PREDICTIVE DUAL SMOOTHING FOR COLUMN GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Solving large-scale linear programs efficiently is an important challenge in many optimization settings. A key technique is column generation, which alternates between solving the master problem over a restricted subset of the variables, and using a pricing subproblem to identify new variables to add. The pricing subproblem is guided by the dual solution of the current restricted master problem, but oscillations in these dual solutions can substantially slow convergence. Dual stabilization methods address this issue. Dual smoothing is a common stabilization method, which guides the pricing subproblem using a combination of the current dual solution and duals from previous iterations. However, while past dual solutions can stabilize the dual trajectory, they do not necessarily guide pricing towards useful new variables. We therefore introduce *predictive dual smoothing*, which instead combines the current dual solution with a learned prediction of future duals to steer pricing towards variables that are more useful in subsequent iterations. The predictor is trained offline using supervision extracted from standard column generation trajectories and is used only to modify the pricing subproblem's objective function, while exact reduced-cost checks and fallback pricing with the unsmoothed duals preserve correctness. Experiments on cutting stock and generalized assignment problems show that predictive dual smoothing substantially reduces generated columns and wall-clock time relative to standard column generation and existing classical and learned stabilization methods. These gains extend to out-of-distribution instance sizes, and predictive smoothing provides further improvements when combined with strong classical stabilization.

1 INTRODUCTION

Column generation (CG) is a widely used technique for solving large-scale linear programs whose full set of variables is too large to enumerate explicitly. Examples include cutting stock, routing and scheduling problems (Lübbecke & Desrosiers, 2005; Desaulniers et al., 2006). Rather than solving the full problem directly, CG alternates between a restricted master problem (RMP), containing only a subset of decision variables (and thus *columns* in the constraint matrix), and a pricing subproblem, hereafter simply called the pricing problem, that searches for new columns (i.e., variables) whose addition would improve the RMP's objective value. The pricing problem is guided by the dual solution of the current RMP, which determines which columns are attractive to generate next.

The duals have an economic interpretation that explains why they are used for pricing. Each dual value is a 'shadow price' associated with a constraint. It expresses the marginal improvement in the optimal RMP objective obtained if that constraint were relaxed, and thus reflects how costly that constraint is to satisfy using the columns currently available in the RMP. At every iteration t, the pricing problem takes optimal RMP duals $\boldsymbol{\pi}_t$ as input and returns the column that best addresses the needs of the RMP expressed by these duals, by favoring columns that make currently expensive constraints easier to satisfy. From the dual perspective, each column that is not yet in the RMP corresponds to a constraint that is absent from the current dual problem. Pricing identifies the missing column whose corresponding dual constraint is most violated by the current dual solution.

However, the current duals reflect only the current incomplete column set. A dual value may be high only because the columns available so far happen to address the corresponding constraint poorly. As other columns are added, the duals can therefore change substantially from one iteration to the next.

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.

In practice, this can lead to *dual oscillation*, which can cause pricing to repeatedly generate columns that are attractive only under temporary dual prices, unnecessarily enlarging the RMP and slowing convergence to the optimal solution (Desrosiers & Lübbecke, 2005). This has motivated research into *dual stabilization* techniques (Du Merle et al., 1999).

A common stabilization strategy is *dual smoothing* (Wentges, 1997; Neame, 2000; Pessoa et al., 2018). Instead of pricing directly with the current RMP dual $\boldsymbol{\pi}_t$, smoothing combines it with a more stable reference point,

$$\widetilde{\boldsymbol{\pi}}_t = (1-\alpha)\boldsymbol{\pi}_t + \alpha \boldsymbol{h}_t,$$

where α is a hyperparameter controlling the strength of smoothing, and $\boldsymbol{h}_t$ is a reference point constructed from dual values observed in previous CG iterations. This can reduce the sensitivity of pricing to dual oscillations, and in turn improve convergence. However, the duals incorporated through the reference point come from earlier, smaller RMPs, and the needs they express may already have been addressed by columns generated in the meantime. As a result, the reference point can continue to direct pricing towards needs that have already been resolved. Thus, smoothing may reduce oscillation, but does not necessarily direct pricing towards columns that will remain useful as the RMP evolves.

This suggests a different choice of reference point. If the duals that will be encountered later in the CG trajectory were known in advance, they could be used as a forward-looking reference point to smooth the current duals. The resulting pricing signal would still reflect the needs of the current RMP, while also anticipating needs that would emerge in the following iterations. This would both counteract the sensitivity of the pricing problem to short-term dual oscillations and favor columns that remain useful as the RMP evolves. This could accelerate CG convergence significantly. However, when the pricing problem must be solved, future duals are, of course, not yet available.

Our key idea is therefore to predict these future duals from the current CG state and use the predictions as smoothing reference points. We call this *predictive dual smoothing*. Given the current CG state s_t, we predict each component of the dual solution k RMP updates ahead using a shared predictor m_θ, and collect these predictions in $\widehat{\boldsymbol{\pi}}_{t+k}$. Pricing is then performed using smoothed duals

$$\widetilde{\boldsymbol{\pi}}_t^{(k)} = (1-\alpha)\boldsymbol{\pi}_t + \alpha \widehat{\boldsymbol{\pi}}_{t+k}.$$

The prediction horizon k determines how far ahead the smoothing reference looks, while α controls how strongly the prediction influences pricing. The prediction is used only to guide the search for new columns. Any generated column is verified against the current RMP solution, and standard pricing (i.e., with $\alpha = 0$) is used whenever the smoothed pricing step fails to identify a valid improving column. Learning can therefore change the sequence of generated columns without affecting the correctness or optimality guarantees of CG.

We train the future-dual predictor offline from trajectories generated by standard CG on training instances from the target problem class. Each visited state s_t can be paired directly with the dual solution observed at iteration $t + k$, so a single CG trajectory provides many supervised training pairs. Moreover, the same recorded trajectory can be reused for different prediction horizons (e.g., when tuning k) by changing only the target iteration.

Our contributions are:

- We introduce *predictive dual smoothing*, which uses predictions of future RMP dual states as a reference point for smoothed pricing, with the prediction horizon controlling how far ahead the method looks.
- We show that future duals can be learned using simple supervised learning from ordinary CG trajectories, with each trajectory providing many training pairs and reusable supervision across prediction horizons.
- We extensively evaluate predictive dual smoothing on cutting stock and generalized assignment problems, showing substantial reductions in generated columns and runtime over standard CG and existing classical and learned stabilization methods. Our experiments further characterize the effects of prediction horizon, smoothing strength, and training-data size. They also study both in-distribution and out-of-distribution generalization across instance scales, and show that predictive smoothing can even provide additional gains when combined with strong classical stabilization methods.

2 Related Work

The integration of machine learning into CG has received increasing attention in recent years, with learning-based components becoming involved in several parts of the CG loop. A first line of work concerns *column selection*, which first generates a *pool* of solutions to the pricing problem, and subsequently uses a learned classifier to decide which of these columns to add to the RMP. Morabit et al. (2021) use imitation learning to train the classifier, with supervision obtained through a relatively expensive one-step lookahead expert. Subsequent work instead uses reinforcement learning to train the classifier, with the goal of optimizing single-column choices for long-term CG performance rather than immediate improvement (Chi et al., 2022). Later extensions generalize this approach to selecting multiple columns per iteration, either with a fixed (Yuan et al., 2024) or variable (Hu et al., 2025) number of columns. Other approaches intervene inside or around pricing: Shen et al. (2022) predict pricing solutions for graph coloring, Morabit et al. (2023) learn to restrict the network used by routing pricing problems, and Koutecká et al. (2025) consider settings with multiple pricing problems per CG iteration and learn which one to solve first. Other directions include learning to *remove* redundant columns (Fang et al., 2023), and predicting whether sampled columns are likely to appear in an optimal integer solution (Sun et al., 2022).

Most relevant to our work are methods that use learning to influence the duals guiding CG. Babaki et al. (2022) address non-uniqueness of RMP dual solutions and learn, via imitation learning and a differentiable optimization layer, which point on the current dual-optimal face to use for pricing. Kraul et al. (2023) instead predict the optimal full-master dual values for the cutting stock problem and use these predictions as stabilization centers in a conventional box-stabilized CG scheme. Similarly, Shen et al. (2024) predict the eventual optimal dual solution and use it to guide adaptive stabilization for graph coloring. More recently, Fang et al. (2025) use reinforcement learning to output stabilized dual vectors directly for pricing.

Our approach differs from these methods in both the prediction target and how the prediction is used. Rather than selecting among current RMP-optimal duals or predicting the eventual full-master optimum, we predict dual solutions at a finite horizon along the future CG trajectory. These predictions are used directly as smoothing reference points for pricing, rather than as centers of a box-stabilized master. This not only reduces the sensitivity of pricing to short-term dual oscillations, but also biases pricing towards columns that address needs anticipated to arise in subsequent CG iterations. Finally, our predictor uses generic features of the CG state rather than problem-specific representations, allowing predictive smoothing to be applied across different problem classes.

3 Preliminaries: Column Generation

CG solves large linear programs without explicitly enumerating all columns. It maintains an RMP over a subset of columns and uses its dual solution to identify improving columns through a pricing problem.

Illustrative example: cutting stock. We use the cutting stock problem (CSP) as an illustrative example. In the CSP, we are given an unlimited supply of stock rolls of fixed length L and a collection of item types that must be cut from these rolls. Item type $i \in \{1, \dots, n\}$ has length l_i and demand d_i. The goal is to satisfy all demands while using as few stock rolls as possible.

A single stock roll can be cut in many different ways, called *cutting patterns*. For example, one cutting pattern may contain two items of one type and three of another, while a different pattern may contain another combination. We represent pattern p by $x_p = (x_{1p}, \dots, x_{np})^\top \in \mathbb{N}^n$, where x_{ip} denotes the number of items of type i produced by pattern p. A pattern is feasible if the total length of the items assigned to the roll does not exceed L. The set of all feasible patterns is thus $\mathcal{P} = \{x \in \mathbb{N}^n : \sum_{i=1}^n l_i x_i \leq L\}$.

Each feasible pattern defines a master variable λ_p, denoting the number of stock rolls cut according to pattern p. As is standard in CG, we work with the LP relaxation of the resulting formulation, so

$\lambda_p \in \mathbb{R}_+$.[1] The Gilmore-Gomory formulation (Gilmore & Gomory, 1961) is then

$$\min_{\lambda \geq 0} \quad \sum_{p \in \mathcal{P}} \lambda_p \tag{1a}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}} x_{ip} \lambda_p \geq d_i, \qquad i = 1, \ldots, n. \tag{1b}$$

The objective minimizes the number of stock rolls used, while each constraint i ensures that enough items of type i are produced to satisfy demand d_i.

The challenge is that the number of feasible cutting patterns can be enormous. Rather than include all of them at once, CG maintains only a subset $\mathcal{P}_t \subseteq \mathcal{P}$ at iteration t and solves the master problem using only the corresponding variables $\{\lambda_p : p \in \mathcal{P}_t\}$. This restricted problem is the RMP.

Solving the RMP produces not only a primal solution but also a dual solution $\boldsymbol{\pi}_t \in \mathbb{R}^n_+$, with one dual value associated with each demand constraint. These dual values define the objective of the *pricing problem*, which is used to generate new columns. It searches over all feasible cutting patterns for a column that could improve the current RMP solution.

For any pattern p, its *reduced cost* under the current RMP dual is $\bar{c}_p(\boldsymbol{\pi}_t) = 1 - \boldsymbol{\pi}_t^\top x_p$, where 1 is the cost of using one stock roll and $\boldsymbol{\pi}_t^\top x_p$ is the total dual value of the items produced by the pattern. Thus, the reduced cost measures the pattern's cost relative to the value it provides under the current dual prices. Adding a pattern to the RMP can improve its objective whenever the pattern's reduced cost is negative. Equivalently, a pattern with negative reduced cost corresponds to a currently violated constraint in the dual of (1). When CG generates a new pattern, it finds the pattern with the most negative reduced cost by solving

$$x_t^\star \in \arg\max_{x \in \mathbb{N}^n} \left\{ \boldsymbol{\pi}_t^\top x : \sum_{i=1}^{n} l_i x_i \leq L \right\}. \tag{2}$$

If $1 - \boldsymbol{\pi}_t^\top x_t^\star < 0$, the corresponding pattern is added to the RMP and the RMP is solved again. Otherwise, no feasible pattern has negative reduced cost, which certifies that the current RMP solution is optimal for problem (1), and CG is terminated. We refer to this procedure, with exact pricing and one column of minimum reduced cost added per iteration, as *standard CG*.

Dual stabilization and smoothing. Standard CG uses the current RMP dual $\boldsymbol{\pi}_t$ directly to define the pricing objective. However, this dual only reflects the needs of the *current* restricted column set. Some of these needs may disappear once new columns are added. The dual solution can therefore change substantially after each RMP update, causing pricing to repeatedly target temporary needs and generate columns that remain useful only for a small number of iterations. This behavior can lead to slow convergence and has motivated research into *dual stabilization* methods.

Stabilization can intervene at different points in the CG loop. Some stabilization methods modify the RMP so that its dual solution is encouraged to remain near a stability center, as in the box/penalty approach of Du Merle et al. (1999). In contrast, *dual smoothing* leaves the RMP unchanged and instead modifies the dual vector used in pricing. These mechanisms are complementary: smoothing can be applied either to the dual of an ordinary RMP or to the dual produced by a stabilized master.

In dual smoothing, the dual in the pricing objective is replaced by a convex combination

$$\widetilde{\boldsymbol{\pi}}_t = (1 - \alpha)\boldsymbol{\pi}_t + \alpha \boldsymbol{h}_t, \qquad \alpha \in [0, 1], \tag{3}$$

where $\boldsymbol{h}_t$ is a reference point and α controls the strength of the stabilization.

Different smoothing methods differ primarily in how the reference point $\boldsymbol{h}_t$ is constructed. For example, Neame smoothing (Neame, 2000) uses the previous smoothed pricing vector $\boldsymbol{h}_t = \widetilde{\boldsymbol{\pi}}_{t-1}$, whereas Wentges smoothing (Wentges, 1997) uses an incumbent dual vector associated with the best Lagrangian bound found so far.

[1] In an exact integer-programming method, the same CG procedure can be used to solve LP relaxations within a branch-and-price framework.

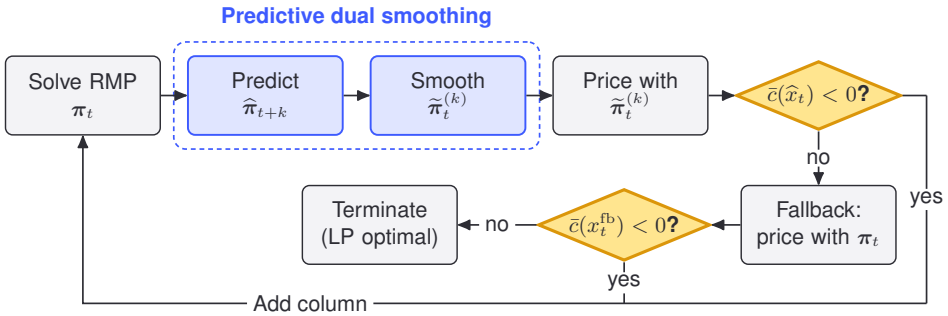


Figure 1: Predictive dual smoothing. The learned predictor and the smoothing step (blue) replace conventional dual smoothing. A column proposed under $\widetilde{\boldsymbol{\pi}}_t^{(k)}$ is accepted only if its reduced cost is negative under the true current dual $\boldsymbol{\pi}_t$. When the test fails, standard pricing with $\boldsymbol{\pi}_t$ either supplies an improving column or certifies LP optimality.

In both cases, the reference point is constructed from duals obtained at earlier stages of CG. As columns are added, however, the needs reflected in these earlier duals may already have been addressed. The resulting pricing signal can therefore remain influenced by needs that are no longer important for the current RMP. While this can reduce the sensitivity of pricing to dual oscillations, it does not necessarily favor columns that will remain useful as the RMP continues to evolve.

4 Predictive Dual Smoothing

Conventional dual smoothing constructs its reference point from information observed in previous CG iterations. Predictive dual smoothing instead uses a prediction of a future dual state, with the goal of guiding pricing towards columns that remain useful as the RMP evolves. Our approach is illustrated in Figure 1. We now discuss its components in turn.

Let $\boldsymbol{\pi}_t$ denote the dual vector given to pricing by the underlying CG procedure. For standard CG, this is the current RMP dual. When a stabilized RMP is used, it is the corresponding dual of this stabilized RMP. Let s_t denote the CG state at iteration t, comprising information from the current RMP and its solution, the pricing formulation, and the progress of the CG procedure. Given s_t, we predict the dual solution k RMP updates ahead and denote the resulting vector by $\widehat{\boldsymbol{\pi}}_{t+k}$. We then define the smoothed pricing vector as

$$\widetilde{\boldsymbol{\pi}}_t^{(k)} = (1 - \alpha_t)\boldsymbol{\pi}_t + \alpha_t \widehat{\boldsymbol{\pi}}_{t+k}, \tag{4}$$

where $\alpha_t \in [0, 1]$ controls the influence of the predicted future dual and is initialized to a hyperparameter α_0. Standard pricing is recovered for $\alpha_t = 0$, whereas $\alpha_t = 1$ prices directly using the predicted future duals. Horizon k controls how far ahead the reference point lies along the CG trajectory: $k = 1$ targets the next RMP duals, while larger k targets more distant future states. We also consider the terminal duals $\boldsymbol{\pi}_T$, to support smoothing with respect to a prediction of the final duals.

4.1 Preserving Correctness

By using smoothed objective $\widetilde{\boldsymbol{\pi}}_t^{(k)}$ in pricing, we lose the guarantee that pricing will produce a column with negative reduced cost whenever one exists. Without an additional safeguard, this would break the correctness and termination guarantees of CG. We therefore first evaluate whether the actual reduced cost of the produced column is negative, using the current duals $\boldsymbol{\pi}_t$.

If this test passes, the column is added to the RMP, and CG continues. If this test fails, we resort to standard pricing with $\boldsymbol{\pi}_t$ as a fallback. If this produces a column with negative reduced cost, this column is added to the RMP. Otherwise, the fallback has certified that no column with negative reduced cost exists, and CG terminates. Predictive smoothing can therefore change *which* improving columns are generated and may occasionally require an additional pricing call, but it cannot add a non-improving column or cause premature termination.

Whenever a fallback occurs, we reduce the smoothing strength as $\alpha_{t+1} = \gamma\alpha_t$, where $\gamma \in [0, 1]$. Otherwise, $\alpha_{t+1} = \alpha_t$. Since predictive pricing becomes more likely to fail as CG approaches convergence, this decay progressively moves pricing towards the current dual and avoids repeated fallbacks. We ablate this mechanism, and evidence its importance, in Appendix E.

4.2 Learning Future Duals

We train the future-dual predictor offline from CG trajectories generated on a set of training instances. For each instance, we run the base CG procedure and record the sequence of states $s_0, \ldots, s_T$ and corresponding dual solutions $\boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_T$, where T denotes the terminal RMP iteration. The base procedure may use either a standard or stabilized RMP.

We formulate future-dual prediction at the level of individual RMP constraints. For each constraint i in state s_t, we construct a fixed-dimensional feature vector $f_i(s_t)$ and predict the corresponding future dual value with a shared model m_θ. For prediction horizon k, the target is $\pi_{i,\tau(t,k)}$, where $\tau(t, k) = \min\{t+k, T\}$. States within k iterations of convergence use the terminal dual as their target rather than being discarded. Let $\mathcal{D}_k$ denote the resulting set of constraint-level training examples. We train m_θ by minimizing

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}_k|} \sum_{(t,i)\in\mathcal{D}_k} \left(m_\theta(f_i(s_t)) - \pi_{i,\tau(t,k)}\right)^2. \tag{5}$$

A single collection of trajectories can be reused across prediction horizons. Changing k only changes which future dual supplies the target label. Further training details are given in Appendix B.

Feature representation. For each dual value π_i, we construct a fixed-dimensional feature vector $f_i(s_t)$ from quantities already available in the current RMP and pricing formulation. The features are designed to capture four aspects of the CG state relevant to predicting how the dual associated with constraint i will evolve: (i) the *global CG state*, such as the iteration, RMP size, and objective value; (ii) the *constraint state*, including the current dual value, right-hand side, and primal activity of constraint i; (iii) the *pricing structure*, summarizing how variables associated with constraint i enter the pricing objective and constraints; and (iv) the *column context*, summarizing how constraint i is represented among the columns already present in the RMP. All features are expressed through generic CG quantities rather than problem-specific representations, allowing the same feature construction to be used across problem classes. Full feature definitions are given in Appendix A.

Predictor. Because the predictor is evaluated at every CG iteration, its inference cost directly offsets any runtime gains obtained from better pricing. We therefore deliberately use a lightweight multilayer perceptron (MLP) m_θ. This MLP is single-output, and predicts one future dual value at a time, i.e., $\widehat{\pi}_{i,t+k} = m_\theta(f_i(s_t))$. These evaluations are independent and can be batched in practice.

This approach is permutation equivariant: reordering the RMP constraints simply reorders the corresponding predictions, without changing their values. The predictor therefore does not depend on the arbitrary ordering of the RMP constraints. Moreover, because its input and output dimensions are fixed for a single constraint, the same network can be applied to instances with different numbers of RMP constraints, without modifying the architecture.

5 Experiments

We evaluate predictive dual smoothing on two structurally different CG problems: the cutting stock problem and the generalized assignment problem (GAP). We investigate its effectiveness relative to standard CG and existing stabilization methods, its sensitivity to the prediction horizon, smoothing strength, and amount of training data, and its ability to generalize across instance sizes and complement strong classical stabilization.

Common setup. All learned models are trained offline on trajectories generated from training instances using standard CG (unstabilized for CSP, and Du Merle stabilized for GAP). We select hyperparameters using validation instances and keep them fixed during test evaluation. Our primary efficiency metric is the *paired runtime ratio*, computed by dividing each method's runtime by

Table 1: CSP results. Reported values are means and standard errors of per-instance metrics.

Method	Paired runtime ratio ↓	Columns added ↓	Runtime (s) ↓
Standard CG	1.000 (0.000)	2394 (97)	33.49 (3.11)
Du Merle stabilization	1.266 (0.019)	2500 (93)	43.21 (4.30)
Neame smoothing	0.865 (0.005)	2119 (92)	30.22 (2.88)
Kraul et al. (2023)	0.987 (0.036)	2207 (93)	29.24 (2.70)
Predictive smoothing (Ours)	**0.630** (0.016)	**1756** (102)	**24.84** (2.70)

standard CG's runtime on the same instance and then averaging these ratios across instances. This normalizes for differences in instance size and difficulty, and gives each test instance equal weight. We additionally report the mean number of non-initial columns generated before convergence and the mean wall-clock runtime, averaged across instances. Results report the mean and standard error over independent test instances. All code will be made available upon acceptance of the paper. Further training, hyperparameter, and implementation details are given in Appendix B.

5.1 Cutting Stock Problem

Setup. We generate CSP instances using the CUTGEN1 generator of Gau & Wäscher (1995). We fix the stock length to $L = 10\,000$ and set the average demand to 10. Following Kraul et al. (2023), for each instance we sample the lower and upper relative item-length bounds independently as $v_1 \sim U[0.05, 0.45]$ and $v_2 \sim U[0.50, 0.85]$. We consider larger instances, sampling the number of item types n uniformly from $\{500, \dots, 1500\}$, compared with $\{50, \dots, 100\}$ in Kraul et al. (2023). We then sample n distinct integer item lengths uniformly without replacement from $[\lceil v_1 L \rceil, \lfloor v_2 L \rfloor]$. If this interval contains fewer than n distinct lengths, we resample v_1 and v_2 until the condition is satisfied. We use 100 instances for training, 50 for validation, and a separate 50 for testing. Hyperparameters are selected by mean paired runtime ratio relative to standard CG on the validation set. For predictive smoothing, this selects $k = 50$, $\alpha_0 = 1$, and $\gamma = 0.9$.

Baselines. We compare against standard unstabilized CG, which prices using the current RMP duals. We then consider two classical stabilization strategies. Du Merle stabilization (Du Merle et al., 1999) represents box-based stabilization with penalties, while Neame smoothing (Neame, 2000) represents dual smoothing by using a convex combination of the current dual and the previous smoothed pricing vector. Finally, we include the learned stabilization method of Kraul et al. (2023), which was developed specifically for the CSP. This method predicts the terminal dual once from static instance features and uses this fixed prediction as the center of a Du Merle stabilized master.

Comparison with existing methods. We report test set results in Table 1. Predictive smoothing achieves the lowest paired runtime ratio, reducing this by 37% relative to standard CG on average. Neame smoothing gives a smaller improvement, with a paired runtime ratio of 0.865, while Kraul et al. (2023) remains close to standard CG at 0.987.[2] Du Merle stabilization is slower than standard CG. Predictive smoothing also generates fewer columns than all baselines, with reductions of 27% relative to standard CG and 20% relative to Kraul et al. (2023).

Effect of the prediction horizon and smoothing strength. Figure 2 shows the validation results across prediction horizons k and initial smoothing weights α_0. Predictive smoothing performs best at an intermediate horizon, with $k = 50$ achieving the lowest paired runtime ratio, while both $k = 1$ and prediction of the terminal dual are less effective. Several factors likely contribute to this trade-off. Very short horizons provide limited look-ahead, so pricing remains sensitive to short-term fluctuations, whereas very long horizons may become too detached from the needs of the current RMP. Prediction accuracy also decreases with k (Appendix D), and larger horizons trigger somewhat more fallback pricing problems, as shown in the right panel of Figure 2.

[2]The (Kraul et al., 2023) result is weaker than reported in their original study, under our setting with less training data and larger instances. We verified that our implementation reproduces their results under their original setting.

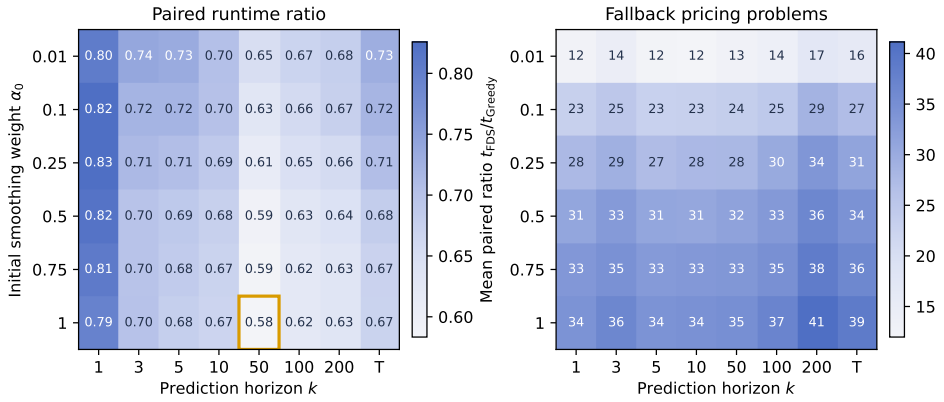


Figure 2: Validation performance of predictive dual smoothing as a function of horizon k and initial smoothing weight α_0. The best configuration is marked in orange.

Table 2: Training data efficiency results for the CSP.

	Predictive Smoothing (Ours)			(Kraul et al., 2023)		
N	Validation R^2 ↑	Paired ratio ↓	Cols. ↓	Validation R^2 ↑	Paired ratio ↓	Cols. ↓
1	-0.858	1.559	3211	0.017	1.222	2507
5	0.836	0.744	1904	0.424	0.920	2193
10	0.857	0.712	1853	0.700	0.920	2176
20	0.869	0.703	1852	0.738	0.930	2150
50	0.873	0.678	1823	0.765	0.935	2114
100	0.873	0.630	1756	0.782	0.987	2207

From Figure 2, we also observe that a small smoothing weight already provides a significant benefit. At $\alpha_0 = 0.01$, predictive smoothing consistently improves over standard pricing. This is because the CSP's pricing problem often has many optimal solutions with identical reduced cost. Because of this, a small perturbation towards the predicted future dual already acts as an informed tie-breaker among these columns.

Finally, we observe that the best paired runtime ratio is obtained at $\alpha_0 = 1$. Although strong initial smoothing increases the number of fallback pricing problems, the corresponding reduction in CG iterations more than compensates for this additional work. In Appendix E, we show that it is the smoothing strength decay mechanism that allows strong initial smoothing $\alpha_0 = 1$ to work well. Also note that $\alpha_0 = 1$ and $\gamma = 0.9$ are also selected in tuning for GAP. We thus find that these are robust default values to use.

Training data efficiency. Table 2 shows that predictive smoothing degrades gradually as the amount of training data is reduced. Performance remains strong with as few as five training instances, while training on a single instance leads to overfitting and poor validation accuracy, resulting in worse CG performance. The comparison with Kraul et al. (2023) also shows that predictive smoothing benefits more from additional training data: both validation accuracy and runtime improve substantially as N increases, whereas Kraul's downstream performance is largely unaffected.

Out-of-distribution generalization. Table 3 shows that when generalizing to smaller instances of size $n = 250$, predictive smoothing remains faster than standard CG but is outperformed by Neame smoothing. A likely reason is that $k = 50$ spans a much larger fraction of the shorter CG trajectories at this scale, making the predictor behave like a terminal predictor. In contrast, the method generalizes well to instances larger than those seen during training. At $n = 2000$ and $n = 2500$, it achieves paired runtime ratios of 0.646 and 0.695, respectively, substantially outperforming all baselines.

Table 3: Paired runtime ratios relative to standard CG on out-of-distribution test sets.

	Paired runtime ratio ↓		
	Smaller than training	Larger than training	
Method	$n = 250$	$n = 2000$	$n = 2500$
Standard CG	1.000 (0.000)	1.000 (0.000)	1.000 (0.000)
Neame smoothing	**0.849** (0.018)	0.843 (0.018)	0.902 (0.014)
Du Merle stabilization	1.253 (0.035)	1.187 (0.036)	1.312 (0.044)
Kraul et al. (2023)	1.187 (0.073)	0.853 (0.054)	0.895 (0.057)
Predictive smoothing (Ours)	0.926 (0.035)	**0.646** (0.036)	**0.695** (0.032)

Table 4: GAP results. Reported values are means and standard errors of per-instance metrics.

Method	Paired runtime ratio ↓	Columns added ↓	Runtime (s) ↓
Standard CG	1.000 (0.000)	18991 (104)	388.93 (5.55)
Du Merle stabilization	0.078 (0.001)	3327 (12)	30.42 (0.43)
Du Merle + Neame smoothing	0.077 (0.001)	3063 (10)	29.74 (0.44)
Du Merle + Pred. smooth. (Ours)	**0.041** (0.001)	**2598** (27)	**15.87** (0.27)

5.2 Generalized Assignment Problem

Setup. The GAP involves assigning jobs to capacitated machines, where assignment costs and resource consumptions are specific to each job-machine combination. When addressed with CG, each machine leads to a separate pricing problem. At every CG iteration, we solve all machine pricing problems and add every optimal pattern with negative reduced cost to the RMP. Predictive smoothing is applied to the shared job-assignment duals. We use Martello-Toth Type C instances with 400 jobs and 20 machines (Romeijn & Romero Morales, 2001). Training, validation and testing use 100, 50 and 50 instances, respectively. The full formulation and implementation details are given in Appendix C. For predictive smoothing, tuning selected $k = 200$, $\alpha_0 = 1$, and $\gamma = 0.9$.

Baselines. On GAP, Du Merle stabilization strongly outperforms standard unstabilized CG. We therefore evaluate predictive smoothing on top of this stabilized master and compare against both Du Merle and Du Merle combined with Neame smoothing.

Comparison with existing methods. Du Merle stabilization reduces the paired runtime ratio to 0.078, while adding Neame smoothing provides essentially no further improvement (0.077). Predictive smoothing instead reduces the ratio to 0.041 and lowers the number of generated columns from 3327 to 2598. This shows how the two methods play complementary roles. Du Merle stabilization dampens dual oscillations, while predictive smoothing additionally steers pricing towards columns that address needs expected to arise in later CG iterations.

6 Conclusions and Future Work

We introduced *predictive dual smoothing*, which stabilizes CG by smoothing the current pricing duals towards a learned prediction of a future dual state. The predictor is trained offline from standard CG trajectories, while reduced-cost checks and fallback pricing preserve correctness. Across cutting stock and generalized assignment, predictive smoothing reduces both generated columns and runtime, including when applied on top of strong master-side stabilization. Future work includes embedding predictive smoothing within *branch-and-price* schemes for integer linear programs, where column generation is repeatedly solved throughout the branch-and-bound tree. Another direction is to adapt the prediction horizon and smoothing strength online, potentially using measures of prediction uncertainty. Finally, evaluating transfer across instance distributions and CG formulations would clarify how broadly the approach generalizes.

AI USE STATEMENT

In this work, we used generative AI tools to polish writing and for research execution, specifically for writing feedback and coding assistance. We reviewed all AI-assisted work. AI-assisted code was verified and tested for correctness by the authors, and all writing suggestions were reviewed and edited by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the details needed to reproduce our results throughout the main paper and appendices. Section 5 describes the experimental setup and evaluation metrics shared across both benchmarks, while Section 5.1 and Appendix C describe the instance generation, problem formulations, and column-generation implementations for the cutting stock and generalized assignment problems, respectively. Appendix A specifies the complete feature representation used by the predictor, and Appendix B provides the predictor architecture, trajectory collection and sampling procedures, training procedure, train/validation/test splits, hyperparameter search spaces and selection procedure, as well as software, solver, and hardware details. All code will be made publicly available upon acceptance.

REFERENCES

Behrouz Babaki, Laurent Charlin, and Sanjay Dominik Jena. *COIL: A deep architecture for column generation*. Bureau de Montreal, Université de Montreal, 2022.

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.

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

Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pp. 1–32. Springer, 2005.

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

Lichang Fang, Haofeng Yuan, Yuli Zhang, and Shiji Song. Accelerating column generation algorithm using machine-learning-based column elimination. In *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1945–1950. IEEE, 2023.

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

T Gau and G Wäscher. Cutgen1: A problem generator for the standard one-dimensional cutting stock problem. *European journal of operational research*, 84(3):572–579, 1995.

Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

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.

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

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

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

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

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

Philip James Neame. *Nonsmooth dual methods in integer programming*. PhD thesis, University of Melbourne, Department of Mathematics and Statistics, 2000.

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

H Edwin Romeijn and Dolores Romero Morales. Generating experimental data for the generalized assignment problem. *Operations Research*, 49(6):866–878, 2001.

Subhash C Sarin, Hanif D Sherali, and Seon Ki Kim. A branch-and-price approach for the stochastic generalized assignment problem. *Naval Research Logistics (NRL)*, 61(2):131–143, 2014.

Martin Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations research*, 45(6):831–841, 1997.

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

Yunzhuang Shen, Yuan Sun, Xiaodong Li, Zhiguang Cao, Andrew Eberhard, and Guangquan Zhang. Adaptive stabilization based on machine learning for column generation. *arXiv preprint arXiv:2405.11198*, 2024.

Yuan Sun, Andreas T Ernst, Xiaodong Li, and Jake Weiner. Learning to generate columns with application to vertex coloring. In *The Eleventh International Conference on Learning Representations*, 2022.

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

Haofeng Yuan, Lichang Fang, and Shiji 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.

A Feature Representation

We construct the feature representation from quantities that are available directly from the current RMP and pricing formulation in state s_t. Consider an RMP in canonical form

$$\min_{\lambda \geq 0} \sum_{p \in \mathcal{P}_t} c_p \lambda_p \qquad \text{s.t.} \qquad \sum_{p \in \mathcal{P}_t} A_{ip} \lambda_p \geq b_i, \quad i = 1, \ldots, n,$$

with optimal dual solution $\boldsymbol{\pi}_t$ in iteration t. For each master constraint i, we construct a fixed-dimensional feature vector $f_i(s_t)$. The representation uses only quantities derived from the CG iteration, the corresponding master row, the pricing formulation, and the columns currently present in the RMP. Concretely, we use the following four groups of features:

1. **Global CG state.** We include the current iteration t, the number of columns $|\mathcal{P}_t|$ in the RMP, the current RMP objective value, and the mean reduced cost of the current columns.
2. **Constraint state.** For constraint i, we include its current dual value $\pi_{i,t}$, right-hand side b_i, and current primal activity $a_{i,t} = \sum_{p \in \mathcal{P}_t} A_{ip} \lambda_p$.
3. **Pricing structure.** We identify the pricing variables whose values determine the coefficient A_{ip} of a newly generated column in row i. We include the minimum, mean, and maximum of their coefficients in the pricing objective and pricing constraints. When multiple variables or pricing subproblems are associated with the same master row, we use the minimum, mean, and maximum. Pricing-constraint coefficients are additionally normalized by their corresponding right-hand sides.
4. **Column context.** The fourth group summarizes how constraint i is represented by the columns already generated. Let $\mathcal{P}_t(i) = \{p \in \mathcal{P}_t : A_{ip} \neq 0\}$ denote the current columns that contain a nonzero coefficient in row i. We include the minimum positive, mean, and maximum values of A_{ip} over the current column set, together with the fraction $|\mathcal{P}_t(i)|/|\mathcal{P}_t|$ of columns containing the row. We also include row-specific statistics of the reduced costs of columns in $\mathcal{P}_t(i)$, including the fraction whose reduced cost is close to zero. Finally, when pricing decomposes into multiple blocks or subproblems, we include summary statistics of the pricing-block quantities associated with the columns in $\mathcal{P}_t(i)$.

Features are standardized using training-set statistics. The same transformations are applied at validation and test time.

B Learning and Experimental Details

B.1 Predictor Architecture

We train a separate predictor for each horizon $k \in \{1, 3, 5, 10, 50, 100, 200, \text{terminal}\}$. Each predictor is a row-wise MLP that maps the fixed-dimensional feature vector $f_i(s_t)$ of one RMP constraint to a scalar future-dual prediction. The same MLP is applied independently to every RMP constriant, so its parameter count is independent of instance size.

For both CSP and GAP, the MLP has one fully-connected hidden layer with 32 ReLU units and a linear output layer. Input features are standardized, but target dual values are not, so predictions are produced directly on the original dual scale.

For CSP, the target at state s_t for horizon k is $\boldsymbol{\pi}_{\min(t+k,T)}$, while the terminal model targets $\boldsymbol{\pi}_T$ at every state. Labels are obtained from standard CG trajectories. GAP uses the same target convention, but predicts only the job-partitioning duals (i.e., the machine convexity duals are left unchanged, as they do not affect pricing). GAP labels are obtained from Du Merle trajectories with $\epsilon = 0.5$.

RMP dual solutions are not always unique. We experimented with always using the minimum-norm duals of the optimal dual face as canonical targets (which requires solving a QP per state at data collection time), but found no improvement in downstream CG performance, so we use the duals returned by the LP solver.

B.2 Training Procedure

Training, validation, and test instances are generated using disjoint random seeds. For both problem classes, we use 100 training instances, 50 validation instances, and 50 test instances.

For CSP, we first collect complete standard CG trajectories. Retaining every state and row would produce an unnecessarily large training set, so we subsample each trajectory to obtain data from diverse instances, while keeping training manageable. Concretely, from each training instance we sample 50 states stratified over the trajectory, choosing one state uniformly at random from each of 50 equal intervals. From each selected state, we then sample 256 item rows uniformly at random without replacement. We do this because we found that covering a larger and more diverse set of instances was more useful than using complete trajectories from fewer instances. Each instance therefore contributes $50 \times 256 = 12\,800$ constraint-level examples, leading to approximately 1.28 million training examples. The validation set is constructed independently using the same procedure, producing approximately $640\,000$ validation examples. Sampled states and rows are shared across horizons, with only the future-dual targets changing with k.

For GAP, we retain all job rows from mature states of the Du Merle trajectories. A state is considered mature once no more artificial Big-M initialization patterns are used anymore in optimal RMP solution. Earlier startup states are excluded from training and validation, and predictive smoothing is likewise disabled during this phase at test time.

Continuous features are standardized using means and standard deviations computed from training rows only, and the same transformation is reused for validation and test inference. Near-constant standard deviations are replaced by one for numerical stability.

All predictive smoothing predictors minimize mean squared error using Adam with learning rate 10^{-2}. CSP uses mini-batches of 8192 examples; GAP uses mini-batches of 4096. Examples are shuffled at each epoch.

Training is capped at 100 epochs and uses validation MSE for early stopping. The used paatience is three validation checks, and patience is reset only when validation MSE improves by at least 0.5% relative to the current best value. The checkpoint with the lowest validation MSE is restored before evaluation and saving.

B.3 Hyperparameter Selection

All hyperparameters are selected using the validation set. Validation instances may vary in size, so configurations are ranked by mean paired runtime ratio, $(1/|\mathcal{V}|)\sum_{i\in\mathcal{V}} T_h(i)/T_{\text{Standard}}(i)$. This weights each instance equally and avoids having larger/harder instances dominate the raw mean runtime.

For CSP, we evaluated predictive smoothing with horizons $k \in \{1, 3, 5, 10, 50, 100, 200, \text{terminal}\}$, initial smoothing weights $\alpha_0 \in \{0.01, 0.1, 0.25, 0.5, 0.75, 1\}$, and alpha decay factor $\gamma \in \{0.1, 0.5, 0.9, 1\}$. For Neame smoothing, we evaluated the same α grid. For Du Merle stabilization, we evaluated $\epsilon \in \{0.03125, 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8\}$. Validation selected $\alpha = 0.75$ for Neame, $\epsilon = 0.125$ for Du Merle, $\epsilon = 2$ for Kraul, and $k = 50$, $\alpha_0 = 1$, $\gamma = 0.9$ for predictive smoothing.

For GAP, we evaluated the same grids. Validation selected $\epsilon = 0.5$ for Du Merle stabilization. When tuning smoothing methods used in combination with Du Merle stabilization, we tune the remaining hyperparameters for smoothing after setting the stabilization $\epsilon = 0.5$. Validation selected $\alpha = 0.25$ for Du Merle with Neame smoothing, and $k = 200$, $\alpha_0 = 1$, and $\gamma = 0.9$ for Du Merle with predictive smoothing.

During mature GAP states, predictive smoothing is applied only to the job duals. If no machine produces an improving pattern under predictive pricing, all machine pricing problems are rerun using the exact current job duals, and α_t is decayed only after this global fallback. Du Merle's ϵ is halved only when neither predictive nor exact pricing produces a new column, and termination requires a final run of exact pricing.

B.4 Implementation Details

RMPs were solved using Gurobi 12.0.3 with one solver thread, while pricing used an exact Numba dynamic-programming implementation. Experiments were run on Ubuntu 24.04.4 LTS, using an AMD EPYC 9334 CPU and 252 GiB of RAM.

C Generalized Assignment Problem

The generalized assignment problem (GAP) involves assigning n jobs to m machines. Assigning job j to machine i incurs cost c_{ij} and consumes a_{ij} units of machine capacity, where machine i has capacity b_i. Each job must be assigned to exactly one machine, and the objective is to minimize total assignment cost without exceeding any machine's capacity.

We use a pattern-based master formulation for GAP. Let $\mathcal{P}_i$ denote the set of feasible job subsets for machine i. A pattern $p \in \mathcal{P}_i$ has incidence vector $a^p \in \{0,1\}^n$, where $a^p_j = 1$ if job j is assigned to machine i, and cost $c_p = \sum_j c_{ij} a^p_j$. Feasibility requires $\sum_j a_{ij} a^p_j \le b_i$.

At iteration t, the RMP contains subsets $\mathcal{P}_{i,t} \subseteq \mathcal{P}_i$ and a nonnegative variable λ_p for each generated pattern:

$$\min_{\lambda \ge 0} \quad \sum_{i=1}^{m} \sum_{p \in \mathcal{P}_{i,t}} c_p \lambda_p \tag{6}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} \sum_{p \in \mathcal{P}_{i,t}} a^p_j \lambda_p = 1, \qquad j = 1, \dots, n, \tag{7}$$

$$\sum_{p \in \mathcal{P}_{i,t}} \lambda_p \le 1, \qquad i = 1, \dots, m. \tag{8}$$

The first constraints are job-partitioning constraints and the second are machine-convexity constraints. Let π_j denote the unrestricted dual of job partitioning constraint j and $\sigma_i \le 0$ the dual of the convexity constraint for machine i.

Pricing decomposes into one 0-1 knapsack problem per machine:

$$\max_{x \in \{0,1\}^n} \quad \sum_{j=1}^{n} (\pi_j - c_{ij}) x_j + \sigma_i \tag{9}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i. \tag{10}$$

At each CG iteration, we solve the pricing problem for each machine, obtaining one optimal pattern per machine. We then add every pattern among these optima whose reduced cost is negative.

To obtain an initially feasible RMP, we use a Big-M artificial-column initialization similar to that used in branch-and-price approaches for GAP (Savelsbergh, 1997; Sarin et al., 2014). We introduce one artificial column for each job. Each artificial column covers only that job, and has objective cost of a large value M, which makes artificial columns unattractive once feasible genuine patterns are available. During CG, we consider this initialization phase complete once no more artificial columns are used in the optimal RMP solution. When collecting training data for GAP, we only collect data after the initialization phase. Similarly, at test time, predictive smoothing and Du Merle stabilization are disabled during this initialization phase.

We use Martello–Toth Type C instances with $a_{ij} \sim \mathrm{Uniform}\{5, \dots, 25\}$ and independently $c_{ij} \sim \mathrm{Uniform}\{10, \dots, 50\}$, with $b_i = 0.8 \sum_j a_{ij}/m$ (Romeijn & Romero Morales, 2001). We use $n = 400$ jobs and $m = 20$ machines.

D Predictive accuracy on CSP

Table 5 reports the predictive accuracy of the future-dual predictor on the CSP validation set for different prediction horizons k. Prediction becomes progressively more difficult as the horizon

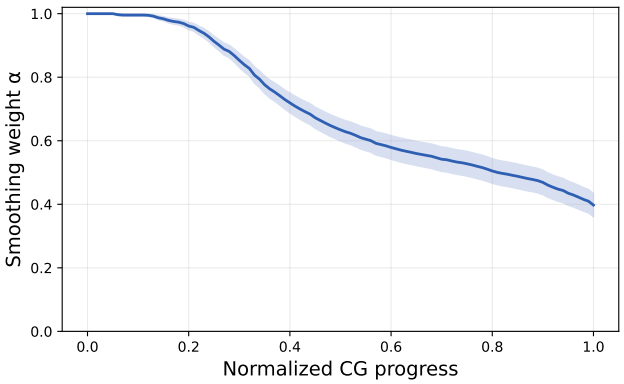


Figure 3: Evolution of the smoothing strength α_t over normalized CG progress for the selected CSP configuration ($k = 50$, $\alpha_0 = 1$, $\gamma = 0.9$). The line shows the mean across validation instances and the shaded region shows SEM across instances.

increases. Validation RMSE generally increases and R^2 decreases as the target dual lies further along the CG trajectory.

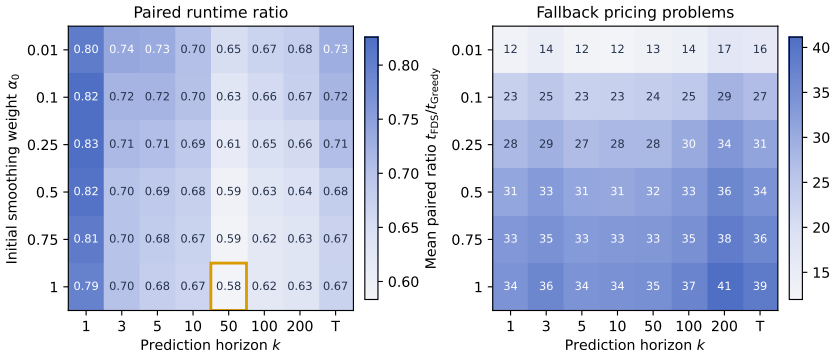
Table 5: Predictive accuracy of the predictor as a function of the horizon k.

Prediction horizon k	1	3	5	10	50	100	200	Terminal
Validation RMSE	0.078	0.091	0.090	0.097	0.112	0.119	0.124	0.131
Validation R^2	0.938	0.916	0.918	0.906	0.873	0.855	0.838	0.802

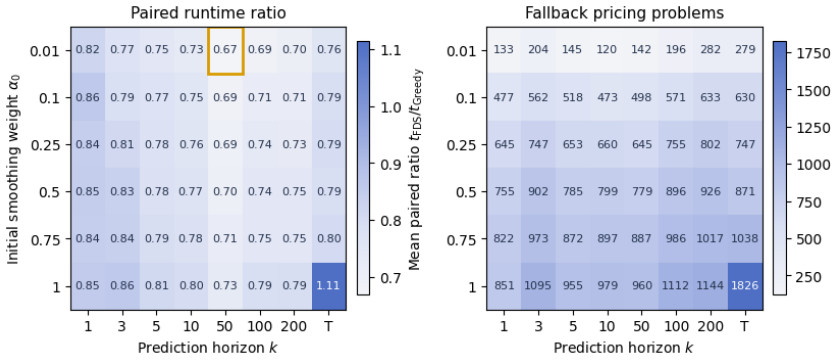
E Ablation of smoothing strength decay γ

Figure 3 shows how the smoothing strength α_t evolves for the selected CSP configuration ($k = 50$, $\alpha_0 = 1$, $\gamma = 0.9$). The method retains strong predictive smoothing early in CG, while fallbacks progressively reduce α_t later in the trajectory, gradually shifting pricing towards the current RMP dual.

This decay is important for efficiency. Figures 4a and 4b compare predictive smoothing with $\gamma = 0.9$ against the non-decaying case $\gamma = 1$. Decaying the smoothing strength after a fallback significantly reduces both runtime and the number of fallback pricing problems. $\gamma = 0.9$ consistently improves runtime across the grid and makes performance considerably less sensitive to large α_0. This supports the adaptive rule used in our main experiments. Predictive smoothing can remain strong while its guidance is useful, and repeated fallbacks provides a simple signal to move pricing progressively closer to the current RMP dual as CG approaches convergence.



(a) $\gamma = 0.9$



(b) $\gamma = 1$ (no decay)

Figure 4: Effect of smoothing decay γ on CSP validation performance. Each panel reports the paired runtime ratio and number of fallback pricing problems across prediction horizons k and initial smoothing weights α_0.