Accelerating Column Generation in Highly Degenerate Integer Programming Problems with Template Pricing

Luke Marshall, Prachi Shah, Santanu S. Dey

Received: date / Accepted: date

Abstract We propose a new pricing strategy for column generation (CG), referred to as *Template pricing*. This method is motivated by the desire to coordinate solutions of different pricing subproblems in order to accelerate the convergence of the CG process and simultaneously obtain good quality integer feasible solutions. Instead of finding a column with the optimal reduced cost, Template pricing tries to maximize the similarity of columns with a given template vector, while restricting the search to columns with suitable reduced cost. We present an exact and heuristic method (based on Lagrangian relaxation) to efficiently solve the Template pricing problem. We conduct extensive computational experiments on benchmark instances of the Generalized Assignment Problem (GAP). Our results demonstrate that Template pricing can significantly accelerate the CG algorithm, especially in the presence of significant degeneracy, where several benchmark GAP instances solved over 1000x faster than Dantzig pricing, and over 100x with adaptive dual-smoothing. Template pricing allows us to achieve CG optimal bounds on all 1735 ISA instances, finding stronger bounds in 43% and improved integer solutions in 9% of these instances than previously released.

Keywords Template Pricing · Dual Degeneracy · Generalized Assignment Problem.

Mathematics Subject Classification (2020) 90C05 · 90C06 · 90C09 · 90C49

1 Introduction

Column generation (CG) is used when the number of variables in a linear program (LP) is so large that it is intractable to be explicitly enumerated. Many discrete optimization problems with compact representations are reformulated using decomposition-based methods such as Dantzig-Wolfe decomposition [21,59], to have such a large set of variables. This is often advantageous, as the reformulated problem may provide stronger dual bounds, use a smaller working model to solve, and parallelize more easily.

While the idea behind CG is simple in theory, its effectiveness in practice is encumbered by challenges, such as efficiently generating columns of good quality, managing existing columns, maintaining the stability of the dual values, reducing the impact of degeneracy and integrating CG with effective branching and cuts [59,44,58]. Moreover, the columns generated for different subproblems in any iteration are often uncoordinated since the pricing problems are solved independently. As a result, only a small fraction of the new columns are likely to enter the basis in the subsequent iteration, leading to slower convergence. Moreover, as Ghoniem and Sherali [33] point out, the optimal restricted master problem (RMP) might not even contain columns that can be used to construct a feasible solution of the underlying integer program, let alone an optimal integer solution. We call a set of columns *compatible* if they satisfy the linking constraints in the master problem and *incompatible* otherwise. This is a generalization of the notion of complementarity introduced in [33] for set-partitioning problems.

In this paper, we focus on coordinating the solutions of pricing subproblems to generate nearly compatible columns. We propose a novel approach to generate high-quality columns using a technique that we call *Template pricing*. Template pricing modifies the standard pricing problem that generates new columns. Instead of finding columns with optimal *reduced cost*, Template pricing tries to maximize the similarity of columns with a given *template* vector, while restricting the search to columns with suitable reduced cost. Therefore, pricing across subproblems is coordinated by choosing templates that constitute a set of compatible columns. Further, due to the altered objective, the direction of the search in the pricing problem only considers the dual values for tie-breaking, thereby reducing the impact of dual instability and primal degeneracy. Moreover, our approach can be used in conjunction with other "dual-value stabilization" techniques, and a combined approach may yield even greater impact.

The main contributions of this paper are the following:

– We propose a new paradigm for pricing, referred to as Template pricing, a novel approach for coordinating solutions of pricing subproblems to accelerate the convergence of the CG process, while simultaneously obtaining good quality integer feasible solutions.

Luke Marshall
Microsoft Research, Redmond, USA E-mail: luke.marshall@microsoft.com

Prachi Shah
School of Industrial and Systems Engineering, Georgia Institute of technology, Atlanta, USA E-mail: prachi.shah6795@gmail.com

Santanu S. Dey
School of Industrial and Systems Engineering, Georgia Institute of technology, Atlanta, USA E-mail: sdey30@gatech.edu

– We present Template pricing in the context of set-partitioning constraints, with a primary focus on the Generalized Assignment Problem (GAP), a special subclass of such problems; however, this approach can be extended to any problem solved using CG.
– We present an exact approach and a Lagrangian relaxation based heuristic to solve the Template pricing problem.
– We study the impact of different initialization methods and column management on the effectiveness of Template pricing, and their relationship with degeneracy.
– We conduct extensive computational experiments on benchmark instances [6,65,31] of GAP [49,15,56,17, 66]. We compare our approach with Dantzig pricing and Pessoa pricing [64]. Our results demonstrate that Template pricing can significantly accelerate the CG algorithm, with some instances solving over 1000x faster.

The rest of the paper is organized as follows. We provide background on CG in Section 2. Section 3 introduces Template pricing and compares our approach with others in the literature, which combat similar challenges. We also present a Lagrangian relaxation based approach to solve the Template pricing problem. In Section 4 we analyze important design choices when implementing CG, such as initialization and column management of the RMP. We make informed parameter choices for all methods we evaluate, via extensive experimentation. We present results of our computational study in Section 5, where we compare Template pricing against existing approaches, and provide evidence and insights into its superior performance. Finally, we conclude and discuss future research in Section 6.

2 Column Generation (CG) Preliminaries

In this section, we introduce CG for problems where the linking constraints are of the set-partitioning type. This constitutes a large class of problems solved using CG and covers GAP [56,54], vehicle routing [23,30,4,50], graph coloring [47,38], bin packing [14], and a wide range of scheduling problems such as crew scheduling [36, 22,5] and the gate assignment problem [42]. The compact formulation is represented as,

$$\min_{\mathbf{x}} \quad \sum_{i\in I} \mathbf{c}_i^\top \mathbf{x}_i \tag{1a}$$

$$\text{s.t.} \quad \sum_{i\in I} \mathbf{e}_j^\top \mathbf{x}_i = \mathbf{1} \qquad \forall j \in J, \tag{1b}$$

$$\mathbf{A}_i \mathbf{x}_i \leq \mathbf{b}_i \qquad \forall i \in I, \tag{1c}$$

$$\mathbf{x}_i \in \{0,1\}^{|J|} \qquad \forall i \in I, \tag{1d}$$

where e_j is the unit vector in the j^{th} direction. Since GAP is a prototypical set-partitioning problem, we discuss (1) in this context. Here, the variable $\mathbf{x}_i := (x_{i1},\ x_{i2},\ \ldots,\ x_{i|J|})^\top$ is a vector with binary elements x_{ij} that represents if job $j \in J$ is assigned to machine $i \in I$. The set-partitioning constraint (1b) ensures that all jobs are assigned exactly once. The complicating constraints (1c) are a single knapsack constraint for each machine that enforces that the assignment of job demands do not exceed machine capacity. Henceforth in the paper, it will often be convenient to refer to the i index as machine and j index as job even for the general set-partitioning instances.

This formulation can be decomposed using Dantzig-Wolfe reformulation, and by taking its linear relaxation gives the following formulation:

$$\min_{\lambda} \quad \sum_{i\in I} \sum_{\mathbf{v}_{ip}\in P_i} \left(\mathbf{c}_i^\top \mathbf{v}_{ip}\right) \lambda_{ip} \tag{2a}$$

$$\text{s.t.} \quad \sum_{i\in I} \sum_{\mathbf{v}_{ip}\in P_i} \mathbb{1}_{\{\mathbf{e}_j^\top \mathbf{v}_{ip}=1\}} \lambda_{ip} = 1 \qquad \forall j \in J \qquad (\pi_j) \tag{2b}$$

$$\sum_{\mathbf{v}_{ip}\in P_i} \lambda_{ip} = 1 \qquad \forall i \in I \qquad (\mu_i) \tag{2c}$$

$$\lambda_{ip} \geq 0 \qquad \forall i \in I,\ \mathbf{v}_{ip} \in P_i. \tag{2d}$$

where $P_i = \left\{\mathbf{x} \in \{0,1\}^{|J|} \mid \mathbf{A}_i\mathbf{x} \leq \mathbf{b}_i\right\}$, and λ_{ip} is the variable corresponding to the column $\mathbf{v}_{ip} \in P_i$. The notation $\mathbb{1}_{\{\mathbf{e}_j^\top \mathbf{v}_{ip}=1\}}$ represents the value 1 if the j^{th} job is assigned to the i^{th} machine in column $\mathbf{v}_{ip}$ and 0 otherwise. The dual variables for (2b) and (2c) are π_i and μ_j, respectively. Enumerating all vectors in P_i is often intractable, so to solve (2), a restriction on the subset of vectors $V_i \subseteq P_i$ is instead solved iteratively. The model defined on V_i is called the Restricted Master Problem (RMP). At every iteration, the pricing problems are solved for each subproblem i to identify the columns to add to V_i, or to prove optimality (and terminate)

if none exist. For the column $\mathbf{x} \in P_i$, to be added to V_i, it must have a negative reduced cost, also referred to as a good reduced cost. Therefore, it must satisfy the following, for some small tolerance $\varepsilon > 0$,

$$\sum_{j \in J} (c_{ij} - \pi_j)\, x_j \leq \mu_i - \varepsilon. \tag{3}$$

At each iteration, at least one column having good reduced cost must be added to the RMP across all machines to guarantee convergence. If no such columns exist for all i, this certifies that the solution of RMP is optimal for the original master problem (2). Formally, at each iteration, having corresponding dual values (π, μ), the pricing subproblem chooses columns from the set $S^i_{(\pi,\mu)} = \{\mathbf{x} \in P_i \mid (3)\}$ or shows that $S^i_{(\pi,\mu)} = \emptyset$ for all $i \in I$.

2.1 Relation with Lagrangian Relaxation

Though not the focus of this paper, Lagrangian Relaxation (LR) is closely related to CG. We include a discussion of LR to provide context for the pricing strategies explored in Section 2.2.2 (dual smoothing), and also because prior studies report strong practical performance on GAP under degeneracy [56].

Both LR and CG are decomposition techniques driven by dual information [32,28]. LR relaxes hard constraints by incorporating them into the objective function with associated penalties (Lagrange multipliers). We outline the LR derivation in the remainder of this section. For GAP, relaxing the set partitioning constraints (1b) in LR yields the same bounds as the CG RMP. By relaxing these linking constraints, LR decomposes GAP into independent binary knapsack problems – again, similar to the CG decomposition. Relaxing the set partitioning constraints transforms the problem into:

$$\begin{aligned} &\max_{\pi} \left\{ \min_{\mathbf{x}} \left\{ \sum_{i \in I} \mathbf{c}_i^\top \mathbf{x}_i + \sum_{j \in J} \pi_j \left(1 - \sum_{i \in I} \mathbf{e}_j^\top \mathbf{x}_i \right) \,\middle|\, (1c), (1d) \right\} \right\} \\ &= \max_{\pi} \left\{ \sum_{j \in J} \pi_j + \sum_{i \in I} \min_{\mathbf{x}} \left\{ \sum_{j \in J} (c_{ij} - \pi_j)\, x_{ij} \,\middle|\, (1c), (1d) \right\} \right\}. \end{aligned}$$

Moreover, these Lagrange multipliers correspond directly to the dual values of the CG RMP. LR explicitly updates the multipliers iteratively (via subgradient, bundle, level methods etc.) depending on the chosen implementation. LR uses the gradient $\nabla\pi$ to update the multipliers at each step:

$$\nabla\pi_j = 1 - \sum_{i \in I} x_{ij} = 1 - \sum_{j \in J} \mathbb{1}_{\{\mathbf{e}_j^\top \mathbf{v}_{ip} = 1\}}.$$

In degenerate GAP instances, LR can often outperform CG since it avoids solving an LP at each iteration. Nevertheless, LR has its own drawbacks: no primal feasibility guarantees (often needs additional heuristics to obtain primal solutions), sensitive to step sizes, potential oscillations and slow convergence, limited coordination across subproblems, and often terminates at a near-optimal solution (without clean certification).

Note that, similar to (2), subproblems in LR yield solutions of the form of v_{ip}. Steps are performed iteratively until some termination criteria; for example, when the change in objective value between two iterations is less than a given threshold.

2.2 Column Generation Pricing Strategies

The most natural approach for choosing columns from $S^i_{(\pi,\mu)}$ is *Dantzig* pricing. It uses (3) as the objective function and checks whether the resulting minimal value has good reduced cost, specifically:

$$rc(i) := \min_{x \in P_i} \left\{ \sum_{j \in J} (c_{ij} - \pi_j)\, x_j \right\} - \mu_i \leq -\varepsilon. \tag{4}$$

It is not necessary to find columns with optimal reduced cost, in fact, it is often unlikely that columns with minimum reduced cost are the best choice to reduce time and iterations for the RMP to converge. Other objective functions, also known as *pricing strategies*, for choosing solutions from $S^i_{(\pi,\mu)}$ have been proposed in the literature in the context of column generation [44] and more generally for simplex method, such as deepest-cut in the dual space [60], and steepest-edge [29,34] and its variants as Devex [37] and Lambda [9] pricing. Although CG can be considered an extension of primal simplex, many pricing strategies applicable for primal simplex are not practical for CG due to non-linearity, since CG pricing involves solving an optimization problem, it is computationally preferable to have a linear objective function.

More generally, an important unresolved question is whether there exists a pivoting rule under which the simplex algorithm operates in polynomial time. See for example, the smoothened analysis [57,20] and the paper [11] and references therein for a comprehensive discussion of proposed rules and several lines of research.

2.2.1 Disadvantages with Dantzig Pricing

Dantzig pricing can be an excellent choice for generating columns. It simplifies $S^i_{(\pi,\mu)}$ by moving (3) into the objective; it yields valid bounds on the RMP at each iteration; and is typically the way to guarantee termination at optimality. However, Dantzig pricing suffers on instances that have degeneracy.

Degeneracy is known to be associated with unstable dual values in the RMP when solving with the simplex algorithm. This is unfortunate, since primal simplex is a good choice for solving the RMP in CG, due to the computational efficiency of warm starts when adding columns. Unstable dual values provide bad search directions for pricing and fluctuate greatly between CG iterations. This issue is well known in the literature and there exist many papers on dual stabilization techniques to improve convergence in CG; see for example [64,25, 27,7,2,55,13,3,33,26,8,52,35,53,19]. These approaches encompass a variety of strategies, including trust-region methods, interior-point stabilization, constraint aggregation and coordination schemes, as well as dual-optimal inequalities.

Dual value instability also yields issues with uncoordinated subproblems. There are computational advantages to solving subproblems independently, such as parallelization. However, the columns generated for different subproblems may be incompatible, that is, may violate the RMP constraints and therefore impossible to simultaneously add to the RMP. Dantzig pricing implicitly uses dual values for coordination, but it is fragile, especially with unstable dual values, since it relies on potentially small differences in objective coefficients. In particular, observe that for job j, the objective coefficient in (4) relies on the dual value π_j, which is shared across every subproblem for each machine and perturbed by the c_{ij} costs. Jobs corresponding to large dual values that outweigh c_{ij} perturbations are likely to be selected by several machines, whereas jobs corresponding to small dual values might not be selected at all. The generated columns often compete for the same jobs and are unlikely to enter the RMP basis simultaneously in the subsequent iteration. This wasted computational effort leads to less improvement in the RMP solution across iterations and therefore has slower convergence.

To avoid the issue of incompatible columns, Ghoniem and Sherali [33] sequentially solve the pricing problem, while cumulatively adding constraints that enforce compatibility within the sequence. This ensures the compatibility of the columns and yields an integer feasible solution; however, the quality of solution is dependent on the choice of sequence ordering and is more computationally expensive, since it prohibits parallel computation. Alternatively, there are dynamic constraint aggregation (DCA) methods, see for example [27,26,8,12], that group jobs together and modifies the pricing such that it first tries to find columns satisfying this grouping while having good reduced cost. In this way, it implicitly yields nearly compatible columns at each iteration. If no suitable columns are found, the aggregation is refined. This approach provides dual stabilization, parallelization, and uses a smaller RMP; at the cost of increased complexity. Specifically, the refinement step may require solving another CG problem, which may have its own issues with degeneracy.

2.2.2 Pricing with Dual Smoothing

Among the numerous approaches to dealing with dual stabilization discussed above, our paper focuses specifically on the impact of pricing strategies. We exclude approaches that either modify the RMP or use alternative algorithms to solve the RMP, so that we can isolate the contribution of pricing.

In this section, we describe a strong dual smoothing technique that is used as a baseline in our study. At a high level, dual smoothing constructs a smoothed dual vector $\tilde{\pi}_t$ for each iteration t, using information gathered across iterations. It then uses $\tilde{\pi}_t$ rather than the raw RMP duals π_t to define the pricing objectives. Two classical forms are:

- *Neame* [48]: a static convex combination of the previous smoothed duals and the current duals,

$$\tilde{\pi}_t = \alpha_0 \tilde{\pi}_{t-1} + (1 - \alpha_0)\, \pi_t$$

- *Wentges* [64]: a static convex combination of the most recent improving dual vector $\hat{\pi}_{t-1}$ (from the most recent iteration that improved the RMP objective) and the current duals,

$$\tilde{\pi}_t = \alpha_0 \hat{\pi}_{t-1} + (1 - \alpha_0)\, \pi_t$$

with the static mixing parameter $\alpha_0 \in [0, 1]$. In our baseline implementation we use the advanced smoothing techniques from *Pessoa* [51].

Pessoa generalizes Wentges by adding (i) an adaptive mixing parameter and (ii) a directional component derived from subgradient information; coupled with a limited backtracking procedure designed to ensure that pricing still produces at least one column with negative reduced cost (when possible). This backtracking progressively reduces the impact of smoothing and ultimately reverts Pessoa to standard Dantzig pricing.

In our GAP setting, Pessoa's directional component uses the subgradient information from Lagrangian Relaxation, defined in Section 2.1. The elements in the subgradient correspond to each job, and reflects whether the solutions found by independently solving each subproblem (per machine) over-cover or under-cover that job

in the current iteration. Pessoa uses this information to penalize over-covered jobs and prioritize under-covered jobs, providing directional correction beyond simple convex smoothing.

While highly effective, the mathematics behind Pessoa is substantially more complex than Wentges, Neame, and even Template pricing. For completeness, we provide the full details of our implemented Pessoa variant in Appendix A.

3 Template Pricing

Template pricing aims to find a set of new columns (one for each subproblem) that maximizes similarity to a corresponding set of template vectors, while also having good reduced cost. These template vectors coordinate the subproblems, even though each subproblem is solved independently. Templates can be chosen from existing columns or generated by some heuristic and serve as a reference point to guide the search.

Formally, suppose we are given $\{y^i\}_{i\in I}$, a near-optimal solution to the relaxed RMP that satisfies the linking constraints. We wish to guide the search to converge towards this solution. We refer to these $y^i \in \mathbb{R}^{|J|}$ vectors as *templates* and define Template pricing as the strategy that finds columns from $S^i_{(\pi,\mu)}$ with maximum similarity to the corresponding template; specifically:

$$\min_{x\in S^i_{(\pi,\mu)}} \left(-d\left(x, y^i\right), \quad \sum_{j\in J} \left(c_{ij} - \pi_j\right) x_j \right), \tag{5}$$

a bi-objective optimization, which can be interpreted as:

$$\min \sum_{j\in J} \left(c_{ij} - \pi_j\right) x_j \tag{6}$$

$$\text{s.t. } x \in \arg\max\nolimits_{x\in S^i_{(\pi,\mu)}} \left\{d(x, y^i)\right\}. \tag{7}$$

Template pricing (5) defines a hierarchical optimization problem where the primary objective is to maximize similarity to the template, with ties broken using the reduced costs. The domain $x \in S^i_{(\pi,\mu)}$ ensures that the optimization produces columns with negative reduced cost. Equivalently, we can view this as the inner problem (7) selecting a column that maximizes similarity to the template, and then the outer problem (6) acting as a tie-breaker, choosing among the optimal solutions of (7) the columns with minimum reduced cost.

3.1 Designing Templates

We propose designing templates in a *static* or *dynamic* fashion, depending on whether they change during the CG process. For example, a static template could be constructed from a heuristic solution to (1), or the solution x^* of its LP relaxation, that is, $y^i = \{x^*_{ij}\}_{j\in J}$ for $i \in I$. Other ideas include using convex combinations of near-optimal linear relaxation solutions, or Lagrangian lower bound solutions.

We propose dynamic templates based on the incumbent RMP solution, updated periodically. Let λ^* be the solution of the RMP at some iteration. The template for each machine i is defined as:

$$y^i = \sum_{v_{ip}\in P_i} \lambda^*_{v_{ip}} v_{ip}, \tag{8}$$

that is, the template is the projected primal solution of the RMP.

Initial experiments showed that static templates are beneficial, particularly when chosen well; however, the dynamic template (8) consistently produced improvements. Therefore, we present only results using (8) in the paper.

3.2 Designing Similarity Function

The function d measures the similarity of x to the template y^i, where larger values indicate greater similarity. In this paper, we define d as follows:

$$d(x,y) = \sum_{j\in J} f\left(y_j\right) x_j, \text{ where } f(y_j) = \begin{cases} 1 & \text{if } y_j \in (1-\delta,\, 1] \\ 0 & \text{if } y_j \in [\delta,\, 1-\delta] \\ -1 & \text{if } y_j \in [0,\, \delta). \end{cases}, \tag{9}$$

for some $\delta \in [0, 0.5]$; we choose $\delta = 10^{-6}$. The intuition behind (9) is as follows: we wish to set $x_j = 1$ when $y_j \approx 1$ and $x_j = 0$ when $y_j \approx 0$. When y_j is fractional (neither close to 0 or 1) we wish to choose x_j by tie

breaking on the reduced cost. Specifically, we choose $x_j \in \{0,1\}$ such that $f(y_j)x_j$ is maximized, so $f(y_j \approx 1) = 1$ encourages $x_j = 1$ and $f(y_j \approx 0) = -1$ encourages $x_j = 0$. When y_j is fractional, $f(y_j) = 0$ does not encourage the choice of x_j in either direction. In other words, maximizing $d(x, y)$ favors "soft fixing" variables to their template values, whenever they are near integral.

Templates that are highly fractional with a small choice of δ, lead to $d(x, y^i) \approx 0$, that is, Template pricing reverts to Dantzig pricing. It could be quite reasonable to combine Template pricing with dual smoothing, like Pessoa, by adjusting the tie-breaking objective; for simplicity, however, we do not explore this in our paper.

3.3 Advantages of Template Pricing

We list below several advantages of Template pricing.

1. *Coordinated independent subproblems:* As discussed above, solving pricing subproblems independently offers computational advantages such as parallelism, but it is likely to produce incompatible columns. Template pricing coordinates these independent subproblems via compatible templates. By attempting to find columns similar to the templates, they are more likely to be compatible. Thus, Template pricing supports parallelization while reducing column incompatibility.
2. *Column stabilization:* Other pricing algorithms attempt to stabilize dual values to reduce fluctuations between iterations and improve convergence. Template pricing attempts to stabilize the generated columns themselves, such that the columns from one iteration to the next are similar. Section 5 demonstrates that Template pricing reduces the number of simplex pivots per iteration in the RMP. Each pivot is also faster, which significantly accelerates the overall convergence speed.
3. *Integrality:* Using (8) for our template and (9) for similarity in our Template pricing ensures compatibility and reinforces column stabilization. This strategy also encourages the RMP towards integer feasible solutions throughout the CG process. In addition to feasible solutions, the resulting integral solution yields a smaller basis, which further reduces pivot times.
4. *Degeneracy:* Prevalent in partitioning based CG problems, particularly when the job/machine ratio is high [56], degeneracy leads to multiple dual optimal solutions, making the direction indicated by a single dual solution less reliable. This leads to degenerate pivots and wasted computational effort. Template pricing reduces the impact of degeneracy by instead focusing on template similarity; specifically, these dual values have less control in the generation of columns (provided the reduced costs are negative). In this regard, Template pricing shares similarities with DCA methods [27,26,12] that iteratively fix a subset of jobs to some machines to create a reduced non-degenerate problem. Likewise, Template pricing prefers anchoring the jobs in the template on a machine, but does not enforce this strictly.
5. *Simplicity:* Template pricing is easy to implement (see Appendix B), since state-of-the-art solvers like Gurobi and HiGHS have good implementations for solving hierarchical optimization problems exactly. As an alternative, the next section presents a heuristic implementation of Template pricing based on Lagrangian relaxation. In comparison, other rules for pricing such as lambda pricing [10] and DCA [27,26,12] rules are significantly more challenging to implement.

3.4 Lagrangian-relaxation based approximation for Template pricing

As discussed in the previous section, most state-of-the-art solvers provide built-in support for hierarchical objectives. However, one drawback of hierarchical optimization is that during the tie-breaking phase one must introduce an additional constraint to the feasible region of the pricing problem (see Appendix B). This modification can reduce the efficiency of specialized solution methods for the pricing problem, such as dynamic programming (DP) algorithms for knapsack problems. To avoid this issue, instead of solving the hierarchical optimization problem exactly, we investigate an approximate approach based on Lagrangian relaxation combined with bisection search. The main advantage of this method is that it preserves the original feasible region of the pricing problem. In particular, our approach uses a DP algorithm for solving the binary knapsack problem, adapted from the implementation in SCIP [39].

We begin by verifying that $S^i_{(\pi,\mu)} \neq \emptyset$, that is, solve the Dantzig pricing problem and check for reduced cost. If (4) does not hold, we terminate since $S^i_{(\pi,\mu)} = \emptyset$. This step is not strictly necessary, but it is extremely fast in our setting, and the optimal reduced cost can be used to improve bounds and terminate early. Assume that $S^i_{(\pi,\mu)} \neq \emptyset$, we proceed as follows. At each iteration of our algorithm we solve the following Lagrangian relaxation problem corresponding to the "Lagrangian parameter" $\alpha \geq 0$:

$$\mathrm{OPT}(\alpha) := \min_{x \in P_i} \left\{ -d(x, y^i) + \alpha \sum_{j \in J} (c_{ij} - \pi_j)\, x_j \right\}. \tag{10}$$

Let x^α denote an optimal solution of $\mathrm{OPT}(\alpha)$. Our goal is to determine the smallest value of α for which $x^\alpha \in S^i_{(\pi,\mu)}$. Intuitively, this corresponds to maximizing the similarity of the generated column to the template, while ensuring that it maintains a negative reduced cost.

To initialize, we set $l := 0$, $u := +\infty$, and $\alpha := \alpha_{\text{init}} > 0$. At each iteration we solve (10) and then update the interval $[l, u]$ as follows. If $x^\alpha \in S^i_{(\pi,\mu)}$, set $u := \alpha$; otherwise set $l := \alpha$. In the former case, α is already sufficiently large to yield a solution in $S^i_{(\pi,\mu)}$, whereas in the latter case it is too small.

The value of α for the next iteration is selected via bisection search as

$$\alpha := \begin{cases} \dfrac{l+u}{2}, & \text{if } u < +\infty, \\ 2\alpha, & \text{if } u = +\infty. \end{cases}$$

The procedure terminates when $\frac{u-l}{l} \leq 10^{-3}$, or if it can prove optimality for a solution with negative reduced cost, that is, using $\lfloor \alpha\mu_i - \mathrm{OPT}(\alpha) \rfloor$ as a valid upper bound for maximum template similarity. We return x^α corresponding to $\alpha = u$. In our implementation, we set $\alpha_{\text{init}} = 0.5$ when solving the pricing problem for a given machine for the first time. For subsequent pricing problems associated with the same machine, α_{init} is set equal to the optimal value of α obtained in the preceding CG iteration.

We call the above procedure the Lagrange Template (LT) pricing method. Figure 1 shows the optimal α obtained via LT pricing at each CG iteration, together with $\Delta\alpha$ (the change in the optimal α between successive CG iterations), for a single machine on one instance from our test library. Observe that optimal α increases significantly as CG process approaches termination. Larger values of α place greater weight on the reduced-cost term when solving the pricing problem, indicating that near termination it becomes increasingly difficult to identify columns with sufficient negative reduced cost; consequently, similarity to the template becomes less critical. The $\Delta\alpha$ remains consistently small (except near CG termination), suggesting that warm-starting LT with the previous value of α is beneficial. Figure 2 reports the average optimal α across all machines, along with the minimum and maximum values of α over machines at each CG iteration. Note that the vertical axis is on a logarithmic scale, and the growth of optimal α appears to follow an exponential trend.

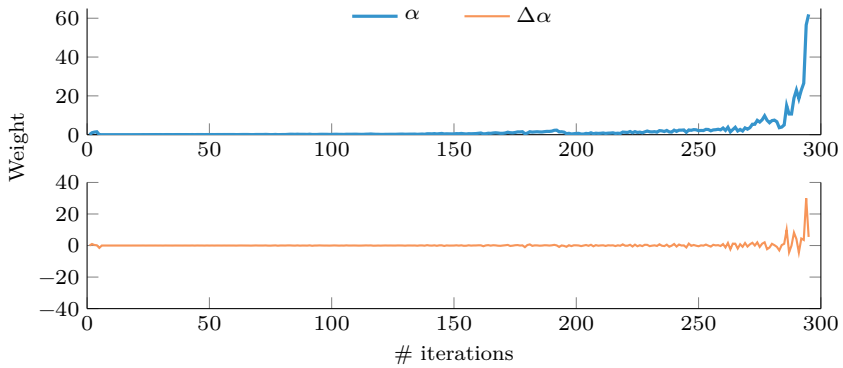

Fig. 1: The optimal α weight and delta change per iteration for a single machine in LT pricing for the instance c10400.

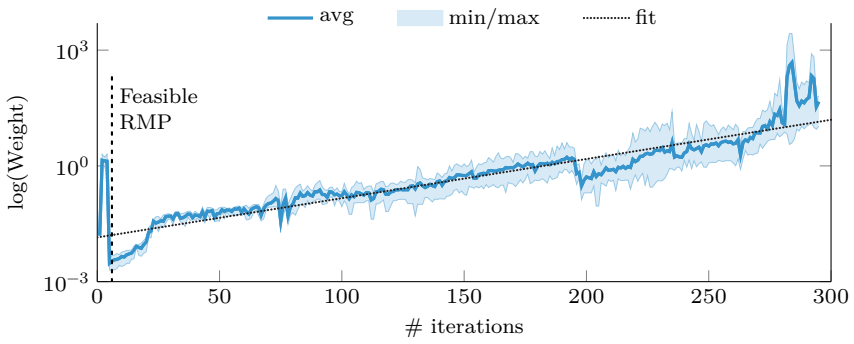

Fig. 2: Average of the optimal α weights across all machines per iteration in LT pricing for instance c10400. Min/Max values of α across machine is shown by shaded region. The fit plotted above corresponds to: $0.014e^{0.0234x}$

4 Key Algorithmic Considerations

To make the experimental analysis as fair as possible, this section covers some important considerations for parameter selection and experimental design. In particular, we discuss the benchmark instances and show some initial results that guide our choices.

4.1 Instances

GAP has been extensively studied with classic sets of instances dating back at least 1994 [16]. Those 60 instances are now trivial and can be solved within 1 second; however, some instances generated in 1997 [18] and extended in 2006 [66] are still computationally challenging. At the time of writing, we have not seen evidence that all instances have been solved to optimality; however, they have all been solved within less than 0.1% optimality gap. We focus the core of our experiments on these 57 instances [18,66]. More recently, however, in 2023 [31], the classic set of instances have been extended even further by performing an instance space analysis, which attempts to systematically design a set of instances that cover a range of various metrics. We also evaluate these additional 1735 instances and provide improved bounds. The aim for [31] was to generate a diverse set of instances and evaluate with fast heuristics. Their computational study evaluated the solutions with a time limit of 15 minutes, whereas we focus on the convergence of CG at the root node. We refer to the 57 instances as *Yagiura* [66] and the 1735 instances as *ISA* [31].

4.2 Degeneracy

Savelsbergh [56] identified that CG with Dantzig pricing has slow convergence when the ratio of jobs to machines is greater than five, and noted that LR is typically more efficient for these harder instances. The job to machine ratio is an excellent proxy for measuring degeneracy in GAP, that is, large ratios are correlated to high degeneracy and unstable dual values, which is particularly challenging for Dantzig pricing. Lübbecke and Desrosiers [45] suggest that instances having a ratio over ten are considered to have massive degeneracy. The instances we consider in our experiments have a minimum ratio of 5, average of 28 ± 18, and maximum of 80. They are excellent examples to highlight the impact of degeneracy. Our results in Section 5 show that CG with Template pricing can be competitive against LR, even with high degeneracy, indicating there are additional factors to consider when choosing between CG and LR.

4.3 Algorithms

Our study aims to evaluate the impact of various pricing strategies. We refrain from exploring algorithms that explicitly and iteratively modify the RMP or solving the RMP with interior point methods to implicitly provide dual stabilization. However, we consider one minor change to the RMP that is commonly applied in CG for dual stabilization; we replace set partitioning with cover constraints. With this in mind, our evaluation considers the following pricing strategies:

(D) Dantzig – no stabilization,
(P) Pessoa – directional and adaptive smoothing,
(LT) Lagrange Template – heuristic template, and
(MT) MIP Template – exact template.

As a final exception, we also include a limited comparison with Lagrangian Relaxation (LR) – due to Savelsbergh's guidance [56] on degenerate instances. Specifically, on Yagiura we compare $\{D, P, LT, MT\}$, and ISA $\{P, LT, LR\}$. We exclude D and MT from ISA experiments to focus on the most tractable algorithms.

4.3.1 Details of Lagrangian Relaxation implementation

Recall the derivation of LR for GAP in Section 2.1. Reasonable effort has been made to ensure that our LR baseline is a fair comparison. We evaluated several approaches and selected the most efficient implementation; however, considering the vast literature on Lagrangian Relaxation, faster methods may exist.

At each iteration, we calculate the current lower bound objective and gradient by solving all knapsack subproblems. For fair comparison, we use the same binary knapsack solver as our CG method. We terminate when the absolute change in objective is less than 10^{-6}. The gradient is used to update the multipliers via a directional step; specifically, we use L-BFGS [43] with a memory of 256. L-BFGS uses previously generated gradients to more accurately approximate the Hessian matrix for faster convergence.

1. Initialize multipliers $\pi = \mathbf{0}$
2. Solve knapsack subproblems for given π
3. Compute lower bound objective and gradient from subproblem solution
4. Update π using L-BFGS backtracking line search step and gradient history
5. Repeat until objective converges (or time out).

4.4 Termination and Integrality

Our computational study terminates each instance at the CG root node; we do not solve the integer problem via branch-and-price. Introducing the complexity of branching would distract from our comparison on pricing strategies. Nevertheless, we can exploit the integrality of GAP to terminate early at the root node. Specifically, all Yagiura and ISA instances have integer cost coefficients, so the optimal objective must be integer and any valid lower bound can be tightened by rounding up. Formally, let the sum of negative reduced-costs be given by

$$\mathrm{rc}_t = \sum_{i \in I} \min\left\{rc\left(i\right), 0\right\}$$

where $rc(i)$ is the reduced cost for the i^{th} machine as defined in (4), and let RMP_t be the objective value of the RMP in the t^{th} iteration. Since rc_t is an optimistic estimate of the remaining improvement in RMP_t and we have convexity constraints (2c), we have that $\mathrm{RMP}_t + \mathrm{rc}_t$ is a valid lower bound [44]. In our implementation, termination based on lower bounds occurs either when $|\mathrm{rc}_t| < 10^{-6}$, or when $\lceil \max_t\{\mathrm{RMP}_t + \mathrm{rc}_t\}\rceil \geq \mathrm{RMP}_t$ by exploiting integrality.

Note that primal heuristics are not explicitly implemented, so a feasible integer upper bound is only obtained when the RMP solution happens to be integral at an iteration. Empirically, this rarely occurs with Dantzig or Pessoa, but is common with Template pricing. Regardless, consider UB_t, the best upper bound found at iteration t, we also terminate when the MIP gap is less than 0.001%, that is: $\mathrm{UB}_t - \lceil \max_t\{\mathrm{RMP}_t + \mathrm{rc}_t\}\rceil < 0.00001\,\mathrm{UB}_t$.

4.5 Initialization

Since CG is inherently a primal method, it requires a primal feasible RMP at the start. Typical initialization approaches include heuristic, Phase I (artificial variables with big M), and Farkas pricing. These approaches are discussed by [44,62]. There are benefits and drawbacks to each; we briefly justify our choice.

Heuristics often yield high quality integer solutions; however, may struggle to find feasibility on challenging instances, and are often not ideal starting points for Dantzig or Pessoa pricing (due to integrality). Phase I with big M, requires suitably tuning large costs per instance for best performance. Farkas pricing [1,44], uses the Farkas certificate of infeasibility as the dual values. It avoids tuning, but the construction of dual values is dependent on the implementation underlying LP solver.

Our choice for initialization is equivalent to Farkas pricing, but is best described as Phase I (*without* big M). Specifically, we solve the following RMP; adding artificial variables y_j^+ and y_j^- for $j \in J$ and minimizing their sum. When the optimal objective of this modified RMP is non-zero, the dual variables π and μ give the corresponding Farkas certificate for the original RMP.

$$\begin{aligned}
\min_{\lambda, y} \quad & \sum_{j \in J} \left(y_j^+ + y_j^-\right) & & & \\
\text{s.t.} \quad & \sum_{i \in I} \sum_{\mathbf{v}_{ip} \in P_i} \mathbb{1}_{\{\mathbf{e}_j^\top \mathbf{v}_{ip} = 1\}} \lambda_{ip} \;\geq\; 1 + \left(y_j^+ - y_j^-\right) & \quad & \forall j \in J & \quad (\pi_j) \\
& \sum_{\mathbf{v}_{ip} \in P_i} \lambda_{ip} \;=\; 1 & & \forall i \in I & \quad (\mu_i) \\
& \lambda_{ip} \;\geq\; 0 & & \forall i \in I,\ \mathbf{v}_{ip} \in P_i & \\
& y_j^+, y_j^- \;\geq\; 0 & & \forall j \in J. &
\end{aligned}$$

For the above, recall that in Section 4.3 our implementation uses a set cover formulation.

Since the modified RMP ignores column costs, the pricing subproblems also ignore the costs. Specifically, Dantzig and Template pricing become:

$$\min_{x \in P_i} -\pi^\top x, \text{ and } \min_{x \in S^i_{(\pi,\mu)}} \left(-d\left(x, y^i\right), -\pi^\top x\right).$$

As far as we are aware, Phase I initialization (Farkas pricing) traditionally uses Dantzig pricing until reaching RMP feasibility. While it might be possible to adapt Pessoa pricing to be used during initialization, we do not attempt it in our study. For our Pessoa results, we first initialize the RMP using Dantzig pricing. Template

pricing can be used during Phase I initialization with a minor change. Templates can be easily constructed from the solution of the modified RMP; however, for the initial template (when the modified RMP is initially empty), our implementation uses the LP solution of the compact GAP formulation. Heuristics or other methods could also be used.

The modified RMP is solved iteratively; generating columns every iteration until the optimal objective becomes zero – indicating that a feasible solution to the original RMP exists. At this point, we can remove all artificial variables, and reinstate the correct costs for each column in the RMP. Since the RMP is now initialized with a feasible solution, we continue to solve to optimality.

	Initialization			Convergence to optimality (#its)			
Phase I	#its	% gap	×basis	Dantzig	Pessoa	LT	MT
Dantzig	5	200.8	2.9	3,469	634	2,104	2,315
LT	6	2.0	1.5	4,439	1,296	288	368
MT	5	2.0	1.0	4,618	1,314	295	370

Table 1: Comparison of initialization quality and convergence on the c10400 instance (seed = 0). The ×basis indicates the fractionality of the initial solution (1.0 is integral). % gap stands for the quality of the initial feasible solution discovered compared to the optimal RMP objective value. #its is the number of CG iterations. See Figure 3 for convergence behavior.

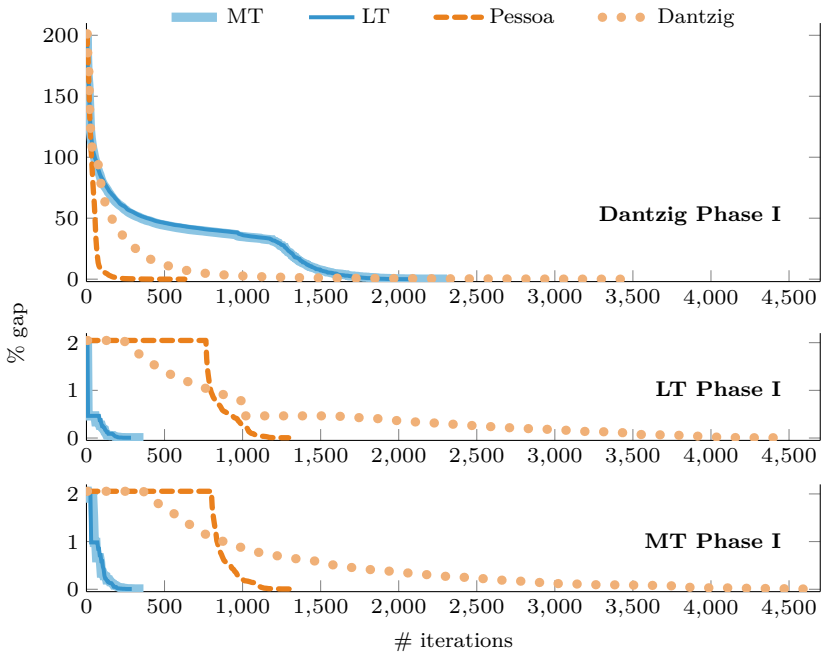

Fig. 3: Convergence of c10400 to optimality with various initialization methods (seed = 0). See Table 1 for additional statistics.

Initialization of the RMP has a significant impact on convergence. For our experiments to be as fair as possible, we consider this impact. Table 1 shows (for a single instance) that Template Phase I finds a near optimal integer initial solution; whereas Dantzig Phase I has significantly worse quality and high fractionality. One might think that using Template Phase I to initialize all methods gives a fair comparison; Table 1 and Figure 3 show that this is not the case. In particular, the integrality of Template Phase I seems to slow convergence for Dantzig and Pessoa pricing methods to converge to optimality; whereas the fractionality of Dantzig Phase I hurts Template pricing methods convergence to the optimal RMP solution. Using integer solutions as a starting basis has long been known [61] to yield poor convergence, so this behavior of Template pricing is interesting.

Note that when the RMP is highly fractional, our chosen template scheme may result in many or all elements set to zero. When this occurs, Template pricing essentially reverts to Dantzig pricing; however, our dynamic scheme implicitly tries to recover the template structure and terminates sooner than Dantzig. Other template schemes could alternatively revert to Pessoa instead of Dantzig; we have chosen this particular scheme for its simplicity.

4.6 Column management

Our implementation aims to add one column per machine at each iteration. After many iterations, however, the RMP can become unnecessarily large. For example, with the instance in Table 1, after 4000 iterations, the RMP could contain 40,000 columns; whereas the RMP basis requires at most 410 columns, and an integer solution could be represented using only 10 columns.

RMP solve times increase with the number of columns, many of which are not needed for convergence. Thus, we periodically remove columns to improve performance. Identifying the minimal set of necessary columns is difficult, so it is performed heuristically. Common choices include age and reduced-cost based retention policies; we use age-based retention for its simplicity.

Age-based retention keeps, for each column, the most recent iteration in which it was in the basis. At each iteration, only basic columns need to be updated. Newly added columns are initialized with the iteration in which they were added, regardless of whether they immediately enter the basis. Retention is controlled by the age threshold τ. At iteration t, we retain columns whose recorded age is in the interval $\{t-\tau, \dots, t\}$, and remove all others.

Larger values of τ retain more columns in the RMP, making it less likely that columns important for approximating the dual polyhedron are removed. Small values of τ can occasionally produce very fast solves when the retained columns happen, by chance, to be sufficient; however, this behavior is unreliable and typically leads to highly volatile performance due both to dual instability and to the repeated regeneration of previously discarded columns. Increasing τ reduces these destabilizing effects and yields more robust performance, at the cost of larger RMPs and increased solve times. We seek a policy that minimizes the RMP solve time while remaining robust, in the sense that small changes in the threshold do not significantly impact performance.

We also considered an activation-frequency parameter controlling when column removal is triggered; however, preliminary experiments indicated that activating at every iteration was sufficient in our setting, with little downside under primal simplex since the basis remains unchanged. We therefore fix activation at every iteration and determine the age threshold for each instance using the policy shown in Figure 4. The chosen threshold reflects an empirically selected near-optimal choice for each pricing rule and job to machine ratio combination in order to reduce the time for CG convergence. Details on how these policies are determined via an extensive parameter sweep are presented in Appendix C.

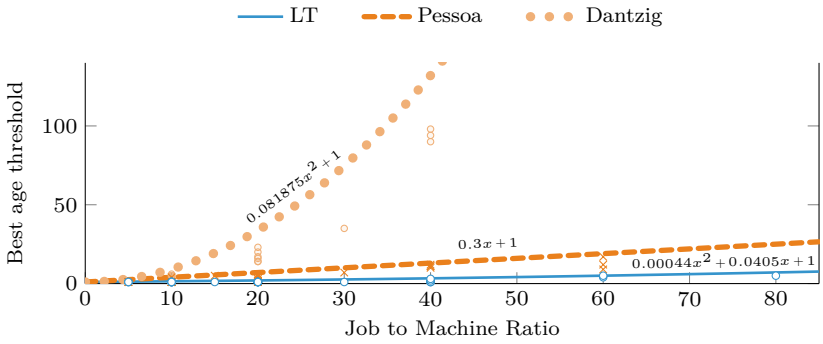

Fig. 4: Chosen age threshold policy vs. degeneracy and algorithm. Policy curves are determined by minimizing the quadratic that covers all data points.

An important takeaway is that stronger dual stabilization reduces sensitivity to age threshold. In particular, Template pricing appears to require the least tuning and is the most robust among the methods we tested.

5 Experimental Results

We present computational experiments that evaluate Template pricing and compare with other approaches. All methods are implemented in C++, with HiGHS 1.11.0 [40] used to solve the RMP. All pricing algorithms share the same knapsack solver to ensure fair comparison – except MT, which solves an IP. The knapsack solver implements a standard dynamic programming algorithm, with the code adapted from SCIP [39]. The experiments were run on an AMD EPYC 7V12 64-Core 1.5 GHz processor with 1TB RAM. Each instance had exclusive access to the hardware and the subproblems were run in parallel.

The details on instances, initialization, termination criteria, and column management have been described in the previous section. Further details of the implementation can be found in the released source code [46].

5.1 Yagiura Results

The experiments on Yagiura use a time limit of 6 hours and have 5 replications (different random seed) per instance, unless otherwise specified. A total of 285 experiments per algorithm. We evaluate Dantzig (D), Pessoa (P), Lagrange Template (LT), and MIP Template (MT) pricing.

5.1.1 Yagiura Initialization

Recall that our RMP uses a set cover instead of a set partition formulation for additional stabilization; moreover, with this choice of formulation, it is also much easier to construct an initial feasible RMP – although, by design, this may include jobs more than once. We compare the impact of formulation choice on Phase I initialization in Table 2.

Phase I	RMP (ms)	Pricing (ms)	# its	% gap	% integral	% timeout
Dantzig	18.1	2.1	4.4	336.9	31.9	0.0
LT	2.3	4.2	3.8	6.5	83.5	0.0
MT	4.0	117.6	4.8	5.0	99.3	0.0

(a) RMP set cover

Phase I	RMP (ms)	Pricing (ms)	# its	% gap	% integral	% timeout
Dantzig	172,987.9	69.2	76.2	103.7	0.0	26.3
LT	34,936.6	128.4	33.8	16.3	49.5	3.2
MT	56.2	603.6	16.9	6.1	97.5	0.0

(b) RMP set partition

Table 2: Yagiura: Comparison of initialization quality. % gap is relative to optimal RMP (excludes timeout). 5 replications, 10 min time limit.

For these experiments we have restricted the time limit to 10 minutes. As can be seen by these results, Template Phase I (both LT and MT) finds high quality initial solutions in similar or fewer iterations; while taking less compute time in the RMP. The Template solutions are also more likely to be integral. The differences between LT and MT are emphasized by the set partition results; the benefits of exact Template Phase I (MT) can be clearly seen, however, heuristic Template Phase I (LT) still dominates over Dantzig Phase I.

5.1.2 Yagiura Results

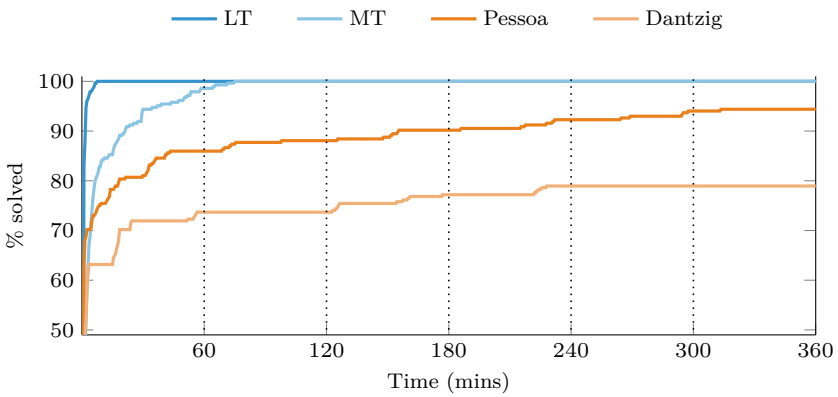

Fig. 5: Yagiura: solved vs solve time.

Figure 5 presents the percentage of instances that solved over the 6-hour time limit. Both LT and MT successfully solve all instances, clearly outperforming D and P pricing methods. In particular, LT solved each instance in less than 7 minutes. Summary statistics are provided in Table 3. Notably, LT and MT requires approximately one-third as many iterations while consuming orders of magnitude less RMP time. This indicates that the reduction in RMP time is not solely attributable to fewer iterations. In particular, consider the number

of pivots required per column and pivots per second; Template pricing requires around 10× fewer pivots, each executed 10× more efficiently.

(a) Computational effort of RMP and pricing

Method	% timeout	# CG its	RMP (s)	Pricing (s)
D	21.1	940	5,285.2	1.0
P	7.0	438	2,642.3	1.6
LT	0.0	325	15.4	5.7
MT	0.0	436	14.9	372.8

(b) Pivot efficiency and integrality of solutions

Method	RMP pivots/col	RMP pivots/s	% integral	integer gap %
D	53.8	215.4	37.9	50.7
P	37.9	222.1	40.0	46.7
LT	3.6	2,363.4	100.0	0.8
MT	2.7	2,019.3	100.0	0.3

Table 3: Yagiura: high-level summary.

A possible explanation for fewer pivots and these pivots being faster with Template pricing is the following. Template pricing aims to generate nearly-compatible sets of columns that are structurally similar to those in the current basis B. Consider the newly generated column a_i for machine $i \in I$. When a_i is structurally similar to the basic column a_{B_r} we have $B^{-1}a_i \approx e_r$, that is, the unit vector with 1 in position r (corresponding to that basic column). The simplex algorithm calculates $B^{-1}a_i$ via FTRAN (forward transformation) and this linear algebra is fastest when both a_i and the resulting vector is sparse; moreover, sparsity implies fewer candidates to evaluate in the ratio-test and ultimately yields a more localized basis change; that is, the first pivot is almost a one-for-one replacement. After this pivot, B^{-1} incurs only a small change, which has multiple benefits. First, this implies only a relatively small perturbation in the duals $c_B^\top B^{-1}$, which is less likely to trigger a cascade of additional improving pivots. At the same time, relatively small changes to B^{-1} requires less bookkeeping for steepest-edge or Devex pricing-weight updates and fewer re-inversions or refactorization of B. So, generated columns that are structurally close to the current basis tend to create sparser transformed columns and sparser update matrices, which means each solve is cheaper, the accumulated basis-factor updates grow more slowly, and costly reinversion and refactorization can be postponed or made cheaper when it does occur. Moreover, at each iteration we add a set of nearly-compatible columns, so the impact is even stronger; each pivot tends to replace different basic columns with limited interaction across rows. These new columns can enter the basis with relatively little "undoing" of previous pivots. From a linear algebra perspective, the pivotal columns remain sparse and separated, so FTRANs, ratio tests, pricing-weight updates, and basis-factor updates all stay relatively cheap.

Furthermore, Template pricing consistently finds high-quality integer solutions for all instances. Finally, Table 4 provides a breakdown of RMP and CG pricing time by job and machine, highlighting the effects of scale and degeneracy (approximated by the jobs/machines ratio).

		RMP (s)				Pricing (s)			
$\|J\|$	$\|I\|$	D	P	LT	MT	D	P	LT	MT
100	5	1.6	0.3	0.1	0.2	0.2	0.1	0.1	37.0
	10	0.4	0.2	0.0	0.1	0.1	0.0	0.0	11.6
	20	0.2	0.1	0.0	0.1	0.0	0.0	0.0	6.8
200	5	96.5	6.7	0.8	1.0	1.3	0.5	0.9	108.6
	10	12.5	1.8	0.3	0.4	0.2	0.2	0.4	45.6
	20	3.0	1.0	0.2	0.3	0.1	0.1	0.1	25.5
400	10	957.1	67.0	3.8	4.1	3.6	1.6	3.7	223.0
	20	131.5	19.3	1.6	2.1	0.6	0.4	1.4	198.3
	40	32.3	8.8	0.9	1.3	0.2	0.2	0.4	88.7
900	15	21,596.0	4,147.4	26.3	42.2	3.0	11.7	16.2	1,180.3
	30	10,769.0	864.7	9.8	12.8	4.6	3.0	7.0	600.1
	60	1,946.5	302.7	5.5	7.2	1.0	0.9	2.2	454.1
1600	20	21,600.0	21,597.5	163.9	144.6	1.4	3.5	48.1	1,398.7
	40	21,598.0	16,788.3	46.7	42.9	1.4	5.3	15.7	1,238.9
	80	21,598.0	6,390.4	32.4	21.9	0.6	3.1	10.2	1,308.8

Table 4: Yagiura: RMP and Pricing time (s) for different job/machine

5.2 ISA Instances

The ISA experiments use a time limit of 1 hour and perform a single replication per instance. Thus, a total of 1735 experiments per algorithm were conducted. We primarily compare Pessoa (P) and Lagrange Template (LT), however also include results from MIP Template (MT) and Lagrangian Relaxation (LR) where appropriate.

5.2.1 ISA Initialization

We perform additional experiments to highlight the quality of initialization. Table 5 shows a comparison of the various Phase I initialization with a time limit of 10 minutes. Dantzig Phase I fails to find a primal feasible RMP within this time limit for 2.1% of the instances. In a separate experiment, we found that some of these instances required over 74 hours to initialize a feasible RMP. Template Phase I initialized all instances within the time limit; spending a maximum of around 1 minute for LT and 5 minutes for MT.

As expected, Template Phase I finds initial solutions with significantly lower gap, and are much more likely to be integral – requiring less time and similar (or fewer) iterations. The quality difference can be seen between the heuristic LT and the exact method MT.

Phase I	RMP (s)	Pricing (s)	# its	% gap	% integral	% timeout
Dantzig	20.3	0.1	6.2	1,156.5	34.8	2.1
LT	0.1	0.1	4.2	17.3	68.2	0.0
MT	0.8	1.7	7.9	11.2	89.0	0.0

Table 5: ISA instances: Comparison of initialization statistics on ISA instances. % gap is relative to optimal RMP bound. 10 min time limit.

5.2.2 ISA Results

Figure 6 shows the percentage of instances that solved over the 1-hour time limit. Many of these instances are larger than those from Yagiura, and some were intentionally constructed to be more challenging by making feasible integer solutions harder to find. LT performs well, solving nearly all instances; whereas LR quickly finds good bounds but fails to converge to the optimal. Pessoa solves 58% of the instances. Extrapolating from the results in Yagiura, we expect Dantzig would perform considerably worse. See Table 6 for detailed results. In particular, we note that the RMP pivots are faster and fewer for LT than that of Pessoa, as also observed and discussed for the Yagiura instances in Section 5.1.2. Similarly, Table 7 shows the results only on the 824 instances that were solved by all methods. These instances can be considered 'easy', and we expect the impact of LT is more pronounced in the harder instances.

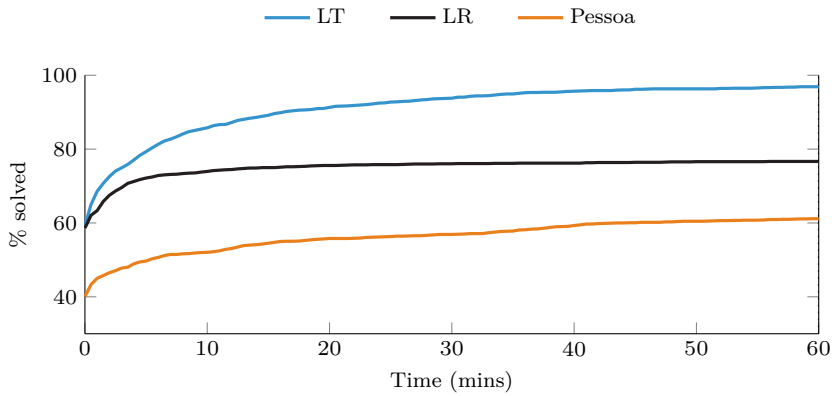

Fig. 6: ISA: Solved vs solve time. Instances where LR does not reach optimal bound are treated as timed-out.

In Table 6, LR does not actually have a RMP nor generate columns per se, but there is a similar analogue, since at each iteration it solves the pricing problem for each machine (Pricing), and uses this information to update the multipliers (RMP). LR spends most of its time in CG pricing, with significantly more iterations; note these iterations (# CG its) include the backtracking linesearch within L-BFGS. Although LR uses the

(a) Computational effort of RMP and pricing

Method	% solved	# CG its	RMP (s)	Pricing (s)
P	58.3	245	1,631.0	3.1
LT	96.9	422	246.3	81.0
LR	76.7	7,768	2.9	409.0

(b) Pivot efficiency and integrality of solutions

Method	RMP pivots/col	RMP pivots/s	% integral	integer gap %
P	53.3	310.0	39.6	41.3
LT	5.1	955.7	98.3	1.3
LR	-	-	6.5	0.8

Table 6: ISA: high-level statistics for all 1735 instances

(a) Computational effort of RMP and pricing

Method	% solved	# CG its	RMP (s)	Pricing (s)
P	100.0	341	230.9	1.1
LT	100.0	226	3.0	2.3
LR	100.0	1,531	0.2	4.4

(b) Pivot efficiency and integrality of solutions

Method	RMP pivots/col	RMP pivots/s	% integral	integer gap %
P	23.0	1,227.4	45.6	29.8
LT	2.9	7,074.8	99.9	0.9
LR	-	-	12.4	0.9

Table 7: ISA: high-level statistics for the 824 'easier' instances that were solved by all algorithms

same pricing algorithm, 0.5% of the instances could not be solved by LR due to memory errors in the DP subproblem; these issues were not seen in the CG methods.

LR seldom found integer solutions implicitly, whereas LT consistently produces high-quality solutions across almost all instances. For the 7% of instances where LR obtained integer solutions, their quality was high; however, these instances were trivial to solve. This can be seen in Table 8, which explores how class and scale (number of jobs) impact outcomes. Large number of jobs correlate with higher degeneracy, although difficulty also varies between classes. The 'f' class instances, specifically designed to impede finding integer feasible solutions, proved challenging for all algorithms, though LT managed them best. As mentioned, LR only found integral solutions in trivial instances, that is, the 'easier' classes with smaller number of jobs. As a dual method, LR generally does not produce feasible solutions without additional heuristics. In Pessoa's case, achieving integrality appears random and unrelated to scale or degeneracy; however it failed to find any feasible integer solutions for the f class instances, and surprisingly also struggled on c class.

When LR does not converge to the optimal bound, we treat it as a timeout. Table 8 highlights when this is the case, as it shows many timeouts even though the time to solve is relatively small. This accounts for 15% of instances, where 13% converged to a near optimal bound, while the remaining were far from optimal (the n class in particular). LR is known to quickly give excellent bounds, while having issues with convergence; this is again shown here. Even considering this, LT still dominates with its stronger convergence guarantees, but it can take longer to reach similar bounds. A hybrid LR/LT approach could be very promising.

6 Conclusion and Future Work

It is worth emphasizing that Template pricing, while primarily discussed for set partitioning problems, can also be applied to other problem classes. The approach proves particularly beneficial when faced with highly degenerate cases. Our similarity function $d(x, y^i)$, was chosen for its simplicity, other choices could improve the results even further, especially on instances that have high degree of inherent fractionality at the root CG node.

Our Template pricing implementation efficiently uses a Lagrangian heuristic; however, other approaches may be more suitable when the subproblems are complex. Template pricing shares similarities to local search, that is, they both try to find suitable solutions without deviating too far from some baseline. Local search around active columns is known to be a successful strategy [5,62,63], typically with the assumption that checking valid perturbations (also known as exchange vectors) are much cheaper than solving the subproblem. While this is true, we have shown there can be additional benefits when solving the RMP. Our ideas could be easily incorporated into local search heuristics designed to approximate Template pricing for complex subproblems, for even further impact.

		RMP (s)			Pricing (s)			% integral			% timeout		
class	$\|J\|$	P	LT	LR	P	LT	LR	P	LT	LR	P	LT	LR
a	100	0.2	0.0	0.0	0.0	0.0	0.0	87	100	43	0	0	0
	200	2.2	0.0	0.0	0.1	0.0	0.0	63	100	13	0	0	0
b	100	0.2	0.0	0.0	0.1	0.0	0.0	57	100	30	0	0	10
	200	2.8	0.3	0.0	0.2	0.2	0.1	20	100	10	0	0	10
c	100	0.2	0.0	0.0	0.0	0.0	0.1	13	100	17	0	0	13
	200	2.7	0.3	0.0	0.2	0.2	0.1	7	100	7	0	0	10
	400	18.9	0.7	0.1	0.3	0.3	0.2	0	100	0	0	0	3
	900	1,048.0	7.3	0.1	1.2	1.5	0.4	0	100	0	0	0	3
	1600	2,903.6	42.0	0.4	0.6	3.8	1.2	0	100	0	67	0	0
d	100	0.2	0.1	0.0	0.1	0.2	0.2	43	100	0	0	0	10
	200	4.0	1.1	0.0	0.6	1.6	0.4	30	100	0	0	0	3
	400	39.0	3.6	0.1	1.5	4.1	1.6	30	100	0	0	0	3
	900	1,783.4	19.4	0.4	5.3	15.0	14.7	47	100	0	33	0	0
	1600	3,597.6	114.6	1.0	2.6	74.0	73.4	47	100	0	100	0	3
e	100	0.2	0.1	0.0	0.1	0.0	0.1	43	100	0	0	0	37
	200	3.9	0.5	0.0	0.2	0.3	0.2	37	100	7	0	0	13
	400	36.6	1.8	0.1	0.4	0.9	0.3	50	100	0	0	0	7
	900	1,618.9	14.4	0.2	1.5	5.0	1.2	60	100	0	33	0	0
	1600	3,598.0	114.4	0.7	0.8	17.6	9.3	50	100	0	100	0	7
f	900	83.3	4.2	5.5	2.0	3.7	173.3	0	82	0	0	0	100
	1600	1,319.5	17.5	10.5	9.6	24.2	737.8	0	77	0	0	0	100
	3200	3,580.7	200.7	20.3	2.2	164.2	3,006.0	0	91	0	100	0	100
	4000	3,566.4	478.0	21.7	2.4	260.8	3,569.0	0	84	0	100	0	100
	5000	3,551.2	1,401.8	19.8	3.0	464.2	3,580.6	0	84	0	100	23	100
n	100	0.2	0.1	0.0	0.1	0.2	0.1	55	100	30	0	0	29
	200	3.7	0.5	0.0	0.4	1.1	0.4	43	100	16	0	0	26
	400	28.6	1.9	0.1	1.2	2.9	1.9	58	100	8	0	0	18
	900	1,055.9	11.6	0.4	5.5	14.0	16.5	45	99	6	16	0	14
	1600	2,674.4	93.7	1.1	4.4	44.2	69.4	43	100	4	64	1	9
	3200	3,352.1	255.4	3.9	5.1	188.3	612.5	53	99	2	93	0	29
	4000	3,325.2	602.4	4.2	4.7	173.3	530.0	47	99	1	93	5	19
	5000	3,258.9	1,348.0	5.2	10.3	280.0	716.0	41	99	2	92	25	27

Table 8: ISA: Statistics per class and number of jobs. Instances where LR terminated early without finding optimal are treated as timeout.

Another aspect that we have not addressed is that of a branch-and-price implementation. Finding ways to leverage templates within branching decisions presents an intriguing direction for future research. Empirically, Template pricing is more likely to find integer solutions, so we imagine it could help reduce the branch-and-bound search. There are several other interesting research areas to explore. For example: combining Template pricing with other stabilization methods (like Pessoa); hybrid solutions with Lagrangian Relaxation providing initial dual bounds; modifications to the RMP (similar to DCA [27], or trust region hybrids).

Lastly, we have focused our research on problem classes with independent subproblems, where the number of subproblems is known. There are several problem classes that share subproblems across multiple columns, for example, bin packing or graph coloring. In these problems, it is interesting to consider how a template is defined. We imagine that it could involve clustering the columns based on similarity within the current primal solution, or other heuristic approaches. Finally, a potential connection between recent theoretical results on escaping degeneracy for the simplex method, for example [41], and our proposed Template pricing approach could be an interesting direction to examine.

In closing, we have proposed Template pricing as a coordinated pricing paradigm for column generation that markedly accelerates convergence while incidentally finding high-quality integer solutions. We present both exact and heuristic variants, and show that their effectiveness stems from improved column compatibility and stability, which in turn speeds up the solution of the RMP. We further analyze the roles of initialization, column management, and degeneracy in overall performance. Computational results on GAP benchmarks show consistent and sometimes orders-of-magnitude improvements over Dantzig and Pessoa pricing. Moreover, Template pricing is competitive with Lagrangian relaxation techniques, which have traditionally been the preferred approach for highly degenerate problems.

References

1. Achterberg, T.: Scip: solving constraint integer programs. Mathematical Programming Computation **1**, 1–41 (2009)
2. Amor, H.B., Desrosiers, J., Frangioni, A.: Stabilization in column generation. Les Cahiers du GERAD ISSN **711**, 2440 (2004)
3. Amor, H.M.B., Desrosiers, J., Frangioni, A.: On the choice of explicit stabilizing terms in column generation. Discrete Applied Mathematics **157**(6), 1167–1184 (2009)
4. Baldacci, R., Christofides, N., Mingozzi, A.: An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. Mathematical Programming **115**(2), 351–385 (2008)
5. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W., Vance, P.H.: Branch-and-price: Column generation for solving huge integer programs. Operations research **46**(3), 316–329 (1998)
6. Beasely, J.E.: OR Library. https://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html, accessed: 2024-12-05
7. Ben Amor, H., Desrosiers, J., Valério de Carvalho, J.M.: Dual-optimal inequalities for stabilized column generation. Operations Research **54**(3), 454–463 (2006)
8. Benchimol, P., Desaulniers, G., Desrosiers, J.: Stabilized dynamic constraint aggregation for solving set partitioning problems. European Journal of Operational Research **223**(2), 360–371 (2012)
9. Bixby, R.E., Gregory, J.W., Lustig, I., Marsten, R.E., Shanno, D.F.: Very large-scale linear programming: A case study in combining interior point and simplex methods. Oper. Res. **40**, 885–897 (1992), https://api.semanticscholar.org/CorpusID:31277854
10. Bixby, R.E., Gregory, J.W., Lustig, I.J., Marsten, R.E., Shanno, D.F.: Very large-scale linear programming: A case study in combining interior point and simplex methods. Operations research **40**(5), 885–897 (1992)
11. Black, A.E.: Exponential lower bounds for many pivot rules for the simplex method. In: International Conference on Integer Programming and Combinatorial Optimization. pp. 86–99. Springer (2025)
12. Bouarab, H., El Hallaoui, I., Metrane, A., Soumis, F.: Dynamic constraint and variable aggregation in column generation. European Journal of Operational Research **262**(3), 835–850 (Nov 2017). https://doi.org/10.1016/j.ejor.2017.04.049
13. Briant, O., Lemaréchal, C., Meurdesoif, P., Michel, S., Perrot, N., Vanderbeck, F.: Comparison of bundle and classical column generation. Mathematical programming **113**(2), 299–344 (2008)
14. Valério de Carvalho, J.: Exact solution of bin-packing problems using column generation and branch-and-bound. Annals of Operations Research **86**(0), 629–659 (1999)
15. Cattrysse, D.G., Van Wassenhove, L.N.: A survey of algorithms for the generalized assignment problem. European journal of operational research **60**(3), 260–272 (1992)
16. Cattrysse, D., Salomon, M., Van Wassenhove, L.N.: A set partitioning heuristic for the generalized assignment problem. European Journal of Operational Research **72**(1), 167–174 (Jan 1994). https://doi.org/10.1016/0377-2217(94)90338-7
17. Chu, P.C., Beasley, J.E.: A genetic algorithm for the generalised assignment problem. Computers & Operations Research **24**(1), 17–23 (1997)
18. Chu, P., Beasley, J.: A genetic algorithm for the generalised assignment problem. Computers & Operations Research **24**(1), 17–23 (Jan 1997). https://doi.org/10.1016/S0305-0548(96)00032-9
19. Costa, L., Contardo, C., Desaulniers, G., Yarkony, J.: Stabilized column generation via the dynamic separation of aggregated rows. INFORMS Journal on Computing **34**(2), 1141–1156 (2022)
20. Dadush, D., Huiberts, S.: A friendly smoothed analysis of the simplex method. In: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing. pp. 390–403 (2018)
21. Dantzig, G.B., Wolfe, P.: Decomposition principle for linear programs. Operations research **8**(1), 101–111 (1960)
22. Desaulniers, G., Desrosiers, J., Dumas, Y., Marc, S., Rioux, B., Solomon, M.M., Soumis, F.: Crew pairing at air france. European Journal of Operational Research **97**(2), 245–259 (1997)
23. Desaulniers, G., Desrosiers, J., Solomon, M.M.: Column generation, vol. 5. Springer Science & Business Media (2006)
24. Desrosiers, J., Lübbecke, M., Desaulniers, G., Gauthier, J.B.: BRANCH-AND-PRICE. SPRINGER INTERNATIONAL PU, S.l. (2025), oCLC: 1520195899
25. Du Merle, O., Villeneuve, D., Desrosiers, J., Hansen, P.: Stabilized column generation. Discrete Mathematics **194**(1-3), 229–237 (1999)
26. Elhallaoui, I., Metrane, A., Soumis, F., Desaulniers, G.: Multi-phase dynamic constraint aggregation for set partitioning type problems. Mathematical Programming **123**, 345–370 (2010)
27. Elhallaoui, I., Villeneuve, D., Soumis, F., Desaulniers, G.: Dynamic aggregation of set-partitioning constraints in column generation. Operations Research **53**(4), 632–645 (2005)
28. Fisher, M.L.: The lagrangian relaxation method for solving integer programming problems. Management science **27**(1), 1–18 (1981)
29. Forrest, J.J., Goldfarb, D.: Steepest-edge simplex algorithms for linear programming. Mathematical programming **57**(1), 341–374 (1992)
30. Fukasawa, R., He, Q., Song, Y.: A branch-cut-and-price algorithm for the energy minimization vehicle routing problem. Transportation Science **50**(1), 23–34 (2016)
31. Geibinger, T., Kletzander, L., Musliu, N.: Instance space analysis for the generalized assignment problem. In: Di Gaspero, L., Festa, P., Nakib, A., Pavone, M. (eds.) Metaheuristics. pp. 421–435. Springer International Publishing, Cham (2023)
32. Geoffrion, A.M.: Lagrangean relaxation for integer programming. In: Approaches to integer programming, pp. 82–114. Springer (2009)
33. Ghoniem, A., Sherali, H.D.: Complementary column generation and bounding approaches for set partitioning formulations. Optimization Letters **3**(1), 123–136 (Jan 2009). https://doi.org/10.1007/s11590-008-0097-2
34. Goldfarb, D., Reid, J.K.: A practicable steepest-edge simplex algorithm. Mathematical Programming **12**, 361–371 (1977)
35. Gschwind, T., Irnich, S.: Dual inequalities for stabilized column generation revisited. INFORMS Journal on Computing **28**(1), 175–194 (2016)
36. Haase, K., Desaulniers, G., Desrosiers, J.: Simultaneous vehicle and crew scheduling in urban mass transit systems. Transportation science **35**(3), 286–303 (2001)
37. Harris, P.M.J.: Pivot selection methods of the devex lp code. Mathematical Programming **5**, 1–28 (1973), https://api.semanticscholar.org/CorpusID:45065141
38. Held, S., Cook, W., Sewell, E.C.: Maximum-weight stable sets and safe lower bounds for graph coloring. Mathematical Programming Computation **4**(4), 363–381 (2012)
39. Hojny, C., Besançon, M., Bestuzheva, K., Borst, S., Chmiela, A., Dionísio, J., Eifler, L., Ghannam, M., Gleixner, A., Göß, A., Hoen, A., van der Hulst, R., Kamp, D., Koch, T., Kofler, K., Lentz, J., Maher, S.J., Mexi, G., Mühmer, E., Pfetsch, M.E., Pokutta, S., Serrano, F., Shinano, Y., Turner, M., Vigerske, S., Walter, M., Weninger, D., Xu, L.: The SCIP Optimization Suite 10.0. Technical report, Optimization Online (November 2025), https://optimization-online.org/2025/11/the-scip-optimization-suite-10-0/

40. Huangfu, Q., Hall, J.A.J.: Parallelizing the dual revised simplex method. Mathematical Programming Computation **10**(1), 119–142 (Mar 2018). https://doi.org/10.1007/s12532-017-0130-5
41. Kukharenko, K., Sanità, L.: On the number of degenerate simplex pivots. In: International Conference on Integer Programming and Combinatorial Optimization. pp. 252–264. Springer (2024)
42. Li, Y., Clarke, J.P., Dey, S.S.: Using submodularity within column generation to solve the flight-to-gate assignment problem. Transportation Research Part C: Emerging Technologies **129**, 103217 (2021)
43. Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Math. Program. **45**(1–3), 503–528 (Aug 1989)
44. Lübbecke, M.E.: Column Generation. In: Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Ltd (2011). https://doi.org/10.1002/9780470400531.eorms0158, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470400531.eorms0158
45. Lübbecke, M.E., Desrosiers, J.: Selected Topics in Column Generation. Operations Research **53**(6), 1007–1023 (Dec 2005). https://doi.org/10.1287/opre.1050.0234
46. Marshall, L.: Template pricing (2026), https://github.com/mathgeekcoder/template-pricing
47. Mehrotra, A., Trick, M.A.: A column generation approach for graph coloring. informs Journal on Computing **8**(4), 344–354 (1996)
48. Neame, P.: Nonsmooth Dual Methods in Integer Programming. University of Melbourne, Department of Mathematics and Statistics (2000)
49. Öncan, T.: A survey of the generalized assignment problem and its applications. INFOR: Information Systems and Operational Research **45**(3), 123–141 (2007)
50. Pecin, D., Pessoa, A., Poggi, M., Uchoa, E.: Improved branch-cut-and-price for capacitated vehicle routing. Mathematical Programming Computation **9**(1), 61–100 (2017)
51. Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F.: Automation and Combination of Linear-Programming Based Stabilization Techniques in Column Generation. INFORMS Journal on Computing **30**(2), 339–360 (May 2018). https://doi.org/10.1287/ijoc.2017.0784
52. Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F.: In-out separation and column generation stabilization by dual price smoothing. In: Experimental Algorithms: 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings 12. pp. 354–365. Springer (2013)
53. Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F.: Automation and combination of linear-programming based stabilization techniques in column generation. INFORMS Journal on Computing **30**(2), 339–360 (2018)
54. Pigatti, A., De Aragao, M.P., Uchoa, E.: Stabilized branch-and-cut-and-price for the generalized assignment problem. Electron. Notes Discret. Math. **19**, 389–395 (2005)
55. Rousseau, L.M., Gendreau, M., Feillet, D.: Interior point stabilization for column generation. Operations Research Letters **35**(5), 660–668 (2007)
56. Savelsbergh, M.: A branch-and-price algorithm for the generalized assignment problem. Operations research **45**(6), 831–841 (1997)
57. Spielman, D.A., Teng, S.H.: Smoothed analysis: an attempt to explain the behavior of algorithms in practice. Communications of the ACM **52**(10), 76–84 (2009)
58. Uchoa, E., Pessoa, A., Moreno, L.: Optimizing with Column Generation: Advanced branch-cut-and-price algorithms (Part I). Tech. Rep. L-2024-3, Cadernos do LOGIS-UFF, Universidade Federal Fluminense, Engenharia de Produção (August 2024)
59. Vanderbeck, F., Wolsey, L.A.: Reformulation and decomposition of integer programs. Springer (2010)
60. Vanderbeck, F.: Decomposition and column generation for integer programs. Ph.D. thesis, UCL-Université Catholique de Louvain (1994)
61. Vanderbeck, F.: Decomposition and column generation for integer programs. Ph.D. thesis, Universite' Catholique do Lovain (1994)
62. Vanderbeck, F.: Implementing Mixed Integer Column Generation. In: Desaulniers, G., Desrosiers, J., Solomon, M.M. (eds.) Column Generation, pp. 331–358. Springer US, Boston, MA (2005)
63. Vanderbeck, F., Savelsbergh, M.W.P.: A generic view of Dantzig–Wolfe decomposition in mixed integer programming. Operations Research Letters **34**(3), 296–306 (May 2006)
64. Wentges, P.: Weighted dantzig-wolfe decomposition for linear mixed-integer programming. International Transactions in Operational Research **4**(2), 151–162 (1997)
65. Yagiura, M.: GAP (generalized assignment problem) instances. http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/gap/, accessed: 2024-12-05
66. Yagiura, M., Ibaraki, T., Glover, F.: A path relinking approach with ejection chains for the generalized assignment problem. European journal of operational research **169**(2), 548–569 (2006)

7 Statements and Declarations

7.1 Funding

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

7.2 Competing Interests

The authors have no relevant financial or non-financial interests to disclose.

7.3 Author Contributions

All authors contributed to the study conception and design. Implementation, material preparation, data collection and analysis were performed by Luke Marshall. An initial draft of the manuscript was prepared by Prachi Shah and substantially revised by Luke Marshall (incorporating additional algorithms and results), with input from all authors. All authors read and approved the final manuscript.

7.4 Data Availability

Datasets generated and analyzed during the current study are available in the https://github.com/mathgeekcoder/template-pricing repository.

A Pessoa Directional Dual Smoothing

This section provides the full details of the directional dual smoothing algorithm defined by Pessoa, which is used in our computational study. We include these details to clearly document the precise variant implemented, as Pessoa [51] describe multiple stabilization schemes. See Figure 7 for a geometric interpretation.

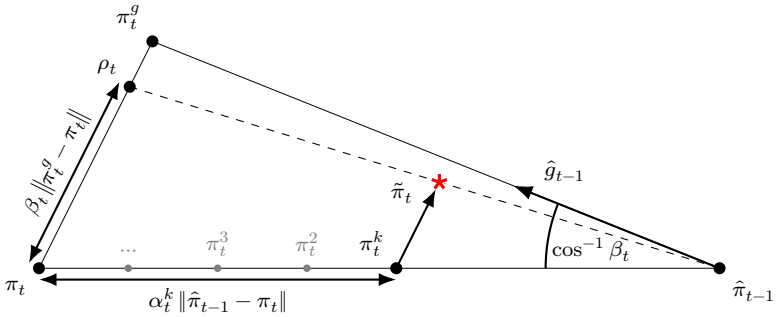

Fig. 7: Geometric view of Pessoa's directional smoothing and limited α_t^k backtracking line search. Terminates at the first point that yields a column with good reduced cost. Backtracking starts at *, when the gradient is available.

At each CG iteration t, we maintain the following information:

π_t dual vector obtained from solving the RMP
$\tilde{\pi}_t$ smoothed dual vector used for pricing
$\hat{\pi}_t$ "best" dual vector corresponding to the most recent iteration at which the RMP objective strictly improved
$\hat{g}_t$ subgradient associated with $\hat{\pi}_t$
α_t adaptive mixing parameter
k backtracking index controlling smoothing intensity

For GAP, the pricing subproblems are decomposed by machine. Let $\{v_i^t\}_{i\in I}$ denote the set of columns produced by pricing at iteration t, one column for each machine $i \in I$. Each subgradient element corresponding to job $j \in J$ is defined as:

$$g_{tj} = \nabla \pi_{tj} = 1 - \sum_{i\in I} \mathbb{1}_{\{\mathbf{e}_j^\top v_i^t = 1\}}.$$

This is the same definition used by Lagrangian Relaxation in Section 2.1. Intuitively, when:

$g_{tj} = 1$ job j was not selected by any machine
$g_{tj} < 0$ job j was selected by multiple machines
$g_t = 0$ the set of columns $\{v_i^t\}_{i\in I}$ partition the jobs across machines

Let $\hat{\pi}_{t-1}$ be the previous best dual vector and $\hat{g}_{t-1}$ be its associated subgradient. Pessoa combines Wentges with an adaptive mixing and a directional step along the subgradient. First, consider the family of convex combinations, corresponding to the backtracking index $k = 1, \ldots, 9$ and the previous adaptive mixing parameter α_{t-1}:

$$\alpha_t^k = \max\{0,\ 1 - k\,(1 - \alpha_{t-1})\}$$
$$\pi_t^k = \alpha_t^k \hat{\pi}_{t-1} + (1 - \alpha_t^k)\,\pi_t$$

Next, if $\|\hat{g}_t\| > 0$, the directional step is calculated as:

$$\pi_t^g = \hat{\pi}_{t-1} + \|\pi_t - \hat{\pi}_{t-1}\| \frac{\hat{g}_{t-1}}{\|\hat{g}_{t-1}\|}$$
$$\beta_t = \frac{(\pi_t - \hat{\pi}_{t-1}) \cdot (\pi_t^g - \hat{\pi}_{t-1})}{\|\pi_t - \hat{\pi}_{t-1}\| \|\pi_t^g - \hat{\pi}_{t-1}\|}$$
$$\rho_t = \beta_t \pi_t^g + (1 - \beta_t)\pi_t$$

such that the smoothed dual vector is given by:

$$\tilde{\pi}_t = \max\left\{\mathbf{0},\ \hat{\pi}_{t-1} + \|\pi_t^k - \hat{\pi}_{t-1}\| \frac{\rho_t - \hat{\pi}_{t-1}}{\|\rho_t - \hat{\pi}_{t-1}\|}\right\}$$

Non-negativity is enforced in the definition above, that is, $\max\{\mathbf{0},\ \ldots\}$ is taken element wise.

Pessoa performs a limited backtracking search to guarantee finding at least one column having negative reduced cost (if one exists). That is, minimizing k such that there exists some $i \in I$ having

$$x^i = \arg\min_{\mathbf{x} \in P_i} \sum_{j \in J} (c_{ij} - \tilde{\pi}_{tj}) x_j,$$
$$\text{with } \sum_{j \in J} (c_{ij} - \pi_{tj})\, x_j^i \leq \mu_{ti} - \varepsilon.$$

However, if we require any backtracking, that is, $k > 1$, or when $\|\hat{g}_t\| = 0$, the smoothed dual vector is simply given by:

$$\tilde{\pi}_t = \pi_t^k$$

Notice that as k increases (more backtracking), $\alpha_t^k \to 0$ and $\tilde{\pi}_t \to \pi_t$. If no negative reduced columns can be found for all $k \in \{1, \ldots, 9\}$, Pessoa reverts to Dantzig pricing:

$$\tilde{\pi}_t = \pi_t$$

The final step is to update the adaptive mixing parameter for the next iteration:

$$\alpha_t = \begin{cases} \min\{0.9999,\ 0.9\alpha_{t-1} + 0.1\} & \text{if } \tilde{g}_t \cdot (\pi_t - \hat{\pi}_{t-1}) > 0 \\ \max\{0,\ \alpha_{t-1} - 0.1\} & \text{o/w} \end{cases}$$

B Solving Template pricing exactly via hierarchical optimization

We can solve the Template pricing problem exactly using a hierarchical multiple objective integer program. Formally, here is the sequence of optimization problems we solve:

1. Verify $S^i_{(\pi,\mu)} \neq \emptyset$, that is, solve the Dantzig pricing problem (4). If non-negative reduced costs, then we have $S^i_{(\pi,\mu)} = \emptyset$ and we stop.
2. Else, optimize template objective over the *good* columns, that is, solve (7):

$$\begin{aligned} \text{OPT} := \max\ & d(x, y^i) \\ \text{s.t.}\ & x \in S^i_{(\pi,\mu)} \equiv (3) \end{aligned}$$

3. Finally, break ties among the optimal solutions of (7), that is, solve (6):

$$\begin{aligned} \min\ & \sum_{j \in J} (c_{ij} - \pi_j)\, x_j \\ \text{s.t.}\ & d(x, y^i) \geq \text{OPT} \\ & x \in P_i \end{aligned} \tag{11}$$

Due to numerical issues, (11) can fail at times to find a column with negative reduced costs, even when one exists. In these cases, we reduce the value of OPT by 1 and re-solve (11) until a suitable column is found.

C Column management policy

This section describes how the column management policy mentioned in Section 4.6 was constructed.

Recall that our implementation uses age-based retention. For each column, we record its age, equal to the most recent iteration in which it was in the optimal RMP basis; or the iteration when it was first added to the RMP. At iteration t, columns are retained when their recorded age is in the interval $\{t - \tau, \ldots, t\}$, where τ denotes the age threshold.

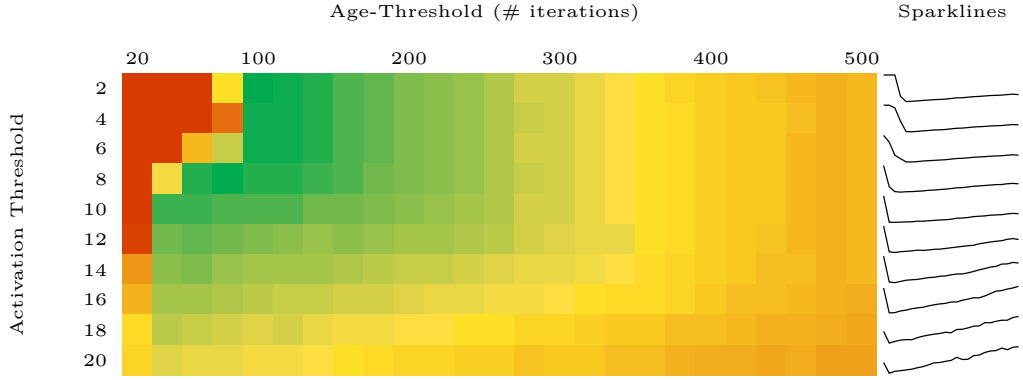

Fig. 8: Column management heatmap for Dantzig pricing on instance c10400. Solve time is shown as a function of age threshold and activation frequency. Green and yellow correspond to shorter solve times, while red indicates 1-hour timeout. Each cell is a locally smoothed average of 200 replications including nearby parameter settings. The sparklines on the right show the relative solve-time profile for each activation frequency setting.

In addition to τ, we also considered an activation-frequency parameter controlling when column removal is triggered. This parameter was implemented as a multiplier on the number of rows in the RMP, following guidance from the literature [24], so that activation scales appropriately with problem size. We first performed a two-parameter sweep over age threshold and activation frequency on representative instances. Figure 8 shows an example. These preliminary experiments indicate that, in our setting, activating column removal at every iteration was sufficient to minimize solve times. Since the RMP is solved via primal simplex, this choice has little downside in practice because the basis itself remains unchanged. Accordingly, we fixed activation at every iteration in the following experiments and those reported in the main text.

The choice for age threshold τ is a tradeoff. Small values of τ can occasionally yield very fast solves when the retained columns happen, by chance, to be sufficient; however, this behavior is unreliable and typically leads to volatile performance. Larger values of τ are more likely to retain columns that provide a good approximation of the dual polyhedron; improving robustness, at the cost of larger RMPs and increased solve times.

To calibrate τ, we performed parameter sweeps on a representative set of instances. For each instance and candidate τ, we ran five replications and measured time to convergence. To reduce noise, we computed the geometric mean across replications and then applied a centered rolling weighted average (across five data points). We selected the smallest threshold whose smoothed value was effectively tied with the minimum, where ties were defined as being within 1% or 1 second of the best observed value. Figure 9 illustrates the selection procedure on a single instance.

We repeated this procedure for Dantzig, Pessoa, and LT pricing. Our assumption is that the best age-threshold policy is driven by instance degeneracy, for which we use the job-to-machine ratio as a proxy. To define our policy, rather than fitting a polynomial to the selected thresholds, we construct a quadratic *covering* policy. Specifically, for each method, we collect the selected threshold from every instance, and then solve a quadratic program to find polynomial coefficients such that the resulting curve covers these data points while remaining as small as possible. We use a cover rather than a fit because a fitted curve could assign τ values below the empirically selected "best" thresholds, thereby increasing the risk of choosing values that are too aggressive and hence too volatile. Requiring the policy to lie above the selected thresholds yields a more conservative and robust rule.

Figure 4 shows the chosen policy and the selected thresholds (per instance) against the job-to-machine ratio. The chosen thresholds increase with degeneracy, and are well approximated by a quadratic function. These quadratic covering curves define the age-threshold policy used in the experiments in Section 5.

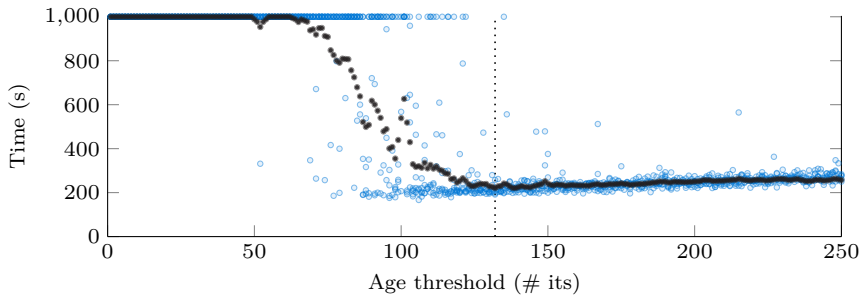

Fig. 9: Age-threshold sweep for Dantzig pricing on instance a05200. Solve time is shown as a function of the age threshold using five replications and a 1000-second time limit. Black solid dots show the centered rolling geometric mean (window size 5), and the dashed vertical line marks the selected threshold. The figure illustrates the high variability at small thresholds and the robustness-based criteria used to select the threshold.

D Full Yagiura results

			RMP (s)				Pricing (s)				× Time vs LT	
class	$\|J\|$	$\|I\|$	D	P	LT	MT	D	P	LT	MT	D	P
a	100	5	2.0	0.2	0.0	0.0	0.2	0.1	0.0	0.0	452	59
		10	0.5	0.1	0.0	0.0	0.1	0.0	0.0	1.4	18	6
		20	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	44	20
	200	5	117.9	5.0	0.0	0.0	0.9	0.2	0.0	0.0	23,224	1,022
		10	13.7	1.2	0.0	0.0	0.2	0.1	0.0	0.1	2,380	211
		20	3.3	1.2	0.0	0.0	0.1	0.1	0.0	1.6	64	24
b	100	5	1.5	0.3	0.1	0.2	0.2	0.1	0.1	37.7	11	3
		10	0.4	0.1	0.0	0.1	0.1	0.0	0.0	3.7	9	3
		20	0.2	0.1	0.0	0.0	0.0	0.0	0.0	1.9	7	4
	200	5	96.3	6.6	0.8	1.2	1.0	0.4	0.5	106.5	75	5
		10	12.4	1.4	0.2	0.3	0.2	0.1	0.1	24.1	43	5
		20	2.6	0.6	0.1	0.1	0.1	0.0	0.0	7.5	26	6
c	100	5	1.6	0.3	0.1	0.2	0.1	0.1	0.1	22.4	11	3
		10	0.4	0.1	0.0	0.1	0.0	0.0	0.0	5.4	7	3
		20	0.1	0.1	0.0	0.0	0.0	0.0	0.0	2.0	5	3
	200	5	96.6	6.2	0.5	0.8	1.1	0.4	0.4	88.4	100	7
		10	11.5	1.4	0.2	0.3	0.2	0.1	0.1	25.4	37	5
		20	2.5	0.6	0.1	0.1	0.1	0.0	0.0	8.4	21	5
	400	10	932.3	44.5	1.6	3.8	1.4	0.5	0.9	196.3	381	18
		20	104.4	9.5	0.5	1.4	0.2	0.2	0.2	92.6	144	13
		40	22.2	3.7	0.2	0.4	0.1	0.1	0.1	11.6	62	11
	900	15	21,598.0	2,998.3	11.1	19.0	1.3	2.9	2.8	263.5	1,538	214
		30	7,507.9	336.1	3.9	6.3	1.0	0.5	0.7	136.3	1,587	71
		60	1,054.6	84.1	2.1	2.6	0.2	0.2	0.3	31.4	402	32
	1600	20	21,600.0	21,598.7	90.8	81.7	0.8	1.4	8.7	677.0	221	221
		40	21,600.0	7,752.2	25.5	20.5	0.4	1.4	2.0	244.1	786	282
		80	21,600.0	1,515.9	11.2	9.1	0.3	0.5	0.9	61.4	1,695	118
d	100	5	1.6	0.4	0.2	0.3	0.2	0.2	0.4	81.5	3	1
		10	0.4	0.2	0.1	0.1	0.1	0.1	0.1	31.4	2	1
		20	0.2	0.1	0.1	0.1	0.0	0.0	0.1	17.6	2	1
	200	5	85.1	8.4	1.9	1.5	2.8	1.3	3.1	251.1	18	2
		10	13.1	2.6	0.9	0.7	0.4	0.3	1.3	143.7	6	1
		20	3.5	1.2	0.5	0.5	0.1	0.1	0.4	63.7	4	1
	400	10	1,060.6	85.7	6.6	4.2	7.7	3.4	8.5	317.1	70	6
		20	160.6	25.5	3.0	2.4	1.2	0.8	3.2	293.3	26	4
		40	44.6	11.3	1.7	1.9	0.3	0.3	1.0	95.3	16	4
	900	15	21,594.0	5,043.5	33.9	33.5	5.7	25.9	36.4	1,033.3	307	72
		30	14,729.0	1,271.3	15.6	15.4	10.1	6.5	16.4	517.3	454	39
		60	3,370.0	500.4	7.4	6.4	2.1	1.8	4.9	137.9	265	39
	1600	20	21,600.0	21,594.5	233.7	191.4	2.6	7.2	106.3	1,569.1	64	64
		40	21,596.0	21,594.8	51.0	37.8	3.1	5.9	32.7	405.1	256	256
		80	21,599.0	11,758.9	26.7	19.3	0.8	5.9	20.8	268.3	441	238
e	100	5	1.4	0.3	0.1	0.2	0.1	0.1	0.1	43.4	9	3
		10	0.4	0.2	0.1	0.1	0.1	0.0	0.0	15.8	4	2
		20	0.2	0.2	0.1	0.1	0.0	0.0	0.0	12.3	3	2
	200	5	86.4	7.4	0.9	1.3	0.9	0.4	0.6	97.2	62	6
		10	11.9	2.6	0.3	0.6	0.2	0.1	0.3	34.6	20	4
		20	3.2	1.4	0.2	0.5	0.1	0.1	0.1	46.3	10	4
	400	10	878.4	70.9	3.3	4.5	1.8	0.9	1.6	155.6	181	15
		20	129.6	22.8	1.4	2.6	0.4	0.3	0.8	208.9	57	10
		40	30.1	11.4	0.8	1.7	0.1	0.1	0.2	159.3	24	9
	900	15	21,597.0	4,400.4	34.0	74.1	2.0	6.5	9.3	2,244.2	543	111
		30	10,069.0	986.8	10.0	16.6	2.6	1.9	3.7	1,146.5	713	70
		60	1,414.9	323.6	7.0	12.6	0.6	0.6	1.5	1,193.1	150	34
	1600	20	21,599.0	21,599.3	167.3	160.6	0.7	2.0	29.2	1,950.1	111	111
		40	21,599.0	21,017.8	63.7	70.5	0.6	8.4	12.3	3,067.6	279	272
		80	21,596.0	5,896.3	59.4	37.3	0.6	3.0	8.8	3,596.6	312	85

Table 9: Average of RMP time (columns 4–7) and Pricing time (columns 8–11) across classes, jobs, and machines. Additionally, total time (RMP + Pricing) comparison against LT (columns 12-13), clearly showing instances where Template pricing is over 1000× faster than Dantzig, and 100× faster than Pessoa. Five replications each, six hour time limit.