $\mathcal{R}'$, the RMP is re-solved, and the process repeats until no improving route remains, at which point the full LP relaxation is solved. CG can subsequently be embedded within branch-and-bound for exact integer optimization. We focus on accelerating the CG loop; notation is summarized in Appendix A.

3.2 PROBLEM FORMULATION: COUPLED ACCELERATION OF COLUMN GENERATION

Acceleration objective. We seek a CG acceleration strategy that generalizes across problem instances $\mathcal{I} \sim \mathcal{D}$, where $\mathcal{D}$ represents the instance distribution of the target application and may be specified through an instance sampler. For each instance, CG repeatedly solves a pricing problem P_t and reoptimizes the RMP until termination (Appendix B). Acceleration therefore requires considering both the computational effort of individual pricing calls and the progress induced by the generated and selected columns.

Controlling pricing search through extension ordering. The pricing problem is first solved by heuristic labeling with early stopping, which terminates when the number of explored labels reaches

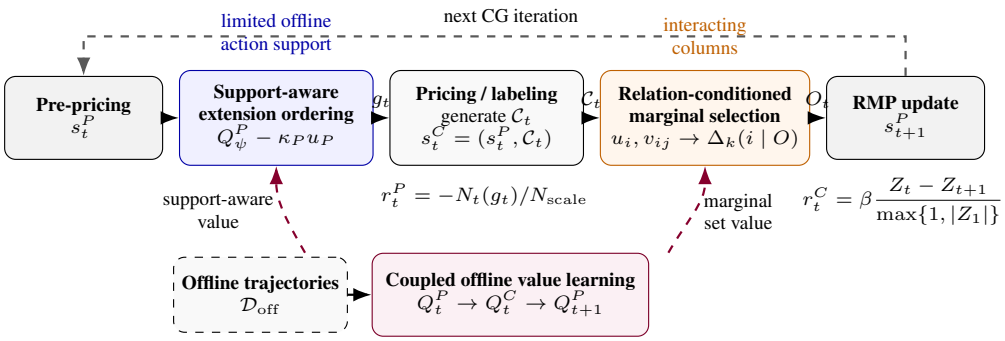


Figure 1: Overview of COUPLED-CG. Each CG iteration couples extension ordering and column selection through $Q_t^P \to Q_t^C \to Q_{t+1}^P$ to accelerate CG.

a predefined limit Γ; exact labeling is invoked when needed for optimality certification. At iteration t, an extension-ordering rule $g_t \in \mathcal{G}$ determines the order in which active labels are processed in pricing problem P_t. We consider strategies that modify only this processing order while leaving the extension, feasibility, reduced-cost, and valid dominance rules unchanged. Details of both labeling procedures are provided in Appendix C.

The effect of extension ordering differs between exact and early-stop labeling. Under exact labeling, ordering can change the search trajectory and computational effort, for example, by discovering dominating labels earlier and avoiding unnecessary extensions, but it does not change the pricing optimum: all orderings solve the same pricing problem exactly and therefore return the same optimal reduced-cost value. Under early stopping, however, the ordering also determines which parts of the search space are explored within the fixed budget Γ, and can therefore change the candidate routes and reduced-cost values returned. Thus, extension ordering affects pricing effort in both settings and, under early stopping, additionally shapes the candidate columns available to the subsequent RMP update.

Joint pricing and column-selection objective. At iteration t, extension ordering shapes the candidate set $\mathcal{C}_t$, while the selected subset $O_t \subseteq \mathcal{C}_t$ determines the RMP update and the dual information defining the next pricing problem:

$$g_t \longrightarrow \mathcal{C}_t \longrightarrow O_t \longrightarrow \text{RMP update} \longrightarrow P_{t+1}. \tag{2}$$

Pricing decisions therefore affect the choices available to column selection, while column-selection decisions affect subsequent pricing effort. We optimize these decisions jointly using cumulative labeling effort as a surrogate for CG computation. Let $N_t(g_t)$ denote the number of labels explored in pricing call t, including heuristic labeling and exact fallback when invoked. With extension-ordering policy π^P and column-selection policy π^C, our objective is

$$\mathcal{M} : \min_{\pi^P, \pi^C} \mathbb{E}_{\mathcal{I} \sim \mathcal{D}} \left[\sum_{t=1}^{T(\pi^P, \pi^C; \mathcal{I})} N_t(g_t) \right], \tag{3}$$

where $T(\pi^P, \pi^C; \mathcal{I})$ denotes the number of CG iterations until exact pricing certifies termination. This objective captures both per-call labeling effort and the effects of the two policies on subsequent pricing problems and trajectory length.

4 COUPLED-CG: JOINT LEARNING FOR CG ACCELERATION

Solving $\mathcal{M}$ from offline CG trajectories requires coordinating the two decision stages, handling uneven support for extension-ordering actions, and selecting interacting columns from a combinatorial candidate set. COUPLED-CG addresses these issues through coupled offline value learning, support-aware extension ordering, and relation-conditioned marginal column selection. Figure 1 summarizes the overall framework.

4.1 TWO-STAGE MDP FOR COUPLED-CG DECISIONS

We formulate the coupled pricing and column-selection decisions as a two-stage MDP that directly follows the interaction identified in Section 3. At iteration t, the current RMP defines the pre-pricing state s_t^P, represented as a bipartite graph between column and constraint nodes with instance- and iteration-level features (Details are in Appendix D). The pricing policy first selects an extension-ordering rule

$$g_t \sim \pi^P(\cdot \mid s_t^P), \qquad g_t \in \mathcal{G}, \tag{4}$$

which determines the labeling search and candidate set $\mathcal{C}_t$, producing

$$s_t^C = \mathrm{PricingTransition}(s_t^P, g_t) = (s_t^P, \mathcal{C}_t). \tag{5}$$

The column-selection policy then chooses $O_t \subseteq \mathcal{C}_t$, and the selected columns update the RMP:

$$O_t \sim \pi^C(\cdot \mid s_t^C), \qquad s_{t+1}^P = \mathrm{RMPUpdate}(s_t^C, O_t). \tag{6}$$

Thus, one CG iteration forms the sequential transition $s_t^P \xrightarrow{g_t} s_t^C \xrightarrow{O_t} s_{t+1}^P$, which explicitly couples pricing-search control with the downstream RMP update.

We construct stage-specific rewards that provide dense credit assignment while preserving the joint acceleration objective $\mathcal{M}$:

$$r_t^P = -\frac{N_t(g_t)}{N_{\mathrm{scale}}}, \qquad r_t^C = \beta \frac{Z_t - Z_{t+1}}{\max\{1, |Z_1|\}}, \tag{7}$$

where $N_{\mathrm{scale}} > 0$ is a fixed global normalization constant and $\beta \geq 0$ controls the shaping strength.

Proposition 1 (Reward consistency with the acceleration objective) *For an undiscounted CG episode terminating under exact pricing, if N_{scale} is fixed across policies, then*

$$\max_{\pi} \sum_{t=1}^{T} (r_t^P + r_t^C) \quad \Longleftrightarrow \quad \min_{\pi} \sum_{t=1}^{T} N_t(g_t). \tag{8}$$

Hence, the RMP-improvement reward provides dense shaping without changing the optimizer of $\mathcal{M}$.

The proof is provided in Appendix E. Proposition 1 applies to the undiscounted episodic return before introducing the conservative learning regularizers, under a fixed initial RMP and exact CG termination. It shows that the proposed stage rewards are consistent with the cumulative labeling-effort objective. The conservative and support-aware terms introduced for offline learning, however, modify the value-learning procedure and are not covered by this equivalence result.

4.2 COUPLED OFFLINE VALUE LEARNING

We learn the two sequential decisions from pre-collected CG trajectories

$$\mathcal{D}_{\mathrm{off}} = \left\{ \left(s_t^P, \widehat{g}_t, r_t^P, s_t^C, \widehat{O}_t, r_t^C, s_{t+1}^P, \mathrm{done}_t \right) \right\},$$

where $\widehat{g}_t$ and $\widehat{O}_t$ denote the logged extension-ordering and column-selection actions, respectively. We encode the pre-pricing state as

$$h_t = \mathrm{GNN}_\theta(s_t^P),$$

and the post-pricing state as

$$\widetilde{h}_t = \mathrm{Enc}_\theta^C(h_t, \mathcal{C}_t),$$

where $\mathcal{C}_t$ is the candidate-column set contained in s_t^C. We provide details of trajectory construction in Appendix F.

We define $Q_\psi^P(h_t, g)$ and $Q_\phi^C(\widetilde{h}_t, O)$ as the long-term values of extension ordering and column selection, respectively. Because the two decisions occur sequentially within each CG iteration, we couple their value learning across stages:

$$y_t^P = r_t^P + (1 - \mathrm{done}_t) V_{\bar{\phi}}^C(\widetilde{h}_t), \qquad y_t^C = r_t^C + (1 - \mathrm{done}_t) V_{\bar{\psi}}^P(h_{t+1}), \tag{9}$$

where $V_{\bar{\phi}}^C$ and $V_{\bar{\psi}}^P$ denote the target column-selection and support-aware extension-ordering values, respectively. Thus, value information propagates as $Q_t^P \to Q_t^C \to Q_{t+1}^P$, coupling extension ordering with the downstream value of generated columns and column selection with its effect on subsequent pricing. The following subsection introduces the stage-specific target constructions, with additional optimization details in Appendix K.

4.3 STAGE-SPECIFIC POLICY MODELS

Support-aware extension ordering. We use five complementary extension-ordering rules to induce diverse labeling search trajectories

$$\mathcal{G} = \{\text{minimum reduced cost}, \text{minimum load}, \text{minimum time}, \text{maximum load}, \text{maximum time}\}.$$

The reduced-cost rule follows the pricing objective, while the load- and time-based rules steer the search across different resource regimes. Detailed definitions of the five extension-ordering rules are provided in Appendix G.

To motivate adaptive extension ordering, we further conduct experiments comparing different fixed rules across instances. The results, reported in Appendix H, show that the preferred rule varies across instances, motivating the need to select the ordering strategy adaptively. In addition, to make adaptive extension ordering reliable under uneven offline action coverage, we penalize rules that are weakly supported in similar states. Let $h_t = \mathrm{GNN}_\theta(s_t^P)$ and let ξ_g denote the prototype of rule g constructed from its logged state embeddings. We define

$$\widetilde{\omega}_P(h_t, g) = \exp\left(-\frac{\|\bar{h}_t - \xi_g\|_2^2}{2\sigma^2}\right), \qquad \omega_P(h_t, g) = \frac{\widetilde{\omega}_P(h_t, g)}{\sum_{g' \in \mathcal{G}} \widetilde{\omega}_P(h_t, g') + \epsilon}, \tag{10}$$

where $\bar{h}_t = h_t/(\|h_t\|_2 + \epsilon)$. We define the support uncertainty as $u_P(h_t, g) = 1 - \omega_P(h_t, g)$, with prototype construction detailed in Appendix I. The extension-ordering rule is selected as

$$g_t = \arg\max_{g \in \mathcal{G}} \left[Q_\psi^P(h_t, g) - \kappa_P u_P(h_t, g)\right]. \tag{11}$$

where $\kappa_P \geq 0$ penalizes weakly supported actions. This preserves adaptive pricing-search control while reducing extrapolation toward unsupported actions.

Relation-conditioned marginal column selection. To model column interactions without constructing an explicit subset-action catalogue, we perform column selection through sequential marginal decisions. We first encode the candidate set $\mathcal{C}_t$ as a relation graph. An edge-aware encoder produces candidate representations $z_{t,i}$, from which unary and symmetric pairwise heads predict individual values u_i and compatibilities $v_{ij} = v_{ji}$. Implementation details of the relation graph, edge-aware encoder, and marginal selection procedure are provided in Appendix J. Let $\widetilde{h}_t = \mathrm{Enc}_\theta^C(h_t, \mathcal{C}_t)$. For target cardinality $k > 1$, we define the column-set value as

$$Q_{\phi,k}^C(\widetilde{h}_t, O) = \frac{1}{k}\sum_{i \in O} u_i + \frac{2}{k(k-1)} \sum_{\substack{i<j \\ i,j \in O}} v_{ij}. \tag{12}$$

For $k = 1$, we use the unary value only. Under this factorization, the marginal value of adding candidate $i \notin O$ is

$$\Delta_k(i \mid O) = \frac{1}{k}u_i + \frac{2}{k(k-1)} \sum_{j \in O} v_{ij}. \tag{13}$$

Starting from the mandatory minimum-reduced-cost column, the policy repeatedly selects $i^\star$ using $\arg\max_{i \in \mathcal{C}_t \setminus O} \Delta_k(i \mid O)$ until k columns are selected or no unused candidate remains. The marginal construction avoids enumerating $\binom{M}{k}$ subsets while capturing column quality and pairwise interactions, and naturally generalizes across different M and k.

Training and inference. The two value functions are trained with temporal-difference and stage-specific conservative losses,

$$\mathcal{L}^P = \mathcal{L}_{\mathrm{TD}}^P + \lambda_P \mathcal{L}_{\mathrm{CCQL}}^P, \qquad \mathcal{L}^C = \mathcal{L}_{\mathrm{TD}}^C + \lambda_C \mathcal{L}_{\mathrm{MCR}}^C, \tag{14}$$

where $\mathcal{L}_{\mathrm{CCQL}}^P$ and $\mathcal{L}_{\mathrm{MCR}}^C$ denote the conservative pricing and marginal-selection regularizers, respectively. Together, conservative support-aware pricing, relation-conditioned marginal selection, and coupled Bellman updates provide robust and scalable coordination of both decisions toward the common objective $\mathcal{M}$; optimization details are given in Appendix K.

5 EXPERIMENTS

We evaluate COUPLED CG on Solomon CVRPTW instances from the six families C1, C2, R1, R2, RC1, and RC2 (Solomon, 1987). All learning-based methods are trained on the same pool of 300 C-family instances with 6–18 customers, ensuring identical instance-level training access across methods. The competing learning-based methods construct their training data from the same instance pool according to their respective learning procedures. Details of dataset construction and baseline adaptation are provided in Appendices F and L.

Evaluation is conducted entirely on held-out instances. We consider two size regimes: small instances with 6–18 customers, which lie within the training size range, and larger instances with 20–60 customers, which lie outside the training size range and evaluate generalization to larger problems. Since training uses only C-family instances, evaluation on R and RC families additionally assesses transfer across instance structures. We compare COUPLED CG with Plain CG (Desaulniers et al., 2006), Yuan CG (Yuan et al., 2024), Morabit CG (Morabit et al., 2021), and FFCG (Hu et al., 2025). Performance is measured by CG time, end-to-end wall-clock time, the number of labels explored, the number of CG iterations, and the percentage of instances completed within the time limit. Detailed implementation settings are provided in Appendix M.

5.1 OVERALL PERFORMANCE AND GENERALIZATION

Overall performance. Across all four evaluation groups, Table 1 shows that COUPLED-CG consistently lowers computational cost. Relative to Plain CG, CG time decreases by 62.5%, 43.1%, 76.6%, and 39.4% on C-Small, C-Large, R/RC-Small, and R/RC-Large, respectively, with corresponding reductions in explored labels of 67.7%, 60.2%, 78.2%, and 44.2%. On the instances solved by all five methods, COUPLED-CG achieves the lowest mean CG and end-to-end runtime in every group. CG Solved (%) is identical across methods in the two small-instance groups and highest for COUPLED-CG on C-Large, while on R/RC-Large it is slightly lower than for Yuan and Morabit; the reported runtime gains therefore characterize the jointly solved subset rather than uniform dominance over the full test set.

Generalization. Because all learning-based methods are trained only on small C-family instances and every evaluation instance is held out, the four groups probe different levels of transfer. C-Small evaluates unseen instances within the training family and size range, C-Large tests generalization to larger problem sizes, R/RC-Small tests transfer to different instance structures, and R/RC-Large combines both shifts. COUPLED-CG continues to achieve substantial reductions in CG time and labeling effort across all of these settings, including the larger R/RC cases, showing that the learned guidance transfers beyond the training distribution.

5.2 COMPONENT AND HYBRID COMPARISONS

Figure 2 compares the contributions of learned pricing guidance and column selection and examines the additional benefit of their coupled training. The pricing-only and column-selection-only variants retain the corresponding architectures and rewards of COUPLED-CG, but are trained independently using within-stage TD targets (Details in Appendix O). All three value-based variants use the undiscounted setting $\gamma = 1$, eliminating discounting as a source of variation across these comparisons. Both independent components improve over Plain CG, while COUPLED-CG yields the most consistent gains in CG time, pricing effort, and CG iterations. Combining the independently trained pricing policy with FFCG, Yuan, or Morabit produces substantially larger variance and, in some cases, severe degradation, suggesting sensitivity to the candidate-column distribution induced by learned pricing.

5.3 SENSITIVITY TO CANDIDATE-POOL SIZE AND SELECTION CARDINALITY

We further evaluate the sensitivity of COUPLED-CG to the candidate-pool size M and selection cardinality k on C-family instances with 40–60 customers. Table 2 evaluates whether the advantage of COUPLED-CG depends on the specific candidate-pool and selection sizes used in the main experiments. Across all six M/k settings, COUPLED-CG consistently reduces CG time, end-to-end time, labeling effort, and the number of CG iterations relative to Plain CG. Although the magnitude

Table 1: Performance comparison on small and large instances. Runtime, labeling effort, and CG iterations are averaged over instances solved by all five methods within the 900-second time limit, while **CG Solved (%)** reports the percentage of all instances in each group solved by each method within the time limit.

Group	Method	CG Time (s)	End-to-End (s)	Labels Explored	CG Iterations	CG Solved (%)
C-Small (6–18 customers) Solved by all: 281	COUPLED-CG	**2.12**	**2.15**	**22,662**	**18.33**	99.65
	Plain CG (Desaulniers et al., 2006)	5.66	5.69	70,091	19.17	99.65
	Yuan CG (Yuan et al., 2024)	6.08	6.25	72,759	19.44	99.65
	Morabit (Morabit et al., 2021)	5.70	5.70	118,104	42.79	99.65
	FFCG (Hu et al., 2025)	3.87	3.87	66,925	20.32	99.65
C-Large (20–60 customers) Solved by all: 256	COUPLED-CG	**29.54**	**29.56**	**271,943**	**86.03**	**84.47**
	Plain CG (Desaulniers et al., 2006)	51.92	51.97	683,037	102.79	83.23
	Yuan CG (Yuan et al., 2024)	50.20	50.40	678,558	102.38	83.54
	Morabit (Morabit et al., 2021)	44.55	44.55	947,024	91.08	82.30
	FFCG (Hu et al., 2025)	44.50	44.51	863,216	93.14	83.23
R/RC-Small (6–18 customers) Solved by all: 186	COUPLED-CG	**2.90**	**3.29**	**27,399**	15.90	85.32
	Plain CG (Desaulniers et al., 2006)	12.37	12.39	125,670	16.52	85.32
	Yuan CG (Yuan et al., 2024)	12.82	12.95	120,832	16.19	85.32
	Morabit (Morabit et al., 2021)	6.68	6.68	93,110	**12.19**	85.32
	FFCG (Hu et al., 2025)	6.85	6.86	101,611	17.04	85.32
R/RC-Large (20–60 customers) Solved by all: 200	COUPLED-CG	**21.66**	**21.70**	**191,331**	**61.16**	74.46
	Plain CG (Desaulniers et al., 2006)	35.74	35.86	342,953	67.07	72.30
	Yuan CG (Yuan et al., 2024)	29.62	29.93	343,698	67.21	75.90
	Morabit (Morabit et al., 2021)	26.77	27.78	269,481	65.43	**76.26**
	FFCG (Hu et al., 2025)	29.43	29.44	362,566	61.61	74.82

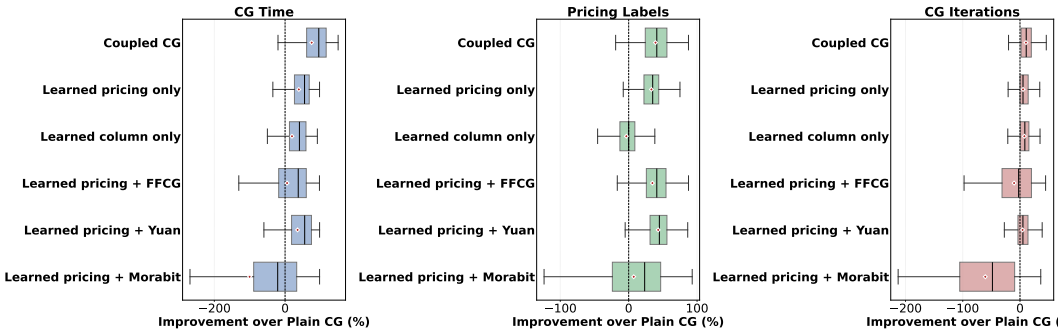


Figure 2: Ablation study of pricing guidance and column selection. Positive values indicate improvement over Plain CG.

of the improvement varies with M and k, the gains persist from the smallest setting $(10, 5)$ to the largest $(60, 30)$. This indicates that the benefit of the proposed framework is not tied to a particular candidate-set configuration.

5.4 WHICH INSTANCE STRUCTURES BENEFIT MOST FROM COUPLED-CG?

We investigate which instance structures benefit most from COUPLED-CG. Each instance is characterized by six properties: load tightness, demand variance, spatial spread, time-window overlap, normalized time-window-width variance, and transition density; detailed definitions are provided in Appendix P. For each property, we partition instances into quintiles, and Figure 3 reports both the average CG-time reduction over Plain CG and the percentage of instances where COUPLED-CG is faster.

The clearest trends arise from temporal complexity and transition density. As transition density increases from Q1 to Q5, the average CG-time reduction rises from -3.2% to 59.8%, while the fraction of instances where COUPLED CG is faster increases from 7.1% to 85.7%. Similar improvements in consistency are observed for time-window overlap and normalized time-window-width variance, whereas load tightness, demand variance, and spatial spread show less monotonic effects. These results indicate that COUPLED-CG is most promising for routing problems with dense feasible transitions and heterogeneous temporal constraints, such as ride-pooling, pickup-and-delivery, and time-window-constrained last-mile delivery.

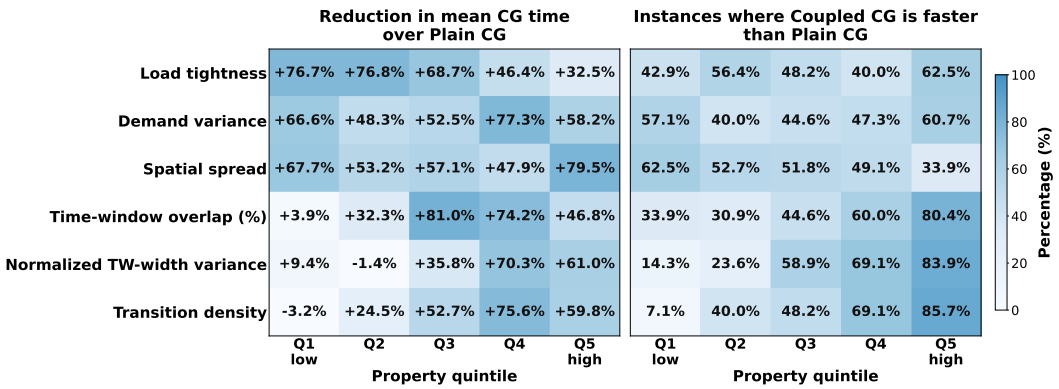


Figure 3: Structural analysis of COUPLED-CG across CVRPTW instance properties. The left panel reports the reduction in mean CG time relative to Plain CG, $100\left(1-\bar{T}^{\text{Coupled}}/\bar{T}^{\text{Plain}}\right)$, within each property quintile, while the right panel reports the percentage of instances for which COUPLED-CG is faster than Plain CG.

Table 2: Performance comparison under different M/k settings on C instances with 40-60 customers. All metrics are averaged over the instances solved by both COUPLED-CG and Plain CG within the time limit.

M/k	Method	Solved by Both	CG Time (s)	End-to-End (s)	Labels Explored	CG Iterations
10/5	COUPLED-CG	22	**43.61**	**45.61**	**419,024**	**136.59**
	Plain CG	22	81.92	82.07	885,506	145.73
20/10	COUPLED-CG	25	**53.43**	**55.92**	**403,916**	**96.20**
	Plain CG	25	132.37	132.78	996,610	110.44
30/15	COUPLED-CG	25	**106.11**	**108.79**	**433,809**	**80.04**
	Plain CG	25	120.21	120.48	767,626	87.68
40/20	COUPLED-CG	26	**96.67**	**99.26**	**390,730**	**69.04**
	Plain CG	26	144.56	144.82	800,774	78.58
50/25	COUPLED-CG	26	**80.06**	**83.12**	**352,220**	**65.04**
	Plain CG	26	106.03	106.31	727,022	74.35
60/30	COUPLED-CG	24	**53.31**	**56.15**	**272,305**	**57.63**
	Plain CG	24	87.67	87.99	639,016	68.96

Additional sensitivity analyses with the reward coefficient β are provided in Appendix Q.

6 CONCLUSION

COUPLED-CG demonstrates that jointly coordinating pricing and column selection can yield additional gains beyond learning either decision independently. By propagating value across the two stages, the method reduces labeling effort and CG runtime while preserving exact termination, and its improvements extend to larger and structurally different CVRPTW instances.

Several limitations remain. First, the pricing policy is restricted to a predefined set of extension-ordering rules rather than making finer-grained label- or extension-level decisions, which limits the expressiveness of pricing control. Second, the learning objective uses cumulative labeling effort as a surrogate for computational cost. Although the experiments additionally evaluate end-to-end runtime, the training objective does not explicitly account for other computational components such as RMP reoptimization and neural inference. Third, the current empirical study is limited to CVRPTW, and further evaluation is needed to determine how the coupled framework behaves for column-generation problems with substantially different pricing structures and master-problem characteristics. Finally, the current study focuses on root-node CG; extending the framework to branch-and-price introduces additional interactions with branching decisions and changing pricing subproblems that are not considered here.

AI USE STATEMENT

In this work, we used ChatGPT to assist with writing and language refinement, including improving clarity, grammar, and presentation of the manuscript. We also used ChatGPT during implementation to generate preliminary code drafts and suggestions for selected components. All AI-assisted text was reviewed and revised by the authors, and all AI-generated or AI-assisted code was inspected, tested, and modified as needed to ensure correctness and consistency with the proposed methodology. ChatGPT was not used to generate experimental results, perform evaluations, or make final scientific conclusions. We take responsibility for the final content of this work, including all text, claims, code, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personally identifiable information, or sensitive private data. The study uses public benchmark instances and focuses on algorithmic methods for accelerating column generation. We are not aware of any specific ethical concerns arising from the methodology or experiments presented in this work.

REPRODUCIBILITY STATEMENT

We provide anonymized source code and scripts as supplementary material to reproduce the proposed COUPLED-CG framework and the reported experiments. Section 4 describes the main algorithm and learning framework. Appendix section M provides details on dataset construction, training procedures, baseline implementations, hyperparameter settings, and evaluation protocols. The supplementary code includes instructions for environment setup, data preparation, model training, and evaluation.

REFERENCES

Ahmed Abouelrous et al. Reinforcement learning for solving the pricing problem in column generation. *Transportation Research Part C*, 2025. arXiv:2504.02383.

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.

Nicolás Cabrera, Jean-François Cordeau, and Jorge E Mendoza. Solving the park-and-loop routing problem by branch-price-and-cut. *Transportation Research Part C: Emerging Technologies*, 157: 104369, 2023.

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

Luciano Costa, Claudio Contardo, and Guy Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4):946–985, 2019. doi: 10.1287/trsc.2018.0878.

Said Dabia, Stefan Ropke, Tom Van Woensel, and Ton De Kok. Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation science*, 47(3):380–396, 2013.

Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

Guy Desaulniers, Diego Pecin, and Claudio Contardo. Selective pricing in branch-price-and-cut algorithms for vehicle routing. *EURO Journal on Transportation and Logistics*, 8(2):147–168, 2019.

Jesper Enerbäck, Lars Eveborn, and Elina Rönnberg. Pricing for the evrptw with piecewise linear charging by a bounding-based labeling algorithm. In *24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024)*, pp. 3:1–3:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004. doi: 10.1002/net.20033.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Qie He, Stefan Irnich, and Yongjia Song. Branch-and-cut-and-price for the vehicle routing problem with time windows and convex node costs. *Transportation Science*, 53(5):1409–1426, 2019.

Yi-Xiang Hu, Feng Wu, Shaoang Li, Yifang Zhao, and Xiang-Yang Li. Ffcg: Effective and fast family column generation for solving large-scale linear program. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 11238–11245, 2025.

Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pp. 33–65. Springer, 2005.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33:1179–1191, 2020.

Hongli Li, Zengxiang Lei, Xinwu Qian, and Satish V Ukkusuri. Anticipatory demand planning for microtransit with strategic use of latent capacity at fixed stops. *Transportation Research Part E: Logistics and Transportation Review*, 212:104939, 2026.

Jiliu Li, Min Xu, and Peng Sun. Two-echelon capacitated vehicle routing problem with grouping constraints and simultaneous pickup and delivery. *Transportation Research Part B: Methodological*, 162:261–291, 2022.

Tayeb Mhamedi, Marilène Cherkesly, and Guy Desaulniers. A branch-price-and-cut algorithm for the multicommodity two-echelon vehicle routing problem with time windows. *INFORMS Journal on Computing*, 2026.

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

Mourad 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. doi: 10.1287/ijoo.2022.0082.

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, 2017.

Rafael Figueiredo Prudencio, Marcos ROA Maximo, and Esther Luna Colombini. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE transactions on neural networks and learning systems*, 35(8):10237–10257, 2023.

Giovanni Righini and Matteo Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 2006.

Giovanni Righini and Matteo Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, 51(3):155–170, 2008.

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.

Yunzhuang Shen, Yuan Sun, Xiaodong Li, Andrew S. Eberhard, and Andreas T. Ernst. Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring. In *AAAI Conference on Artificial Intelligence*, 2022. URL `https://ojs.aaai.org/index.php/AAAI/article/view/21230`.

Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

Christian Tilk, Ann-Kathrin Rothenbächer, Timo Gschwind, and Stefan Irnich. Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, 261(2):530–539, 2017. doi: 10.1016/j.ejor.2017.03.017.

N Touati, Lucas Létocart, and Anass Nagih. Solutions diversification in a column generation algorithm. *Algorithmic Operations Research*, 5(2):86–95, 2010.

Ying Xia et al. A neural column generation approach to the vehicle routing problem. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2024. arXiv:2406.12454.

Kai Xu et al. Enhancing column generation by reinforcement learning-based hyper-heuristic. *arXiv preprint*, arXiv:2310.09686, 2023.

Hao Yuan, Leilei Fang, and Shaowei Song. A reinforcement-learning-based multiple-column selection strategy for column generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 8209–8216, 2024. doi: 10.1609/aaai.v38i8.28661. URL https://arxiv.org/abs/2312.14213.

A Notation table

A.1 Notation

Table 3 summarizes the main notation used throughout the problem formulation and learning framework.

B Column generation framework

Figures 4 and 5 illustrate the standard CG procedure and the proposed Coupled-CG framework, respectively. In the original CG framework, each iteration first solves the RMP and then invokes heuristic labeling to quickly search for negative reduced-cost columns. If heuristic labeling fails to identify any valid improving column, exact labeling is used as a fallback. Unlike heuristic labeling, which may terminate early and therefore does not guarantee pricing optimality, exact labeling solves the pricing problem exactly and certifies whether any negative reduced-cost column remains. CG terminates only when exact labeling finds no improving column.

Coupled-CG preserves this standard pricing structure while introducing learning at two decision points. Before both heuristic and exact labeling, an extension-ordering learner selects the rule used to guide label processing. After valid columns are generated, a column-selection learner selects the subset to be added to the RMP. Thus, the learned components control both pricing search and the RMP update, while exact labeling remains the fallback mechanism that preserves the standard CG termination condition.

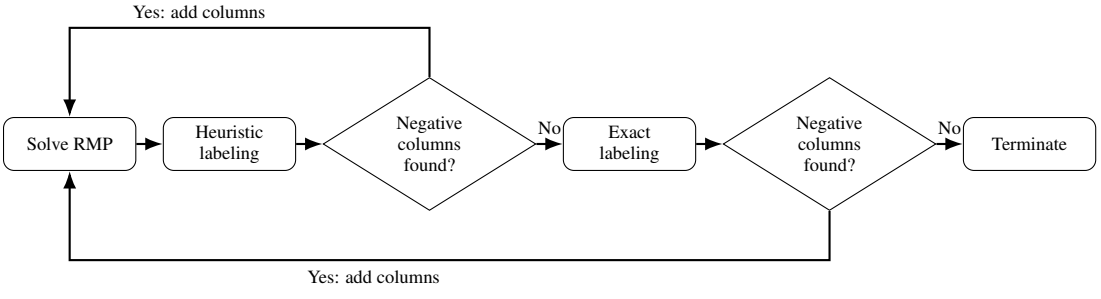


Figure 4: Original CG framework. Heuristic labeling is attempted first, while exact labeling is invoked when no improving column is found. CG terminates only when exact labeling finds no improving column.

Table 3: Summary of the main notation.

Notation	Description
$\mathcal{N}$	Set of customers.
$\mathcal{R}$	Set of all feasible routes satisfying the routing constraints.
$\mathcal{R}_t$	Set of routes currently included in the restricted master problem (RMP) at CG iteration t.
c_r	Cost of route $r \in \mathcal{R}$.
$\alpha_{i,r}$	Indicator equal to 1 if route r serves customer i, and 0 otherwise.
X_r	Master-problem decision variable associated with route r.
V	Maximum number of available vehicles.
d_i	Dual variable associated with the coverage constraint of customer i.
q	Dual variable associated with the vehicle-limit constraint.
$\bar{c}_r$	Reduced cost of route r.
I	Routing instance drawn from the target instance distribution $\mathcal{D}$.
$\mathcal{D}$	Distribution of routing instances.
t	Column-generation iteration index.
T	Number of CG decision iterations before termination.
P_t	Pricing problem at CG iteration t.
$N_t(g_t)$	Number of labels explored during pricing at iteration t under extension-ordering rule g_t.
N_{scale}	Fixed global normalization constant used to normalize labeling effort.
Z_t	RMP objective value before the column insertion at iteration t.
Z_1	Initial RMP objective value of the CG episode.
$\mathcal{G}$	Set of candidate extension-ordering rules.
g_t	Extension-ordering action selected at iteration t, with $g_t \in \mathcal{G}$.
$\mathcal{C}_t$	Candidate-column set returned by pricing at iteration t.
M	Maximum number of candidate columns retained after pricing.
k	Target number of columns selected for insertion at each CG iteration.
O_t	Selected column set at iteration t, with $O_t \subseteq \mathcal{C}_t$ and $\lvert O_t \rvert = k$.
π^P	Extension-ordering policy.
π^C	Column-selection policy.
s_t^P	Pre-pricing state defined by the current RMP at iteration t.
s_t^C	Post-pricing state after applying g_t and generating $\mathcal{C}_t$.
h_t	Learned representation of the pre-pricing state, $h_t = \text{GNN}_\theta(s_t^P)$.
$\widetilde{h}_t$	Learned post-pricing representation, $\widetilde{h}_t = \text{Enc}_\theta^C(h_t, \mathcal{C}_t)$.
$z_{t,i}$	Relation-aware representation of candidate column $i \in \mathcal{C}_t$.
$\omega^P(h_t, g)$	Prototype-based support score of extension-ordering action g at state h_t.
$u^P(h_t, g)$	Support uncertainty, $u^P(h_t, g) = 1 - \omega^P(h_t, g)$.
κ_P	Coefficient controlling the support-uncertainty penalty during extension-ordering selection.
u_i	Unary value assigned to candidate column i.
v_{ij}	Symmetric pairwise compatibility value between candidate columns i and j.
$\Delta_k(i \mid O)$	Marginal value of adding candidate i to the current selected set O.
r_t^P	Pricing-stage reward based on normalized labeling effort.
r_t^C	Column-selection reward based on the RMP objective improvement.
β	Coefficient controlling the strength of the RMP-improvement shaping reward.
$Q_\psi^P(h_t, g)$	Action-value function for extension-ordering action g.
$Q_{\phi,k}^C(\widetilde{h}_t, O)$	Relation-conditioned value of selected column set O with cardinality k.
y_t^P	Temporal-difference target for the extension-ordering value function.
y_t^C	Temporal-difference target for the column-selection value function.
$\mathcal{D}_{\text{off}}$	Processed offline dataset of stage-specific CG transitions used for value learning.
done_t	Episode-termination indicator equal to 1 if CG terminates after iteration t, and 0 otherwise.

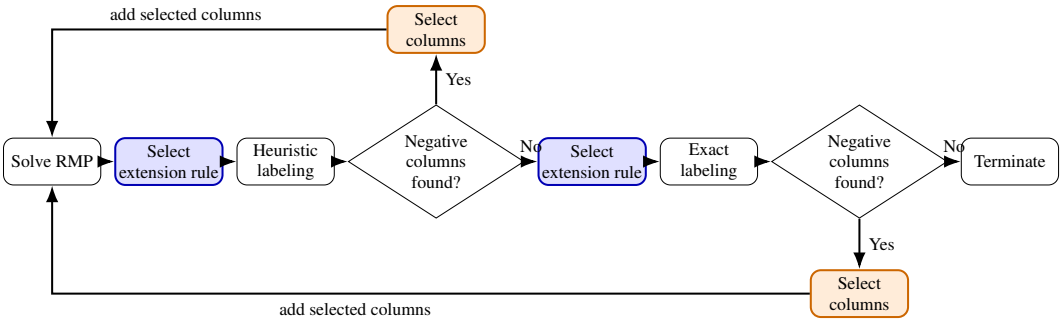


Figure 5: COUPLED-CG framework. The extension-ordering learner guides heuristic and exact labeling, while the column-selection learner selects columns to update the RMP.

C LABELING ALGORITHM

At CG iteration t, the RMP returns dual variables that define the pricing problem P_t. A label ℓ represents a partial route and records its current end node $i(\ell)$, visited customer set $V(\ell)$, accumulated resources, and reduced cost $\bar{c}(\ell)$. Its feasible extension set is

$$\mathcal{F}(\ell) = \{j \in \mathcal{N} \setminus V(\ell) : \ell \oplus j \text{ is feasible}\},$$

where feasibility is determined by the capacity, time-window, and elementarity constraints. Dominance removes labels that cannot lead to a better feasible completion. The extension-ordering rule g_t determines which active label is processed next without changing the pricing formulation, feasible extensions, reduced-cost calculation, or dominance rules.

Exact labeling. Exact labeling is the exact pricing procedure. It processes active nondominated labels until the active label pool is empty and therefore certifies whether any negative-reduced-cost column remains. The extension-ordering rule may affect the number and order of labels explored, but not the optimal reduced-cost value returned.

Algorithm 1 Exact labeling under extension ordering

Require: Dual variables and extension-ordering rule g_t
Ensure: Candidate column set $\mathcal{C}_t$

```
Initialize active label pool L_t and C_t ← ∅
while L_t ≠ ∅ do
    Select and remove the highest-priority label ℓ under g_t
    if ℓ is a complete feasible route then
        if c̄(ℓ) < 0 then
            C_t ← C_t ∪ {r(ℓ)}
        end if
    else
        for each j ∈ F(ℓ) do
            Generate ℓ' ← ℓ ⊕ j
            if ℓ' is not dominated then
                Remove labels dominated by ℓ'
                Add ℓ' to L_t
            end if
        end for
    end if
end while
return C_t
```

Early-stop labeling. Early-stop labeling is used as the heuristic pricing procedure in this paper. It follows the same extension, feasibility, reduced-cost, and dominance rules as exact labeling, but

terminates once a prescribed stopping criterion Γ is satisfied, reaching a labeling budget. Hence, the extension-ordering rule can affect both the number of labels explored and the candidate columns discovered before termination.

Algorithm 2 Early-stop labeling under extension ordering

Require: Dual variables, extension-ordering rule g_t, stopping criterion Γ
Ensure: Candidate column set $\mathcal{C}_t$

```
Initialize active label pool L_t and C_t ← ∅
while L_t ≠ ∅ and Γ is not satisfied do
    Select and remove the highest-priority label ℓ under g_t
    if ℓ is a complete feasible route then
        if c̄(ℓ) < 0 then
            C_t ← C_t ∪ {r(ℓ)}
        end if
    else
        for each j ∈ F(ℓ) do
            Generate ℓ' ← ℓ ⊕ j
            if ℓ' is not dominated then
                Remove labels dominated by ℓ'
                Add ℓ' to L_t
            end if
        end for
    end if
end while
return C_t
```

Other heuristic labeling strategies can also be combined with early stopping. In this paper, we focus on early-stop labeling so that heuristic and exact labeling share the same underlying labeling mechanism and differ only in their stopping condition.

D BIPARTITE GRAPH REPRESENTATION

Node features used for the capacitated vehicle routing problem with time windows (CVRPTW):

1. **Column node features $\mathcal{X}$:** Feature (a) and (c) relate to solving the RMP problem as they all contain information about decision variables in RMP. Each column node corresponds to one decision variable. Features (b) and (d) are determined by the problem formulation of each vehicle routing problem instance, while features (e)-(h) correspond to the dynamic information of each column entering and leaving the basis.
   (a) **Reduced cost:** Reduced cost is a quantity measuring the net benefit of adding a route to the current solution: it compares the route's actual operating cost with the value gained from serving customers that are currently expensive to cover, while also accounting for the opportunity cost of using one vehicle. A route has a negative reduced cost only if the benefit of covering those customers outweighs both its travel cost and the scarcity of available vehicles.
   (b) **Connectivity of column node:** Total number of constraint nodes each column node connects to. As each constraint represents a particular customer, this node feature indicates the ability of a column node (a feasible route) to serve that customer. It also indicates the connectivity of each column node in the bipartite graph representing the state.
   (c) **Solution value:** The solution value of each decision variable corresponding to each column node after solving the RMP in the current iteration. For each column node, this feature is a continuous number greater than or equal to 0. The candidate column nodes have this feature set to 0.
   (d) **Route cost:** The route cost represents the total operational cost incurred by executing a specific route, typically including travel distance or time as well as any penalties for violating time windows, waiting, or other service-related considerations. In the

CVRPTW context, it reflects the true resource consumption of assigning one vehicle to serve the sequence of customers defined by that route.

(e) **Number of iterations that each column node stays in the basis:** If the column node stays in the basis for a long time, it is most likely that the route corresponding to this column node is really good.

(f) **Number of iterations that each column node stays out of the basis:** If the column node stays out of the basis for a long time, it is most likely that it never enters the basis and is not used in the optimal solution in future iterations.

(g) **If the column leaves the basis on the last iteration or not:** This is a binary feature recording the dynamics of each column node.

(h) **If the column enters the basis on the last iteration or not:** Similar binary feature as (g).

2. **Constraint node features $\mathcal{Y}$:** Each constraint node corresponds to one constraint in RMP, so the number of constraint nodes is fixed for each vehicle routing problem instance.

(a) **Dual value:** The dual value or shadow price is the coefficient of each dual variable in the sub-problem objective function, and as each constraint node corresponds to one dual variable, we record the dual value as one feature for a constraint node.

(b) **Connectivity of constraint node:** Total number of column nodes each constraint node connects to, which also indicates the connectivity of each constraint node in the bipartite graph representing the state.

(c) **Instance information:** Customer-related attributes including demand, service duration, time window bounds (earliest and latest allowable service times), and spatial location represented by node coordinates (x, y).

In addition to node features, the bipartite graph representation also incorporates edge features $\mathcal{E}$ to characterize the relationships between column nodes and constraint nodes:

- **Incidence**: Always equal to 1.0 for an existing edge. This feature simply indicates that the corresponding column participates in the associated constraint.
- **Constraint type**: Encoded as 1.0 if the edge connects to the convexity constraint, and 0.0 otherwise. This feature allows the model to distinguish between customer-coverage constraints and the convexity constraint.
- **Customer ID**: The index of the customer associated with a customer-coverage constraint. For edges connected to non-customer constraints, such as convexity constraints, this feature is set to 0.

E Proof of Proposition 1

Proof 1 *For a CG trajectory with T transitions, the cumulative pricing reward is*

$$\sum_{t=1}^{T} r_t^P = -\frac{1}{N_{\text{scale}}} \sum_{t=1}^{T} N_t(g_t). \tag{15}$$

Since $N_{\text{scale}} > 0$ is fixed across policies, maximizing this term is equivalent to minimizing cumulative labeling effort.

The cumulative column-selection reward telescopes as

$$\sum_{t=1}^{T} r_t^C = \frac{\beta}{\max\{1, |Z_1|\}} \sum_{t=1}^{T} (Z_t - Z_{t+1}) = \beta \frac{Z_1 - Z_{T+1}}{\max\{1, |Z_1|\}}. \tag{16}$$

All policies start from the same initial RMP, so Z_1 is fixed. Under exact CG termination, no negative-reduced-cost column remains; hence, Z_{T+1} equals the optimal objective of the full LP relaxation and is also policy-independent. Therefore, the cumulative column-selection reward is constant with respect to the learned policies.

Combining the two reward components gives

$$\sum_{t=1}^{T}(r_t^P + r_t^C) = -\frac{1}{N_{\text{scale}}}\sum_{t=1}^{T} N_t(g_t) + \beta\frac{Z_1 - Z_{T+1}}{\max\{1, |Z_1|\}}. \tag{17}$$

The second term is policy-independent, yielding

$$\arg\max_{\pi^P,\pi^C}\sum_{t=1}^{T}(r_t^P + r_t^C) = \arg\min_{\pi^P,\pi^C}\sum_{t=1}^{T} N_t(g_t). \tag{18}$$

Thus, under undiscounted return and exact termination, the RMP-improvement reward provides dense shaping without changing the optimizer of $\mathcal{M}$.

F Offline Dataset Construction

We construct the offline dataset by running randomized column-generation trajectories on sampled instances from the C family of the Solomon CVRPTW benchmark. The resulting dataset contains 300 instances. Each instance forms one complete episode with a terminal record. We use a candidate-pool size of $M = 10$ and select at most $k = 5$ columns per iteration.

Behavior trajectory generation. For each training instance, we run a randomized CG behavior policy and record one transition per CG iteration. At iteration t, the RMP is first solved and its current state s_t is recorded. An extension-ordering rule $\widehat{g}_t \in \mathcal{G}$ is then sampled and used for pricing. Heuristic labeling is applied first, with exact labeling used when no improving route is found. The resulting novel negative-reduced-cost routes form the candidate set $\widehat{\mathcal{C}}_t$.

The candidate routes are ordered by reduced cost. The most negative-reduced-cost route is retained as the first selected column, and the remaining columns are sampled sequentially without replacement, yielding the logged selection sequence $\widehat{\boldsymbol{O}}_t = (\widehat{i}_{t,1}, \ldots, \widehat{i}_{t,k})$. The corresponding selected set $\widehat{O}_t = \{\widehat{i}_{t,1}, \ldots, \widehat{i}_{t,k}\}$ is then added to the RMP, which is reoptimized to obtain s_{t+1}. If pricing produces no improving route or no column can be added, the transition is marked terminal.

Logged transition. Each CG iteration produces an offline raw transition

$$\left(s_t, \widehat{g}_t, \widehat{\mathcal{C}}_t, \widehat{O}_t, m_t, s_{t+1}, \text{done}_t\right), \tag{19}$$

where $\widehat{g}_t$ is the logged extension-ordering action, $\widehat{\mathcal{C}}_t$ is the generated candidate set, $\widehat{O}_t$ is the logged selected subset, and done_t indicates whether the CG trajectory terminates.

The raw metric vector is

$$m_t = (N_t^{\text{pre}}, N_t^{\text{cur}}, Z_t^{\text{pre}}, Z_t^{\text{cur}}), \tag{20}$$

where N_t^{pre} and N_t^{cur} denote the recorded label-extension counts before and during the current pricing update, and Z_t^{pre} and Z_t^{cur} denote the RMP objective values before and after the corresponding RMP update. These measurements are used to construct the stage-specific rewards during offline training.

During dataset aggregation, an entire source instance is discarded if either Z_t^{pre} or Z_t^{cur} contains a missing, non-numeric, NaN, or infinite value.

Reward construction. For the pricing stage, we use the current labeling effort N_t^{cur} and define

$$r_t^P = -\frac{N_t^{\text{cur}}}{N_{\text{scale}}}, \tag{21}$$

where

$$N_{\text{scale}} = \frac{1}{|\mathcal{I}_{\text{off}}|}\sum_{\mathcal{I}\in\mathcal{I}_{\text{off}}} N(\mathcal{I}), \tag{22}$$

$\mathcal{I}_{\text{off}}$ denotes the set of offline training instances, and $N(\mathcal{I})$ is the total labeling effort over the complete CG trajectory of instance $\mathcal{I}$. Thus, N_{scale} is the average total labeling effort across the offline episodes.

For the column-selection stage, we use the RMP objective improvement induced by the selected columns:

$$r_t^C = \beta \frac{Z_t^{\mathrm{pre}} - Z_t^{\mathrm{cur}}}{\max\{1, |Z_1|\}}, \tag{23}$$

where Z_1 is the initial RMP objective and $\beta \geq 0$ scales the objective-improvement reward.

State and candidate features. The RMP state is represented as a bipartite graph between column and constraint nodes. Column-node features include reduced cost, solution value, route cost, connectivity, and basis-history information, while constraint-node features include dual value, connectivity, customer demand, time-window information, service time, and spatial coordinates. Edge features describe RMP incidence and constraint type.

Each candidate route is represented by its reduced cost, route cost, number of customers, total demand, route duration, normalized candidate rank, and overlap with the current RMP. For relation-aware column selection, we additionally construct a complete candidate-relation graph using pairwise customer overlap, arc overlap, cosine distance, and Jaccard distance.

G Resource-Based Priority Rules

We consider five extension-ordering rules:

$$\{\text{minimum reduced cost}, \text{minimum load}, \text{minimum time}, \text{maximum load}, \text{maximum time}\}.$$

Each rule reflects a different search preference in the labeling process.

- **Minimum reduced cost:** prioritizes partial routes that are currently most promising with respect to the pricing objective, with the goal of discovering negative-reduced-cost routes earlier.
- **Minimum load:** prioritizes labels with lower accumulated load, leaving more residual vehicle capacity and therefore more flexibility for future customer extensions.
- **Maximum load:** prioritizes labels with higher accumulated load, encouraging the search toward more resource-saturated and potentially near-complete routes.
- **Minimum time:** prioritizes labels with smaller accumulated time, preserving more remaining temporal flexibility for feasible future extensions.
- **Maximum time:** prioritizes labels with larger accumulated time, favoring deeper routes that are closer to exhausting their temporal resources or completing the route.

These rules modify only the order in which active labels are processed. Under exact labeling, they preserve pricing optimality while potentially changing the labeling effort; under early-stop labeling, they can additionally affect the set of improving columns discovered before termination.

H Motivation experiments

We conduct preliminary experiments on 672 Solomon CVRPTW instances (Solomon, 1987) to examine the effect of extension-ordering strategies. We compare five fixed and five random strategies.[1]

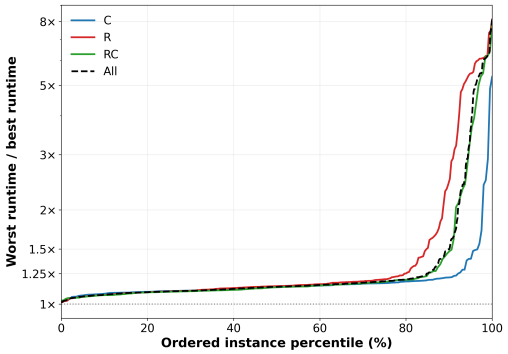
Figure 6(a) reports, for each instance, the ratio between the largest and smallest runtimes obtained by the ten strategies, ordered by percentile and shown separately for the C, R, and RC classes as well as for all instances. The effect of priority-rule selection varies substantially across instances. While the runtime differences are small for many cases, nearly 20% of the instances exhibit a worst-to-best runtime ratio above 1.25, and the largest ratios approach eight. This indicates that selecting an appropriate extension-ordering rule can have a substantial impact on pricing efficiency.

Figure 6(b) further shows that no single strategy is consistently best. Each strategy achieves the minimum runtime on some instances, while even the most frequently best strategy accounts for only

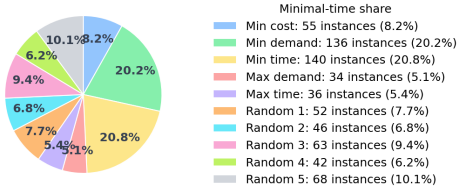
[1] A fixed strategy uses the same extension-ordering rule throughout all CG iterations of an instance, whereas a random strategy samples an extension-ordering rule at each CG iteration.

about 21% of the instances. Together, the large within-instance runtime variation and the absence of a uniformly dominant rule motivate selecting the extension-ordering strategy adaptively according to the current CG state.

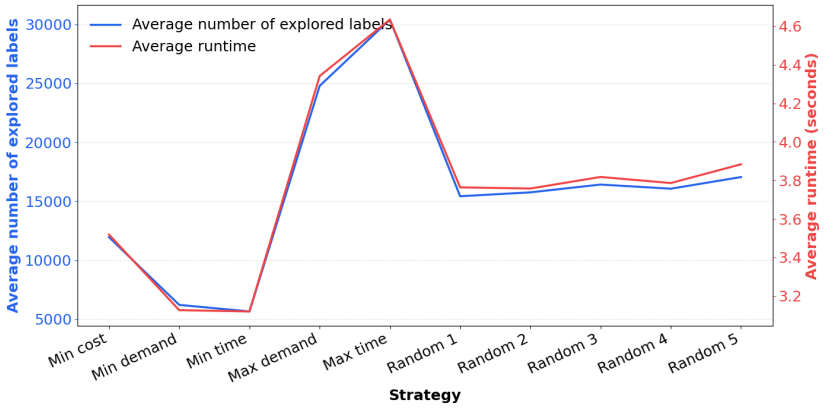
Figure 6(c) compares, across the ten strategies, the average number of explored labels and the average runtime. The two curves exhibit closely aligned trends, indicating that the explored label count tracks the actual computational effort of pricing well. This supports using $N_t(g_t)$ as a practical surrogate for the pricing computational cost $C(P_t)$.



(a) Ordered worst-to-best runtime ratio across extension-ordering strategies.



(b) Fraction of instances for which each strategy achieves the minimum runtime.



(c) Average explored labels and average runtime under different extension-ordering strategies.

Figure 6: Empirical motivation for adaptive extension-ordering selection. Panel (a) shows the instance-wise runtime variation across the ten candidate strategies, Panel (b) shows that no single strategy dominates across instances, and Panel (c) shows that the average number of explored labels closely tracks the average runtime across strategies. These results indicate both substantial potential gains from appropriate rule selection and support the use of explored-label count as a surrogate for pricing computational effort.

I Prototype-Based Local Action Support

For each extension-ordering rule $g \in \mathcal{G}$, we maintain an action-specific prototype ξ_g in the learned state-embedding space. The prototype summarizes the offline states for which rule g is observed. Given the normalized embedding

$$\bar{h}_j = \frac{h_j}{\|h_j\|_2 + \epsilon}, \tag{24}$$

the prototype is initialized from the normalized centroid of the corresponding logged states and updated during training by an exponential moving average. For a minibatch $\mathcal{B}$, let

$$\mathcal{B}_g = \{j \in \mathcal{B} : \widehat{g}_j = g\}, \tag{25}$$

and define

$$\mu_g^{\mathcal{B}} = \frac{\frac{1}{|\mathcal{B}_g|}\sum_{j\in\mathcal{B}_g} \bar{h}_j}{\left\|\frac{1}{|\mathcal{B}_g|}\sum_{j\in\mathcal{B}_g} \bar{h}_j\right\|_2 + \epsilon}. \tag{26}$$

When $\mathcal{B}_g \neq \emptyset$, the prototype is updated as

$$\widetilde{\xi}_g = (1-\rho)\xi_g + \rho\mu_g^{\mathcal{B}}, \qquad \xi_g \leftarrow \frac{\widetilde{\xi}_g}{\|\widetilde{\xi}_g\|_2 + \epsilon}, \tag{27}$$

where $\rho \in (0, 1]$ is the prototype update rate.

For a current state representation h_t, we normalize

$$\bar{h}_t = \frac{h_t}{\|h_t\|_2 + \epsilon} \tag{28}$$

and compute its similarity to each action prototype using

$$\widetilde{\omega}_P(h_t, g) = \exp\left(-\frac{\|\bar{h}_t - \xi_g\|_2^2}{2\sigma^2}\right)\mathbf{1}\{n_g > 0\}, \tag{29}$$

where $\sigma > 0$ controls the similarity bandwidth and n_g denotes the number of logged training samples associated with rule g. The similarities are normalized across extension-ordering rules:

$$\omega_P(h_t, g) = \frac{\widetilde{\omega}_P(h_t, g)}{\sum_{g'\in\mathcal{G}} \widetilde{\omega}_P(h_t, g') + \epsilon}. \tag{30}$$

A larger $\omega_P(h_t, g)$ indicates stronger relative support for rule g around the current state representation. We therefore define

$$u_P(h_t, g) = 1 - \omega_P(h_t, g) \tag{31}$$

as the corresponding support uncertainty used in both conservative training and support-adjusted action selection.

J Relation-Conditioned Marginal Column Selection

This section provides implementation details for the relation-conditioned marginal column-selection policy introduced in the main text.

Candidate relation graph. For each pricing call, the generated candidate set $\mathcal{C}_t = \{c_1, \ldots, c_M\}$ is represented as a complete relation graph. Each candidate c_i is initialized with its route-level features, including reduced cost, route cost, number of customers, total demand, route duration, normalized rank, and overlap with the current RMP. For each pair (c_i, c_j), we construct an edge feature vector

$$e_{ij} = \left[o_{ij}^{\text{cust}}, o_{ij}^{\text{arc}}, d_{ij}^{\cos}, d_{ij}^{\text{Jac}}\right], \tag{32}$$

where the four components denote customer overlap, arc overlap, cosine distance, and Jaccard distance, respectively. These relations provide information about redundancy and complementarity among generated columns.

Relation-aware encoding and value heads. An edge-aware graph encoder propagates information over the candidate-relation graph and produces a representation $z_{t,i}$ for each candidate. The pre-pricing representation h_t is incorporated through Enc_θ^C, yielding the post-pricing representation $\widetilde{h}_t$. A unary head assigns an individual value

$$u_i = f_u(z_{t,i}, \widetilde{h}_t), \tag{33}$$

while a symmetric pairwise head evaluates the interaction between candidates,

$$v_{ij} = \frac{1}{2}\left[f_v(z_{t,i}, z_{t,j}, e_{ij}) + f_v(z_{t,j}, z_{t,i}, e_{ji})\right], \tag{34}$$

so that $v_{ij} = v_{ji}$.

Marginal selection. Selection starts from the mandatory minimum reduced cost column. At each subsequent step, the marginal value $\Delta_k(i \mid O)$ defined in the main text is evaluated for every unused candidate, and the candidate with the largest marginal value is added to O. The procedure continues until k columns have been selected or no unused candidate remains.

Unlike an explicit subset-action representation, which requires evaluating up to $\binom{M}{k}$ candidate subsets, marginal selection evaluates individual remaining candidates sequentially. Given precomputed unary and pairwise values, selecting k columns requires at most $O(kM)$ marginal comparisons.

Variable candidate-pool and selection sizes. The policy does not associate actions with predefined subset indices. Instead, unary and pairwise values are computed directly from the candidates present in the current pricing call. Consequently, the same network parameterization can be applied when the candidate-pool size M or target selection cardinality k changes, without redefining the output action space.

K Learning Details

Temporal-difference objectives. The two value functions are trained using the coupled Bellman targets defined in Section 4.2. For the extension-ordering stage, the temporal-difference loss is

$$\mathcal{L}^P_{\mathrm{TD}} = \mathbb{E}_{\mathcal{D}_{\mathrm{off}}}\left[\left(Q^P_\psi(h_t, \widehat{g}_t) - y^P_t\right)^2\right]. \tag{35}$$

For the column-selection stage, the logged selected set is evaluated using the relation-conditioned set value:

$$\mathcal{L}^C_{\mathrm{TD}} = \mathbb{E}_{\mathcal{D}_{\mathrm{off}}}\left[\left(Q^C_{\phi,k}(\widetilde{h}_t, \widehat{O}_t) - y^C_t\right)^2\right], \tag{36}$$

where

$$k = \min\{k, |\widehat{\mathcal{C}}_t|\}. \tag{37}$$

Conservative regularization for extension ordering. For the extension-ordering stage, unreliable counterfactual values are regularized using

$$\mathcal{L}^P_{\mathrm{CCQL}} = \mathbb{E}_{\mathcal{D}_{\mathrm{off}}}\left[\log \sum_{g \in \mathcal{G}} \exp\left(Q^P_\psi(h_t, g) + \kappa_P u_P(h_t, g)\right) - Q^P_\psi(h_t, \widehat{g}_t)\right]. \tag{38}$$

The uncertainty term $u_P(h_t, g)$ places stronger conservative pressure on high values assigned to weakly supported actions. The extension-ordering loss is

$$\mathcal{L}^P = \mathcal{L}^P_{\mathrm{TD}} + \lambda_P \mathcal{L}^P_{\mathrm{CCQL}}, \tag{39}$$

where $\lambda_P \geq 0$ controls the strength of conservative regularization.

Conservative regularization for marginal column selection. The column-selection policy constructs a set through sequential marginal decisions rather than treating all subsets as explicit actions. We therefore apply conservative regularization to the same marginal decisions used by the policy. For the logged selection sequence $\widehat{\boldsymbol{O}}_t$ defined in Appendix F, let

$$\widehat{O}_{t,\ell} = \left\{\widehat{i}_{t,1}, \ldots, \widehat{i}_{t,\ell}\right\} \tag{40}$$

denote the set of columns selected within the first ℓ marginal decisions. For $k > 1$, the conservative loss is

$$\mathcal{L}^C_{\mathrm{MCR}} = \mathbb{E}_{\mathcal{D}_{\mathrm{off}}}\left[\frac{1}{k-1}\sum_{\ell=1}^{k-1}\left(\log \sum_{i \in \widehat{\mathcal{C}}_t \setminus \widehat{O}_{t,\ell}} \exp\left(\Delta_k(i \mid \widehat{O}_{t,\ell})\right) - \Delta_k\left(\widehat{i}_{t,\ell+1} \mid \widehat{O}_{t,\ell}\right)\right)\right]. \tag{41}$$

For $k = 1$, we set $\mathcal{L}^C_{\mathrm{MCR}} = 0$. This regularization suppresses unsupported high marginal values while preserving the values of logged marginal choices, without enumerating the $\binom{M}{k}$ possible subsets.

The column-selection loss is

$$\mathcal{L}^C = \mathcal{L}^C_{\mathrm{TD}} + \lambda_C \mathcal{L}^C_{\mathrm{MCR}}, \tag{42}$$

where $\lambda_C \geq 0$ controls the strength of conservative regularization for marginal column selection.

Coupled alternating training. Because y_t^P depends on the target column-selection network and y_t^C depends on the target extension-ordering network, the two value functions are trained within one coupled offline-learning procedure. For each minibatch, the target column-selection policy constructs

$$O_t^{\bar{\phi}} = \text{MarginalSelect}\left(\widetilde{h}_t, \widehat{\mathcal{C}}_t, k; \bar{\phi}\right), \tag{43}$$

using the marginal rule in Section 4.3. Its target value is

$$V_{\bar{\phi}}^C(\widetilde{h}_t) = Q_{\bar{\phi},k}^C\left(\widetilde{h}_t, O_t^{\bar{\phi}}\right). \tag{44}$$

The extension-ordering target is

$$y_t^P = r_t^P + (1 - \text{done}_t)V_{\bar{\phi}}^C(\widetilde{h}_t). \tag{45}$$

After the selected columns update the RMP, the target extension-ordering value for the next iteration is

$$V_{\bar{\psi}}^P(h_{t+1}) = \max_{g \in \mathcal{G}}\left[Q_{\bar{\psi}}^P(h_{t+1}, g) - \kappa_P u_P(h_{t+1}, g)\right], \tag{46}$$

and the column-selection target is

$$y_t^C = r_t^C + (1 - \text{done}_t)V_{\bar{\psi}}^P(h_{t+1}). \tag{47}$$

The extension-ordering and column-selection networks are updated alternately using $\mathcal{L}^P$ and $\mathcal{L}^C$, respectively.

The target-network parameters are updated by Polyak averaging:

$$\bar{\psi} \leftarrow \tau_P\psi + (1 - \tau_P)\bar{\psi}, \qquad \bar{\phi} \leftarrow \tau_C\phi + (1 - \tau_C)\bar{\phi}, \tag{48}$$

where $\tau_P, \tau_C \in (0, 1]$ denote the corresponding target-update rates. The shared state encoder is updated through both stage-specific losses during alternating training.

The complete training and inference procedure is summarized in Algorithm 3.

L Dataset Preparation for Learning-Based Baselines

To ensure a controlled comparison, all learning-based methods are trained using the same pool of small C-family Solomon CVRPTW instances used to train COUPLED-CG. Thus, the methods receive the same instance-level training information, while retaining their original learning procedures. All evaluation instances are held out from training.

FFCG. We adapt Hu et al. (2025) to our CVRPTW column-generation implementation. The policy interacts with the restricted master problem and pricing procedure on the training instances, and the resulting transitions are stored in a replay buffer for DQN training with a target network, following the original learning framework.

Yuan CG. We adapt the PPO-based multiple-column selection method of Yuan et al. (2024) to the same CVRPTW environment. The policy is trained on-policy through interaction with column generation. At each iteration, it observes the generated candidate columns and selects columns to enter the restricted master problem. For consistency with the main experimental setting, we use a candidate pool of size $M = 10$ and select $k = 5$ columns.

Morabit. For Morabit et al. (2021), we construct the supervised training data from the same training instances. An expert column-generation procedure is used to generate column-selection labels and the corresponding bipartite graph representations, which are then used to train the supervised classifier.

Summary. All learning-based baselines therefore use the same CVRPTW training-instance pool as COUPLED-CG, while preserving their respective learning paradigms: replay-based DQN training for FFCG, on-policy PPO training for Yuan CG, and expert-labeled supervised learning for Morabit.

Algorithm 3 Coupled Two-Stage Offline Conservative Q-Learning

Require: Offline dataset $\mathcal{D}_{\text{off}}$, strategy set $\mathcal{G}$, target selection size k.
Ensure: Trained extension-ordering and column-selection policies.

Offline training

Initialize the shared encoder, Q_ψ^P, Q_ϕ^C, and target networks.
for each training iteration **do**
  Sample minibatch $\mathcal{B} \subseteq \mathcal{D}_{\text{off}}$ and construct r_t^P, r_t^C from m_t.
  Encode $h_t = \text{GNN}_\theta(s_t^P)$ and $\widetilde{h}_t = \text{Enc}_\theta^C(h_t, \widehat{\mathcal{C}}_t)$.
  Compute $u_P(h_t, g)$ for all $g \in \mathcal{G}$ and set $k = \min\{k, |\widehat{\mathcal{C}}_t|\}$.
  Construct $O_t^{\bar{\phi}} = \text{MarginalSelect}(\widetilde{h}_t, \widehat{\mathcal{C}}_t, k; \bar{\phi})$ when $\widehat{\mathcal{C}}_t \neq \emptyset$.
  Compute the coupled pricing target y_t^P and update θ, ψ by minimizing $\mathcal{L}^P = \mathcal{L}_{\text{TD}}^P + \lambda_P \mathcal{L}_{\text{CCQL}}^P$.
  if $\widehat{\mathcal{C}}_t \neq \emptyset$ **then**
    Encode $h_{t+1} = \text{GNN}_\theta(s_{t+1}^P)$.
    Compute $V_{\bar{\psi}}^P(h_{t+1}) = \max_{g \in \mathcal{G}}[Q_{\bar{\psi}}^P(h_{t+1}, g) - \kappa_P u_P(h_{t+1}, g)]$.
    Compute the coupled column-selection target y_t^C and update θ, ϕ by minimizing $\mathcal{L}^C = \mathcal{L}_{\text{TD}}^C + \lambda_C \mathcal{L}_{\text{MCR}}^C$.
  end if
  Update target networks by Polyak averaging.
end for

Online inference

Initialize the RMP with feasible columns.
while true **do**
  Solve the RMP and compute $h_t = \text{GNN}_\theta(s_t^P)$.
  Select $g_t = \arg\max_{g \in \mathcal{G}}[Q_\psi^P(h_t, g) - \kappa_P u_P(h_t, g)]$.
  Run heuristic labeling under g_t to obtain $\mathcal{C}_t$; invoke exact labeling if $\mathcal{C}_t = \emptyset$.
  if $\mathcal{C}_t = \emptyset$ **then**
    break
  end if
  Compute $\widetilde{h}_t = \text{Enc}_\theta^C(h_t, \mathcal{C}_t)$ and $k = \min\{k, |\mathcal{C}_t|\}$.
  $O_t = \text{MarginalSelect}(\widetilde{h}_t, \mathcal{C}_t, k; \phi)$.
  Add O_t to the RMP.
end while
return Final RMP solution.

M Implementation Details

This section reports the architecture, optimization settings, and evaluation protocol used in our experiments. All methods share the same column-generation backbone, including the RMP formulation, pricing formulation, exact-pricing fallback, termination criterion, and final integer-master procedure. They differ only in the extension-ordering and/or column-selection policies.

M.1 Offline Training Setup

Both learning branches are trained using the offline trajectories described in Appendix F. Complete episodes are split at the episode level into 90% training and 10% validation data to avoid leakage across transitions. Feature-normalization statistics are computed using the training split only and are fixed during validation and inference. Both the training and evaluation instances are sampled from the original 100-customer Solomon CVRPTW instances by constructing smaller subinstances. The training and test sets are sampled independently, with no overlap between their instances.

We use a batch size of 32. At each pricing call, at most $M = 10$ candidate columns are retained and at most $k = 5$ columns are inserted into the RMP. When fewer than k valid candidates are available, all valid candidates are used.

M.2 Shared RMP State Encoder

The current RMP is represented by the bipartite graph defined in Appendix D, with column nodes, constraint nodes, and RMP-incidence edges. The shared encoder GNN_θ is implemented as a two-layer bipartite message-passing network with hidden dimension 256. It outputs a 128-dimensional graph representation

$$h_t = \mathrm{GNN}_\theta(s_t^P), \tag{49}$$

and uses dropout rate 0.1. The same RMP encoder is shared by the extension-ordering and column-selection branches.

The extension-ordering head maps h_t to five action values corresponding to the five rules in $\mathcal{G}$. The prototype-based support estimator described in Appendix I maintains one prototype for each rule. We use prototype update rate $\rho = 0.05$, Gaussian bandwidth $\sigma = 1.0$, and support-adjustment coefficient $\kappa_P = 1.0$.

M.3 Post-Pricing and Relation-Aware Column Encoder

After pricing, improving routes are sorted by reduced cost, with the minimum reduced cost route placed first. Each candidate is represented by seven features: reduced cost, route cost, number of customers, total demand, route duration, normalized candidate rank, and overlap with the current RMP.

Each candidate feature vector is projected to a 128-dimensional embedding. A masked mean of the valid candidate embeddings is concatenated with the shared RMP representation h_t and transformed into the post-pricing representation $\widetilde{h}_t \in \mathbb{R}^{128}$.

To model interactions among candidate columns, we construct a complete relation graph over the valid candidates. Each edge (i, j) contains four features: customer overlap, arc overlap, cosine distance, and Jaccard distance,

$$e_{ij} = \left[e_{ij}^{\mathrm{cust}}, e_{ij}^{\mathrm{arc}}, e_{ij}^{\cos}, e_{ij}^{\mathrm{Jac}}\right]. \tag{50}$$

Self-edges and pairs involving padded candidates are masked.

The relation-aware encoder contains two edge-aware attention layers with four attention heads. For each layer, the relation features contribute to both the attention score and the transmitted message,

$$a_{ij}^{(\ell)} = \frac{\left(W_Q r_i^{(\ell)}\right)^\top \left(W_K r_j^{(\ell)}\right)}{\sqrt{d_{\mathrm{head}}}} + w_E^\top e_{ij}, \tag{51}$$

$$m_{ij}^{(\ell)} = W_V r_j^{(\ell)} + W_{EV} e_{ij}. \tag{52}$$

Masked attention is applied over valid candidates, followed by residual connections, dropout, and layer normalization.

The resulting candidate representations are passed to the unary and symmetric pairwise value heads used by the relation-conditioned marginal policy described in Section 4.3. Column selection then follows the marginal procedure defined there; no explicit subset-action catalogue is constructed.

M.4 Optimization and Training Schedule

Table 4 summarizes the optimization settings.

Training is performed for 60 epochs in three stages. During epochs 1–20, the shared encoder and both decision branches are trained jointly using alternating updates. During epochs 21–40, the shared encoder and column-selection branch are frozen and only the extension-ordering branch is fine-tuned. During epochs 41–60, the shared encoder and extension-ordering branch are frozen and the post-pricing and column-selection networks are fine-tuned.

Model selection is performed using held-out validation instances only; test instances are not used for training or model selection.

Table 4: Default parameter settings used for offline training.

Parameter	Symbol	Value
Batch size	B	32
Learning rate	–	10^{-4}
Weight decay	–	10^{-5}
Gradient-norm clipping	–	5
Shared embedding dimension	d_h	128
GNN hidden dimension	–	256
GNN message-passing layers	–	2
Relation-attention layers	–	2
Relation-attention heads	–	4
Dropout rate	–	0.1
Prototype update rate	ρ	0.05
Prototype bandwidth	σ	1.0
Support coefficient	κ_P	1.0
Extension conservative coefficient	λ_P	1.0
Column conservative coefficient	λ_C	1.0
Polyak update rate	τ	0.005
Optimizer	–	AdamW
Random seed	–	123
Heuristic labeling budget	Γ	200,000

M.5 Common Evaluation Protocol

All comparison methods are evaluated using the same RMP and pricing implementations, candidate-generation procedure, feasibility and reduced-cost filters, exact-pricing fallback, termination criterion, candidate-pool size $M = 10$, insertion limit $k = 5$, and time limits=$900s$. Thus, the compared methods differ only in the learned or prescribed decisions being evaluated.

We report both CG time and end-to-end wall-clock time. CG time includes the RMP and pricing computations,

$$T_{\mathrm{CG}} = T_{\mathrm{RMP}} + T_{\mathrm{pricing}}, \tag{53}$$

whereas end-to-end time additionally includes neural inference, process communication, and orchestration overhead. Completion rate is computed over the full intended test set.

N Comparison Methods

We compare Coupled-CG with four learning-based or heuristic baselines and five ablation configurations. Unless otherwise stated, the candidate-pool size is $M = 10$; fixed-cardinality methods insert $k = 5$ columns, whereas FFCG and Morabit use variable-size selection.

Plain CG (Desaulniers et al., 2006). Plain CG uses the fixed minimum reduced cost extension-ordering rule and inserts the k most-negative-reduced-cost columns. No learned component is used.

FFCG (Hu et al., 2025). FFCG uses fixed minimum reduced cost pricing and a DQN-based sequential column selector. Following the original algorithm, we do not impose the fixed cardinality k used by Coupled-CG; candidates are selected one at a time according to marginal Q-values until a learned STOP action is chosen, yielding a variable-size selected set.

Yuan CG (Yuan et al., 2024). Yuan CG uses fixed minimum reduced cost pricing and an on-policy PPO column selector. With $M = 10$, the most-negative-reduced-cost column is always included and the policy selects four additional columns from the remaining nine candidates, giving $\binom{9}{4} = 126$ subset actions and a fixed $k = 5$.

Morabit CG (Morabit et al., 2021). Morabit CG uses fixed minimum reduced cost pricing and a supervised bipartite-graph classifier. Following the original algorithm, we do not restrict the number of inserted columns to k. All candidates whose predicted selection probability exceeds 0.5 are inserted; if none are selected, the most-negative-reduced-cost column is used as a fallback. The selected-set size is therefore variable.

COUPLED-CG. COUPLED-CG jointly uses the learned extension-ordering and relation-aware column-selection policies. The extension policy selects one of five pricing priorities, and the resulting candidate pool is then processed by the column-selection policy. Both policies are trained with the coupled Bellman formulation so that each stage is optimized with respect to the downstream decision.

Pricing Only. Pricing Only uses an extension-ordering policy trained independently, without the coupled Bellman updates. The learned pricing policy selects the extension-ordering rule, while column selection follows Plain CG by inserting the k most-negative-reduced-cost columns.

Column Selection Only. Column Selection Only uses a column-selection policy trained independently, without coupled Bellman updates from the pricing stage. Pricing is fixed to the minimum reduced cost rule, while the independently trained relation-aware selector chooses the k columns to insert.

Pricing + FFCG. This configuration combines the independently trained extension-ordering policy with the FFCG selector. The pricing policy determines the candidate pool, after which FFCG performs variable-size sequential selection.

Pricing + Yuan. This configuration combines the independently trained extension-ordering policy with the Yuan PPO selector. The learned pricing policy determines the candidate pool, and Yuan selects a fixed five-column subset.

Pricing + Morabit. This configuration combines the independently trained extension-ordering policy with the Morabit classifier. The learned pricing policy determines the candidate pool, after which Morabit performs threshold-based variable-size selection.

The Pricing Only and Column Selection Only policies are trained separately and do not use the cross-stage Bellman coupling of COUPLED-CG. The three hybrid configurations retain the same independently trained pricing policy while replacing only the downstream column selector, allowing us to examine how different selectors respond to candidate pools induced by learned pricing.

O DECOUPLED TRAINING FOR COMPONENT COMPARISONS

To distinguish cross-stage value propagation from independent within-stage learning, the pricing-only and column-selection-only variants, the pricing-only and column-selection-only variants retain the corresponding state representations, architectures, rewards, and conservative regularization used in COUPLED-CG, but replace the cross-stage Bellman targets with within-stage targets. All value-based variants use the undiscounted setting $\gamma = 1$.

Independent pricing model. The pricing reward is unchanged:

$$r_t^P = -\frac{N_t}{N_{\text{scale}}}, \tag{54}$$

where N_t is the number of labels explored at iteration t and $N_{\text{scale}} > 0$ is the fixed normalization constant used in COUPLED-CG. The target pricing value at the next pre-pricing state is

$$V_{\bar{\psi}}^P(h_{t+1}) = \max_{g \in \mathcal{G}} \left[Q_{\bar{\psi}}^P(h_{t+1}, g) - \kappa_P u_P(h_{t+1}, g) \right]. \tag{55}$$

Rather than bootstrapping from the column-selection value at the same iteration, the independent pricing model bootstraps directly from the next pricing state:

$$y_{t,\text{ind}}^P = r_t^P + (1 - \text{done}_t) V_{\bar{\psi}}^P(h_{t+1}). \tag{56}$$

Thus, pricing value propagates only across consecutive pricing states,

$$Q_t^P \longleftarrow Q_{t+1}^P,$$

and does not directly bootstrap from Q_t^C.

Independent column-selection model. The column-selection reward is also unchanged:

$$r_t^C = \beta \frac{Z_t - Z_{t+1}}{\max\{1, |Z_1|\}}. \tag{57}$$

For the next post-pricing state $\widetilde{h}_{t+1} = \mathrm{Enc}_\theta^C(h_{t+1}, \mathcal{C}_{t+1})$, the target network greedily constructs

$$\widehat{O}_{t+1} = \mathrm{GreedySelect}_{\bar{\phi}}\left(Q_{\bar{\phi}}^C, \widetilde{h}_{t+1}\right), \tag{58}$$

and evaluates

$$V_{\bar{\phi}}^C(\widetilde{h}_{t+1}) = Q_{\bar{\phi}}^C\left(\widetilde{h}_{t+1}, \widehat{O}_{t+1}\right). \tag{59}$$

The independent column-selection target is therefore

$$y_{t,\mathrm{ind}}^C = r_t^C + (1 - \mathrm{done}_t) V_{\bar{\phi}}^C(\widetilde{h}_{t+1}). \tag{60}$$

Hence, column-selection value propagates only across consecutive column-selection states,

$$Q_t^C \longleftarrow Q_{t+1}^C,$$

rather than bootstrapping from the next pricing value Q_{t+1}^P.

Difference from coupled training. The independent variants therefore use the within-stage propagation

$$Q_t^P \longleftarrow Q_{t+1}^P, \qquad Q_t^C \longleftarrow Q_{t+1}^C, \tag{61}$$

whereas COUPLED-CG uses the cross-stage Bellman structure

$$Q_t^P \longleftarrow Q_t^C \longleftarrow Q_{t+1}^P. \tag{62}$$

In the independent variants, the other decision stage remains part of the logged CG transition dynamics, but its learned value is not used in the Bellman backup. Since the independent and coupled variants use the same undiscounted setting $\gamma = 1$, rewards, and corresponding model components, their distinction is the presence or absence of cross-stage value propagation.

P STRUCTURAL INSTANCE FEATURES

We characterize each CVRPTW instance using six structural properties capturing capacity, demand, spatial, temporal, and transition characteristics. Let $\mathcal{N}$ denote the customer set, $n = |\mathcal{N}|$, q_i the demand of customer i, Q the vehicle capacity, K the number of vehicles, (x_i, y_i) the customer coordinates, $[a_i, b_i]$ the time window, and s_i the service time.

Load tightness. Load tightness measures total demand relative to the available fleet capacity:

$$\mathrm{LT} = \frac{\sum_{i \in \mathcal{N}} q_i}{KQ}. \tag{63}$$

A larger value indicates tighter aggregate vehicle capacity.

Demand variance. Demand variance measures the heterogeneity of customer demands:

$$\mathrm{DV} = \frac{1}{n} \sum_{i \in \mathcal{N}} (q_i - \bar{q})^2, \qquad \bar{q} = \frac{1}{n} \sum_{i \in \mathcal{N}} q_i. \tag{64}$$

Larger values indicate greater variation in customer load requirements.

Spatial spread. Spatial spread measures the dispersion of customer locations along both spatial dimensions:

$$\mathrm{SS} = \sqrt{\mathrm{Var}(x) + \mathrm{Var}(y)}. \tag{65}$$

A larger value indicates a more geographically dispersed customer set.

Time-window overlap. For each unordered customer pair (i, j), we identify whether their time windows have a nonempty intersection:

$$\mathbb{I}_{ij}^{\mathrm{TW}} = \mathbb{I}[\max(a_i, a_j) \le \min(b_i, b_j)]. \tag{66}$$

The instance-level overlap percentage is

$$\mathrm{TWO} = \frac{100}{\binom{n}{2}} \sum_{i<j} \mathbb{I}_{ij}^{\mathrm{TW}}. \tag{67}$$

A larger value indicates stronger overlap among customer service windows.

Normalized time-window-width variance. Let $H = b_0 - a_0$ denote the depot time horizon and $w_i = b_i - a_i$ the width of customer i's time window. We first normalize each width as

$$\widetilde{w}_i = \frac{w_i}{H}, \tag{68}$$

and define

$$\mathrm{TWV} = \frac{1}{n} \sum_{i \in \mathcal{N}} (\widetilde{w}_i - \bar{\widetilde{w}})^2. \tag{69}$$

Larger values indicate greater heterogeneity in relative time-window widths.

Transition density. We construct a directed feasible-transition graph over customers. Let d_{0i} denote the depot-to-customer distance and d_{ij} the distance from customer i to j. The earliest feasible service time at customer i is

$$e_i = \max\{a_i, a_0 + d_{0i}\}. \tag{70}$$

A directed transition $i \to j$ is considered feasible if

$$q_i + q_j \le Q, \qquad e_i + s_i + d_{ij} \le b_j, \tag{71}$$

and customer j can subsequently return to the depot within the depot horizon:

$$\max\{a_j, e_i + s_i + d_{ij}\} + s_j + d_{j0} \le b_0. \tag{72}$$

Let $A_{ij} = 1$ when these conditions hold and $A_{ij} = 0$ otherwise. The transition density is

$$\mathrm{TD} = \frac{1}{n(n-1)} \sum_{i \ne j} A_{ij}. \tag{73}$$

A larger value indicates a denser feasible-transition graph and more possible customer-to-customer continuations.

Q Sensitivity to the Reward Coefficient

Table 5 examines the effect of the reward coefficient β. The main setting $\beta = 1.0$ was fixed before this sweep and was not selected retrospectively based on the best observed performance. Across $\beta \in [0.1, 1.0]$, CG time remains consistently lower than the $\beta = 0$ setting, with reductions ranging from approximately 4% to 9%. The remaining metrics vary only moderately across the tested values, suggesting that the method is not highly sensitive to the precise choice of β within this range. In particular, $\beta = 1.0$ achieves performance comparable to the best settings while preserving the reward formulation used throughout the main experiments.

Table 5: Sensitivity analysis of the reward coefficient β. The setting $\beta = 1.0$, used in the main experiments, is marked in bold. The comparison is reported for sensitivity analysis only; β was not selected retrospectively based on the best performance in this experiment.

β	**Total CG Time (s)**	**Mean CG Time (s)**	**vs.** $\beta = 0$	**Wall Time (s)**	**Labels**	**Iterations**
0.0	189.14	3.940	–	296.22	99,687	493
0.1	179.44	3.738	-5.1%	284.87	101,005	499
0.2	181.66	3.785	-4.0%	286.90	101,575	499
0.3	173.75	3.620	-8.1%	277.58	102,060	496
0.4	175.16	3.649	-7.4%	279.07	101,360	497
0.5	173.52	3.615	-8.3%	277.65	**99,606**	**493**
0.6	175.97	3.666	-7.0%	280.44	101,957	499
0.7	174.42	3.634	-7.8%	278.62	99,735	**493**
0.8	**172.31**	**3.590**	-8.9%	277.63	101,549	496
0.9	178.39	3.716	-5.7%	282.94	101,282	496
1.0	173.82	3.621	-8.1%	278.31	102,008	496

R Computational Environment

All experiments are conducted on a Linux server running Ubuntu 22.04-compatible Linux (kernel 5.15, x86-64). The machine is equipped with two Intel Xeon Gold 5220 processors, providing 36 physical cores and 72 hardware threads in total, with a base frequency of 2.20 GHz and a maximum frequency of 3.90 GHz. The system contains 188 GiB of RAM and one NVIDIA Quadro RTX 6000 GPU with 24 GB of VRAM.

The learning components are implemented in Python using PyTorch with CUDA 12.1. The main Python dependencies include `numpy`, `torch`, `gurobipy`, `pybind11`, `scikit-build-core`, `ninja`, and `PyYAML`. The native components are compiled with GCC/G++ 11.4, CMake 3.22.1, and Ninja 1.10.1.

The RMP and other mathematical optimization components are solved using Gurobi through `gurobipy`. All experiments use the same computational environment and solver configuration across comparison methods to ensure a controlled runtime comparison.