https://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING

Vol. 00, No. 0, Xxxxx 0000, pp. 000–000

Submitted to *INFORMS Journal on Computing*

ISSN 0899-1499, EISSN 1526-5528

Column Generation Using Domain-Independent Dynamic Programming

Ryo Kuroiwa
National Institute of Informatics / The Graduate University of Advanced Studies, SOKENDAI, Japan, kuroiwa@nii.ac.jp

Edward Lam
Monash University, Australia, edward.lam@monash.edu

Abstract. Column generation and branch-and-price are leading methods for large-scale exact optimization. Column generation iterates between solving a master problem and a pricing problem. The master problem is a linear program, which can be solved using a generic solver. The pricing problem is highly dependent on the application but is usually discrete. Due to the difficulty of discrete optimization, high-performance column generation often relies on a custom pricing algorithm built specifically to exploit the problem's structure. This bespoke nature of the pricing solver prevents the reuse of components for other applications. We show that domain-independent dynamic programming, a software package for modeling and solving arbitrary dynamic programs, can be used as a generic pricing solver. We develop basic implementations of branch-and-price with pricing by domain-independent dynamic programming and show that they outperform a world-leading solver on static mixed integer programming formulations for seven problem classes.

Funding: This work is supported by JSPS KAKENHI grant number JP25K24378 and by the Australian Research Council under grant DE240100042.

Key words: column generation, search, dynamic programming, domain-independent dynamic programming, generic column generation

1. Introduction

Mixed-integer programs (MIPs) with a large number of variables are computationally difficult to construct, let alone solve. Branch-and-price and column generation are two related methods for overcoming this difficulty. Instead of enumerating all variables in advance, column generation loops between solving a restricted master problem and a pricing problem (e.g., Lübbecke and Desrosiers 2005). The restricted master problem contains only a subset of the variables. This set is iteratively expanded by solving the pricing problem. At termination, the subset is still substantially smaller than the full set but is sufficient to prove optimality or infeasibility. To obtain integer solutions, column

generation is embedded in a branch-and-bound tree search, where the linear relaxation of each node is solved using column generation. This process is called branch-and-price.

The pricing problem is often application-specific. Although it can be modeled as a MIP and solved using a black-box software package such as Gurobi (Gurobi Optimization, LLC 2024) or CPLEX (IBM 2024), generic solvers are usually too slow for this purpose. High-performance branch-and-price codes rely on custom pricing solvers that exploit the problem's structure.

In many applications, the pricing problem reduces to a variant of the shortest path problem (Irnich and Desaulniers 2005), which can be solved effectively using dynamic programming (DP) (Bellman 1957). However, these techniques are typically tailored to problem-specific assumptions, limiting reuse and making generic column generation frameworks rare.

Domain-independent dynamic programming (DIDP) is a software package for modeling DP problems and offers a suite of generic search algorithms (Kuroiwa and Beck 2023a, 2025a). Its flexibility enables rapid prototyping of different pricing models and algorithms. This paper shows that straightforward branch-and-price implementations backed by DIDP pricing can outperform Gurobi, a state-of-the-art black-box MIP solver, on multiple problem classes.

The contributions of this paper are:

1. Four new features in DIDP for modeling and solving pricing problems, including a generic labeling algorithm.
2. Declarative DIDP models for pricing in seven problem classes and basic branch-and-price implementations built on them.
3. The first application of branch-and-price to the cumulative vehicle routing problem with time windows (CumVRPTW) (Fernández Gil et al. 2020, Corona-Gutiérrez et al. 2022).
4. Experimental results showing that these implementations can outperform Gurobi, a world-leading black-box MIP solver.

The remainder of this paper is structured as follows. Section 2 reviews column generation and DIDP. Section 3 surveys related work. Section 4 introduces the new features of DIDP. Section 5 reports experimental results. Section 6 concludes this paper. The formulations can be found in the appendix.

2. Background

This section describes column generation and the solving methodology in DIDP.

2.1. Column Generation

Many combinatorial optimization problems can be modeled as a MIP that selects a subset of combinatorial objects (e.g., routes, paths, schedules, cutting patterns, etc.) under compatibility constraints. Every object is represented by a variable, so the number of variables grows exponentially with the instance size, making full enumeration impractical. Branch-and-price and column generation address this issue by considering only a small subset of variables and proving that this subset is sufficient to guarantee optimality or infeasibility.

Let X be the set of all combinatorial objects, indexed $1, \dots, n$ where $n = |X|$. Define the *integer master problem* as:

$$\begin{aligned} & \min && c^\top \lambda \\ \text{(IMP)} \qquad & \text{s.t.} && A\lambda \geq b, \\ & && \lambda \in \mathbb{Z}^n_+, \end{aligned}$$

where $c \in \mathbb{Q}^n$, $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$. Its linear relaxation, called the *master problem*, is:

$$\begin{aligned} & \min && c^\top \lambda \\ \text{(MP)} \qquad & \text{s.t.} && A\lambda \geq b, \\ & && \lambda \in \mathbb{R}^n_+. \end{aligned}$$

Despite being a linear program, (MP) remains intractable because n is large. Instead, we solve a restricted problem with far fewer variables. For some $n' \ll n$, define the *restricted master problem* as:

$$\begin{aligned} & \min && c'^\top \lambda' \\ \text{(RMP)} \qquad & \text{s.t.} && A'\lambda' \geq b, \qquad (1) \\ & && \lambda' \in \mathbb{R}^{n'}_+, \end{aligned}$$

where $\lambda' = (\lambda_1, \dots, \lambda_{n'}) \in \mathbb{R}^{n'}_+$ is a subset of the variables and c', A' are the corresponding submatrices. Column generation solves (RMP), adds improving variables and repeats until no improving variables remain.

Let $\pi \in \mathbb{R}^m_+$ be the dual variables of Constraint (1). Given an optimal primal-dual solution $(\hat{\lambda}, \hat{\pi})$ to (RMP), the reduced cost of any variable λ_j, $j \in \{1, \dots, n\}$, is

$$\bar{c}_j = c_j - A^\top_{\cdot,j} \hat{\pi}.$$

At optimality, the variables present in (RMP) satisfy $\bar{c}_j \geq 0$. Any variable outside (RMP) (i.e., $j \in \{n'+1, \dots, n\}$) with $\bar{c}_j < 0$ can improve the objective when added.

Explicitly scanning $j = n' + 1, \ldots, n$ is impractical because c and A are too large to construct. Instead, we define an oracle, called the *pricing problem*, that searches a $j \in \{n' + 1, \ldots, n\}$ with $\bar{c}_j < 0$. Assume that every $x \in X$ has a vector representation $(x^1, \ldots, x^k) \in \mathbb{Z}^k$ subject to internal feasibility constraints $Dx \geq e$. Formally, $X = \{x \in \mathbb{Z}^k : Dx \geq e\}$. The pricing problem is:

$$\text{(PP)} \qquad \begin{aligned} \min_{j} \quad & \bar{c}_j \\ \text{s.t.} \quad & x_j \in X. \end{aligned}$$

If (PP) finds $\bar{c}_j < 0$, we add the corresponding variable to (RMP) and reoptimize. The process stops when (PP) proves there is no negative reduced cost column, which certifies that the current (RMP) solution is also optimal for (MP).

Column generation can only solve linear programs and must be embedded in a branch-and-bound tree search to solve MIPs. This embedding is called *branch-and-price*. In branch-and-price, branching rules must be designed so that the pricing problem respects the branching decisions. Branching rules are problem-specific, so a full review is beyond the scope of this paper.

While (PP) can be solved by problem-agnostic MIP solvers, specialized pricing solvers typically perform substantially better because they can exploit problem structure. In practice, X often defines a shortest path problem, which admits fast specialized algorithms. Building a competitive generic solver based on column generation for arbitrary MIP problems requires both the ability to recognize the problem structure X and the availability of a specialized algorithm to exploit this structure, making such a solver elusive. Rather than pursuing full generality, we provide modeling tools and a library of pre-built DP algorithms, enabling users to easily prototype different instantiations of branch-and-price. Promising results can then motivate bespoke pricing implementations.

2.2. Dynamic Programming

Many pricing problems are naturally solved by dynamic programming (DP). DP characterizes the problem via states and transitions, with costs or profits on transitions; an optimal policy solves the associated Bellman recursion.

As a running example, consider the shortest path problem with resource constraints (SPPRC) (Irnich and Desaulniers 2005), here instantiated as a VRPTW pricing problem with capacity and time-window resources and an elementary (no-revisit) constraint. Let $(\mathcal{N}, \mathcal{A})$ be a directed graph with nodes $\mathcal{N} = \{0, \ldots, n+1\}$ (source 0, sink $n+1$) and arcs $\mathcal{A} \subseteq \{\mathcal{N} \times \mathcal{N} : i \neq j, i < n+1, j > 0\}$. Each customer i has load $l_i \geq 0$, release time $a_i \geq 0$, due time $b_i \geq 0$, and service duration $s_i \geq 0$; each arc (i, j) has distance $d_{i,j} \geq 0$ and travel cost $c_{i,j}$. The goal is an elementary path from 0 to $n+1$ of

minimum total travel cost such that cumulative load never exceeds capacity Q and each visit respects $[a_i, b_i]$.

Let $V(\mathcal{R}, i, q, t)$ be the minimum cost from node i to $n+1$ when the unvisited set is $\mathcal{R} \subseteq \{1, \dots, n\}$, current load is q, and time is t. The Bellman equation is:

$$V(\mathcal{R}, i, q, t) = \begin{cases} 0 & \text{if } i = n+1 \\ \min_{j \in \mathcal{R} \cup \{n+1\} : (i,j) \in \mathcal{A} \wedge q + l_j \leq Q \wedge t + s_i + d_{i,j} \leq b_j} c_{i,j} + V\left(\mathcal{R} \setminus \{j\}, j, q + l_j, t'(j)\right) & \text{otherwise} \end{cases} \tag{2}$$

where $t'(j) = \max\left\{t + s_i + d_{i,j}, a_j\right\}$. The optimal objective value is $V(\{1, \cdots, n\}, 0, 0, 0)$.

2.3. Domain-Independent Dynamic Programming

Domain-independent dynamic programming (DIDP) is a generic solver framework for DP. Previous work developed Dynamic Programming Description Language (DyPDL) (Kuroiwa and Beck 2023a, 2025a), a declarative modeling formalism for DIDP. In DyPDL, a DP model is defined by *state variables*, *transitions*, *base cases*, and *state constraints*.

A state variable has a type, either numeric, element, or set. A numeric variable takes a value in $\mathbb{Q}$, an element variable in $\mathbb{Z}_0^+$, and a set variable in $2^{\mathbb{Z}_0^+}$. A state is represented by full value assignments to the state variables, and we denote the value of a state variable x in state S by $S[x]$. For our example in Equation (2), $\mathcal{R}$, j, q, and t can be modeled as state variables in DyPDL, and $\mathcal{R}$ is a set variable, j is an element variable, and q and t are numeric variables. An expression e is a function that returns a value $e(S)$ given a state S, built from predefined operations on state variables. In particular, a *numeric expression* returns a value in $\mathbb{Q}$, an *element expression* returns a value in $\mathbb{Z}_0^+$, a *set expression* returns a value in $2^{\mathbb{Z}_0^+}$, and a *condition* returns a Boolean value in $\langle \bot, \top \rangle$, where $\bot/\top$ represents that the condition is unsatisfied/satisfied. When a condition c is satisfied by state S, i.e., $c(S) = \top$, we denote it by $S \models c$.

A transition defines the change of a state by making a decision. For each state variable, an expression e with the corresponding type defines the updated value $e(S)$ after the transition is applied. In addition, each transition has preconditions, conditions that must be satisfied by a state for the transition to be applied. The transition is *applicable* in state S if for each precondition c, $S \models c$. In our example in Equation (2), visiting node j corresponds to a transition in DyPDL that updates $\mathcal{R}$ to $\mathcal{R} \setminus \{j\}$, i to j, q to $q + l_j$, and t to $t'(j)$. This transition has preconditions $j \in \mathcal{R} \cup \{n+1\}$, $(i, j) \in \mathcal{A}$, $q + l_j \leq Q$, and $t + s_i + d_{i,j} \leq b_j$.

A base case defines conditions that a state must satisfy for termination. In other words, no more transitions are applied when a state satisfies such conditions. A state satisfying a base case is called a *base state*. For our example in Equation (2), a base case is defined by a condition $i = n + 1$.

A state constraint defines conditions that must be satisfied by all states. For our example in Equation (2), we do not have particular state constraints.

In addition to the above components, a special state called the *target state* is defined in DyPDL. A solution for a DyPDL model is a sequence of transitions that transforms the target state into a base state. We give a more formal definition in what follows. Let the target state be S^0 and $\mathcal{T}(S)$ be a set of applicable transitions in a state S. For a transition $\tau \in \mathcal{T}(S)$, let $S[[\tau]]$ be a state where the value of each state variable is updated from S according to τ. A *solution* is a sequence of transitions $\langle \tau_1, ..., \tau_n \rangle$ such that $\tau_i \in \mathcal{T}\left(S^{i-1}\right)$ and $S^i = S^{i-1}[[\tau_i]]$ for $i = 1, \ldots, n$, S^i satisfies all state constraints for $i = 0, \ldots, n$, and S^n satisfies a base case. Analogously, we define an *S-solution*, a sequence of applicable transitions that transforms a state S into a base state.

For simplicity, we focus on a subset of the DyPDL formalism, where the objective value of a solution is defined by the weight function w_τ of a state associated with each transition τ. In addition, we consider the weight function v, which maps a base state S to its objective value $v(S)$. Given a solution $\langle \tau_1, ..., \tau_n \rangle$ with $S^i = S^{i-1}[[\tau_i]]$ for $i = 1, \ldots, n$, its objective value is $\sum_{i=1}^{n} w_{\tau_i}\left(S^{i-1}\right) + v\left(S^n\right)$. The objective value of an S-solution is defined analogously. An optimal solution minimizes the objective value. We note that our approach can be easily extended to maximization and the case where the weights are combined by binary operators such as multiplication, min, and max by following previous work (Kuroiwa and Beck 2025a). The optimal objective value can be represented by the following Bellman equation:

$$\text{(DIDP)} \qquad \begin{aligned} &\text{compute } V(S^0) \\ &V(S) = \begin{cases} \infty & \text{if } S \text{ violates a state constraint} \\ v(S) & \text{if } S \text{ is a base state} \\ \min_{\tau \in \mathcal{T}(S)} w_\tau(S) + V(S[[\tau]]) & \text{otherwise.} \end{cases} \end{aligned}$$

The first line declares that the optimal objective value for the problem is $V(S^0)$, the value of the target state. The first case of the equation defines $V(S) = \infty$ if any state constraint is violated. The second case defines the value of a base state. The third case recursively defines the optimal objective value for an S-solution using transitions. Here, we assume that the third case equals ∞ if $\mathcal{T}(S) = \emptyset$.

In DyPDL, redundant information implied by other parts of the DP model can be explicitly defined. Such information is analogous to valid inequalities in a MIP model and can potentially be useful for a solver. DyPDL provides two specific features solely for redundant information: *resource variables* and *dual bound functions*.

In problem-specific DP algorithms, *state dominance* is sometimes exploited, where one state is known to be superior to another. For our example in Equation (2), a state $(\mathcal{R}, i, q_1, t_1)$ leads to a better or equal solution than a state $(\mathcal{R}, i, q_2, t_2)$ if $q_1 \leq q_2$ and $t_1 \leq t_2$. In DyPDL, to represent state dominance, a numeric variable or an element variable can be declared as a resource variable with a preference for less or greater. Given two states S and S', S is preferred over S' if $S[r] \leq S'[r]$ for each resource variable r that prefers less, $S[r] \geq S'[r]$ for each resource variable r that prefers greater, and $S[x] = S'[x]$ for each non-resource variable x. When S is preferred to S', a solver assumes that for each S'-solution, there exists an S-solution that has an equal or better objective value with an equal or shorter number of transitions. Using the value function for minimization,

$$V(S) \leq V(S') \text{ if } S \text{ is preferred to } S'.$$

A dual bound function η returns a lower bound $\eta(S)$ on the optimal value of a state S, i.e.,

$$V(S) \geq \eta(S).$$

In DyPDL, a dual bound function is described by an expression, similar to other components. For our example in Equation (2), since the travel cost of an arc can be negative, we can use a dual bound function that only considers the negative incoming arc for each node. Using the minimum incoming arc cost $c_j^{\text{in}} = \min_{(k,j)\in\mathcal{A}} c_{kj}$ for node j,

$$V(\mathcal{R}, i, q, l) \geq \sum_{j \in \mathcal{R} \cup \{n+1\}} \min\left\{c_j^{\text{in}}, 0\right\}. \tag{3}$$

2.4. State-Space Search Algorithms in Artificial Intelligence

In the field of artificial intelligence (AI), a range of state-space search algorithms has been developed for solving problems such as planning and combinatorial optimization problems (Russell and Norvig 2020), often in parallel and with little communication with the mathematical optimization community.

State-space search algorithms are a class of recursive algorithms for exploring a state transition graph in which vertices represent subproblems (states) and edges represent decisions (transitions).

Conceptually, state-space search algorithms recursively explore the state space until finding a goal state, in which case the path to the state is a solution. This recursion naturally makes them suited for solving DP problems. Two successful state-space search algorithms implemented in DIDP are cost-algebraic A* solver for DyPDL (CAASDy) and complete anytime beam search (CABS) (Kuroiwa and Beck 2023a,b, 2025a).

CAASDy is based on A* (Hart et al. 1968), a highly successful generalization of Dijkstra's shortest path algorithm (Dijkstra 1959). In A*, each state S is assigned a cost $g(S)$, representing the cost to reach S from an initial state S^0. The A* algorithm requires the definition of a heuristic function $h(S)$ that estimates the cost to-go to reach any base state (i.e., a feasible solution) from state S. The total estimated cost of a state S is then $f(S) = g(S) + h(S)$, comprising the cost-so-far $g(S)$ and the cost to-go $h(S)$. A* maintains a priority queue of states ordered by $f(S)$, called the open list, and expands states in order of increasing f-value. If the heuristic function $h(S)$ is *admissible* (i.e., never overestimates the true cost) and all state transitions have non-negative cost, A* is both complete and optimal, guaranteeing that the first solution found is a least-cost path. The effectiveness of A* depends on the quality of the heuristic, which guides the search toward promising regions of the state space and can dramatically reduce the number of states explored. Dijkstra's algorithm is a special case of A* when all state transitions have non-negative cost and $h(S) = 0$ for all states S.

CAASDy uses the dual bound function η as an admissible heuristic function h as it underestimates the optimal path cost. In addition, it uses state dominance defined by resource variables for pruning states. CAASDy also allows more general cost structures that can be represented in DyPDL, based on the cost-algebraic heuristic search framework (Edelkamp et al. 2005). For example, maximization is supported in addition to minimization, cost functions can be combined using operators other than addition, such as multiplication, min, and max, and negative transition costs are allowed.

CABS (Zhang 1998) is based on beam search, an incomplete breadth-first search algorithm that explores only a few of the most promising states at each depth, called the beam. At every iteration, it generates the successor states of all states currently in the priority queue. It then inserts only a subset of the successors into the priority queue for exploration in the next iteration and discards others. In the implementation by Kuroiwa and Beck (2023b, 2025a), CABS select the k states with lowest f-values, and the parameter k is called the beam width. Because it discards states, beam search is incomplete.

CABS guarantees completeness and provides solutions of increasing quality over time by repeatedly running beam search in the inner loop while increasing the beam width in the outer loop

until search space is exhausted. Note that whenever the beam width is increased, the beam search in the inner loop will repeat states explored in the previous iteration. CABS is anytime (it can return the best solution found so far if requested to terminate) and it is complete (it will eventually find an optimal solution given sufficient time). Similar to CAASDy, the CABS solver in DIDP also uses the dual bound function as a heuristic function and state dominance for pruning.

3. Literature Review

Achieving high-performance column generation requires identifying exploitable structure in the pricing problem and implementing a matching specialized algorithm to exploit this structure. This difficult task is the reason that generic column generation solvers remain uncommon. Nevertheless, there are a few attempts at automatic column generation.

Dantzig-Wolfe decomposition is a method for reformulating a *compact* model (often polynomial number of variables in the instance size) into an *extended* model with many more variables (often exponential) (Lübbecke and Desrosiers 2005). The reformulation attains a dual bound no weaker than the original and sometimes significantly stronger (e.g., Letchford and Salazar-González 2006), resulting in much faster solve times despite being significantly larger. GCG is an open-source academic solver that analyzes a given MIP model to obtain a Dantzig-Wolfe reformulation and then solves both the reformulation and the original model side-by-side (Gamrath and Lübbecke 2010). The reformulation may provide a stronger dual bound but the original model is easier for defining cutting planes, branching rules, etc. because these additions do not affect the pricing problem (i.e., they are *robust* (de Aragao and Uchoa 2003, Fukasawa et al. 2006)). Additionally, GCG can take the matrix structure as input, which assists in choosing a subset of variables and constraints for the Dantzig-Wolfe reformulation.

While GCG implements custom pricing solvers in private development versions, they are not publicly available, presumably because they require sophisticated detectors for analyzing the structure of the matrix to determine whether it contains blocks representing structured subproblems for which it has a specialized solver. Therefore, GCG can be considered to solve both the integer master problem and the pricing problem using SCIP, an academic MIP solver (Achterberg et al. 2008). Without the use of bespoke pricing solvers, GCG often performs poorly. Nevertheless, it serves as an important proof-of-concept showing that automatic Dantzig-Wolfe reformulation is theoretically and technically possible.

VRPSolver is a non-commercial and proprietary branch-and-price code for solving limited variations of vehicle routing problems (Pessoa et al. 2020). It contains specialized pricers tailored

to the resource-constrained shortest path pricing problems in vehicle routing. Users can model vehicle routing problems within the limitations of its library. However, details are limited because the code is closed-source. In any case, VRPSolver can only solve vehicle routing problems and related problems such as bin packing.

The field thus far lacks a generic but performant solver that fully automates Dantzig-Wolfe reformulation and column generation. This paper does not attempt to address this issue, but rather, makes it easier for researchers to manually prototype different pricing problems that arise from different Dantzig-Wolfe reformulations and solve them easily using a library of pre-built search algorithms. Should experimental results show that a basic branch-and-price solver based on a black-box dynamic programming pricer is competitive with static MIP models, then that evidence can justify developing a bespoke pricing algorithm.

4. Updates to DIDP

This section introduces four new features of DIDP for modeling and solving search problems commonly seen in the pricing problem of column generation.

4.1. Filter Operation

For our example in Equation (2), a state is represented by a set of unvisited nodes $\mathcal{R}$, the current node i, the current load q, and the current time t. While we update $\mathcal{R}$ to $\mathcal{R} \setminus \{j\}$ when j is visited, we can also remove a node $k \in \mathcal{R}$ that can no longer be visited by its due time b_k from $\mathcal{R}$. Let $d^*_{j,k}$ be the shortest travel time from node j to node k, which can be precomputed. Since we arrive at j at time $t'(j) = \max\left\{t + s_i + d_{i,j}\right\}$, if $t'(j) + s_j + d^*_{j,k} > b_k$, then node k cannot be visited after visiting j from the current state. In addition, k cannot be visited after j if it results in overload, i.e., $q + l_j + l_k > Q$. Thus, $\mathcal{R}$ is updated to $\mathcal{R}'(j) = \left\{k \in \mathcal{R} \setminus \{j\} : t'(j) + s_j + d^*_{j,k} \leq b_k \wedge q + l_j + l_k \leq Q\right\}$.

In the current DyPDL, the change of a state variable by a transition is described by expressions built from predefined operations on state variables. Existing solvers maintain expression tree data structures and evaluate them during solving. For set expressions, set operations such as union, intersection, and difference are implemented. In addition, an 'if-then-else' operation is available, which evaluates to one of two expressions depending on the evaluation result of a condition.)sing these operations, $\mathcal{R}'(j)$ can be implemented by repeatedly removing a singleton or empty set defined by a set expression 'if $k \in \mathcal{R} \wedge (t'(j) + s_j + d^*_{j,k} > b_k \vee q + l_j + l_k > Q)$ then $\{k\}$ else $\emptyset$' for each $k = 1, \ldots, n$. However, such an implementation complicates the model code. Furthermore, it results in an expression tree whose depth is proportional to n, which is slow to evaluate in practice.

For ease of modeling and efficiency, we introduce a *filter operation*, a set expression that returns a subset of a given set whose elements satisfy a specified condition. With our interface, a user specifies a filter operation by two components: a set expression $\mathcal{X}$ and a parameterized condition $c(x)$, a function that returns a condition given a parameter x. The parameter x is a placeholder and is replaced with each element of a set $\mathcal{X}(S)$ when evaluated, given a state S, and an element $i \in \mathcal{X}(S)$ is removed if $S \not\models c(i)$. In other words, the filter operation represents an expression that returns $\{x \in \mathcal{X}(S) : S \models c(x)\}$ given a state S. For our example, $\mathcal{R}'(j) = \{\mathcal{R} \setminus \{j\} : t'(j) + s_j + d^*_{j,k} \leq b_k \wedge q + l_j + l_k \leq Q\}$ can be represented by a filter operation defined by a set expression $\mathcal{R} \setminus \{j\}$ and a parameterized condition $t'(j) + s_j + d^*_{j,k} \leq b_k \wedge q + l_j + l_k \leq Q$, where k is the parameter.

4.2. Set Resource Variables

In the current DyPDL, only numeric and element variables can be resource variables to define state dominance. However, in pricing problems, state dominance is sometimes defined by a set variable. For our example in Equation (2), we could define state dominance where state $(\mathcal{R}_1, i, q_1, t_1)$ is better than or as good as $(\mathcal{R}_2, i, q_2, t_2)$ if $\mathcal{R}_2 \subseteq \mathcal{R}_1$, $q_1 \leq q_2$, and $t_1 \leq t_2$ since having more candidate nodes to visit potentially leads to a shorter path.

We introduce *set resource variables*: a state S is preferred to another state S' only if the value of a set variable in S is a subset or superset of that in S'. Similarly to numeric and element resource variables, the preference, less or greater, specifies whether a subset or superset is better. When less/greater is specified for a set resource variable $\mathcal{X}$, S is preferred to S' only if $S[\mathcal{X}] \subseteq S'[\mathcal{X}]/S'[\mathcal{X}] \subseteq S[\mathcal{X}]$.

A set resource variable can be mimicked by defining a set of numeric or element resource variables, whose values take either 0 or 1. However, our set resource variable implementation uses a bitset to represent a set, which is computationally more efficient.

4.3. Fractional Knapsack Expression

For our example in Equation (2), we presented a dual bound function considering the minimum incoming arc cost $c^{\text{in}}_j = \min_{(k,j)\in\mathcal{A}} c_{k,j}$ for each node j in Example 3. We can also take the current load q and the capacity Q into consideration when computing a dual bound. By visiting node j, we increase the load by l_j and the cost by at least c^{in}_j. Given a state $(\mathcal{R}, i, q, t)$,

$$V(\mathcal{R}, i, q, l) \geq \min_{\mathcal{J}\subseteq\mathcal{R}: q+\sum_{j\in\mathcal{J}} l_j \leq Q} \sum_{j\in\mathcal{J}} c^{\text{in}}_j. \tag{4}$$

This dual bound can be viewed as the negation of the optimal cost of the 0-1 knapsack problem, which is to maximize the total profit of items packed into a knapsack with a fixed capacity. In

particular, the knapsack has the capacity $Q - q$, and each node $j \in \mathcal{R}$ with $c_j^{\text{in}} < 0$ corresponds to an item with the profit $-c_j^{\text{in}}$ and weight l_j. We argue that a similar substructure is common in pricing problems when a subset of elements with the negative reduced costs needs to be selected under a resource constraint.

Since the 0-1 knapsack problem is NP-hard (Karp 1972), computing the right-hand side of Inequality (4) is also NP-hard. Recent work has reported that the Dantzig bound (Dantzig 1957), a polynomial-time upper bound on the optimal objective value for the 0-1 knapsack problem, is useful as the dual bound function for DIDP (Kuroiwa and Beck 2025b). Given the capacity C and a set of items $\mathcal{N}$ with weight $w_j > 0$ and the profit $p_j > 0$ for each $j \in \mathcal{N}$, the Dantzig bound can be computed as follows. First, the items are sorted in a descending order of $\frac{p_j}{w_j}$. Second, the items are included in the knapsack in sorted order as long as the total weight does not exceed the capacity C, and let $\mathcal{I}$ be the set of such items. When the current item j has the weight w_j larger than $C - \sum_{i \in \mathcal{I}} w_i$, it is fractionally included with the profit $\frac{p_j}{w_j}\left(C - \sum_{i \in \mathcal{I}} w_i\right)$. In other words, the optimal objective value is upper bounded by $\frac{p_j}{w_j}\left(C - \sum_{i \in \mathcal{I}} w_i\right) + \sum_{i \in \mathcal{I}} p_i$.

With expressions in the current DyPDL, efficiently modeling the Dantzig bound is difficult due to its algorithmic nature. Therefore, we introduce a new expression, called the *fractional knapsack expression*, dedicated to the Dantzig bound. We denote it by $\mathsf{fractional\_knapsack}\left(\mathcal{X}, C, (p_j)_{j=1,\dots,n}, (w_j)_{j=1,\dots,n}\right)$, where $\mathcal{X}$ is a set expression, C is a numeric expression, and $(p_j)_{j=1,\dots,n}$ and $(w_j)_{j=1,\dots,n}$ are lists of n numeric expressions. Then, the expression represents the Dantzig bound for the 0-1 knapsack problem, where given a state S, the set of items is $\mathcal{X}(S)$, the capacity of the knapsack is $C(S)$, and each item $x \in \mathcal{X}(S)$ has the profit p_x and the weight w_x. For our example, we represent the dual bound function as follows:

$$V(\mathcal{R}, i, q, l) \geq -\mathsf{fractional\_knapsack}\left(\mathcal{R}, Q - q, \left(\min\left\{-c_j^{\text{in}}, 0\right\}\right)_{j=1,\dots,n}, \left(l_j\right)_{j=1,\dots,n}\right). \tag{5}$$

4.4. Generic Labeling Solver

Labeling algorithms are commonly used for solving pricing problems such as SPPRC (Irnich and Desaulniers 2005, Pugliese and Guerriero 2013). In such an algorithm, for each node i in a graph, cumulative resource consumption by a path from the source node to i is represented as a label. A single node i can have multiple labels when there are multiple paths from the source node to i with different resource consumptions. Therefore, a labeling algorithm maintains a set of labels for each node.

In our example pricing problem for VRPTW, we consider SPPRC in a graph $(\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. A label is a 4-tuple $(\mathcal{R}, q, t, g)$, where $\mathcal{R}$ is the set of reachable nodes, q is the cumulative load, t is the time spent so far, and g is the path cost. Initially, the source node 0 has a label $(\mathcal{N} \setminus \{0, n+1\}, 0, 0, 0)$ corresponding to an empty path. When a node i has a label $(\mathcal{R}, q, t, g)$, for each node $j \in \mathcal{R} \cup \{n+1\}$ with $(i, j) \in \mathcal{A}$, $q + l_j \leq Q$, and $t + s_i + d_{ij} \leq b_j$, we can generate a new label $(\mathcal{R}'(j), q + l_j, \max\{t + s_i + d_{ij}, a_j\}, g + c_{ij})$, corresponding to extending the path. A labeling algorithm repeatedly generates labels to find a resource-feasible shortest path from the source node to the sink node. The order in which nodes and labels are selected for treatment depends on concrete algorithms. To reduce computational effort, a labeling algorithm typically prunes labels based on dominance; given two labels for the same node, one can be removed if another is known to be better or equal. In our example, a label $(\mathcal{R}_1, q_1, t_1, g_1)$ dominates another label $(\mathcal{R}_2, q_2, t_2, g_2)$ if $\mathcal{R}_2 \subseteq \mathcal{R}_1$, $q_1 \leq q_2$, $t_1 \leq t_2$, and $g_1 \leq g_2$, and thus the algorithm may discard the latter without loss of optimality.

A labeling algorithm is similar to the AI-style state-space search, already employed in DIDP. State dominance defined by resource variables in DIDP is analogous to dominance between labels and is already exploited by the existing solvers, such as CAASDy and CABS. A state transition can be viewed as generating a new label from an existing label. The practical difference between labeling algorithms and the existing DIDP solvers is in the order in which a label (or a state in DIDP) is selected. If an algorithm detects that label l is dominated by another label l' after treating l, it has done useless work since l could have been discarded without treatment. To reduce such useless work, labeling algorithms typically prioritize labels with better resource consumption by using a lexicographic order (Pugliese and Guerriero 2013). Under some conditions, such approaches have a theoretical guarantee that a treated label will not be discarded later and are described as *label setting*. In contrast, CAASDy and CABS select states based on the f-values and do not consider resource variables. Therefore, in this paper, we propose a generic labeling solver for DIDP, which searches states in a lexicographic order of resource variables. We note that our algorithm is not guaranteed to be a label setting algorithm in general.

Our solver is built on top of the anytime heuristic search framework of DIDP proposed in previous work (Kuroiwa and Beck 2023b, 2025a). In that framework, states to be searched are maintained in a priority queue called an open list. In each iteration, one state is selected and removed from the open list, and its successor states are generated by applying transitions and then inserted into the

Algorithm 1 Generic labeling solver for a DyPDL model. The target state is denoted by S^0 and the dual bound function by η.

1: **if** $S^0 \not\models C$ **then return** $\emptyset$ ▷ Check the state constraints.
2: $\Sigma \leftarrow \emptyset, \overline{\gamma} \leftarrow \infty$ ▷ Initialize solutions.
3: $\sigma(S^0) \leftarrow \langle\rangle$, $g(S^0) \leftarrow 0$ ▷ Initialize the g-value.
4: $O \leftarrow \{S^0\}, G \leftarrow \{S^0\}$ ▷ Initialize the open list.
5: **while** $O \neq \emptyset$ **do**
6: Let $S \in O$ be the lexicographically minimum state
7: $O \leftarrow O \setminus \{S\}$ ▷ Remove the state.
8: **if** S is a base state and $g(S) + v(S) < \overline{\gamma}$ **then**
9: $\overline{\gamma} \leftarrow g(S) + v(S)$ ▷ Update the best solution cost.
10: $O \leftarrow \{S' \in O : g(S') + \eta(S') < \overline{\gamma}\}$ ▷ Prune states in the open list.
11: $\Sigma \leftarrow \Sigma \cup \{\sigma(S)\}$ ▷ Add the new best solution.
12: **else**
13: **for all** $\tau \in \mathcal{T}(S) : S[[\tau]]$ satisfies all state constraints **do**
14: $g_{\text{current}} \leftarrow g(S) + w_\tau(S)$ ▷ Compute the g-value.
15: **if** $\nexists S' \in G$ such that $S[[\tau]]$ is preferred to S' and $g_{\text{current}} \geq g(S')$ **then**
16: $G \leftarrow \{S' \in G : S \text{ is not preferred to } S' \vee g_{\text{current}}(S) < g(S')\}$
17: **if** $g_{\text{current}} + \eta(S[[\tau]]) < \overline{\gamma}$ **then**
18: $\sigma(S[[\tau]]) \leftarrow \langle\sigma(S); \tau\rangle$, $g(S[[\tau]]) \leftarrow g_{\text{current}}$
19: $G \leftarrow G \cup \{S[[\tau]]\}, O \leftarrow O \cup \{S[[\tau]]\}$ ▷ Insert the successor state.
20: **return** Σ ▷ Return solutions.

open list if they are not dominated by existing states. Each concrete algorithm differs in selecting the state to remove from the open list.

We present pseudocode for the generic labeling algorithm for a DyPDL model in Algorithm 1. Except for line 6, the algorithm and implementation details follow the existing solvers. To emphasize that the algorithm returns multiple solutions, we write Σ to explicitly denote a set of solutions found. The set Σ is initialized as an empty set (line 2). When the target state violates state constraints, we immediately return an empty set and terminate (line 1). We maintain the current best solution cost $\overline{\gamma}$, initialized with ∞ (line 2). For each state S, we record the best sequence of transitions to reach it, $\sigma(S)$, and the g-value $g(S)$, corresponding to the accumulated transition weight. Given

$\sigma(S) = \langle \tau_1, \ldots, \tau_m \rangle$, we have $g(S) = \sum_{i=1}^{m} w_{\sigma_i}(S^{i-1})$ where $S^i = S^{i-1}[[\tau_i]]$ for $i = 1, \ldots, m$. For the target state S^0, the path is empty, and the g-value is 0. The set G stores all generated states, and the open list O stores states to be searched, both of which initially contain only the target state (line 4). The algorithm proves optimality (or infeasibility) when the open list becomes empty (line 5) and returns the set of solutions found.

In each step, the lexicographically minimum state S is removed from the open list (lines 6 and 7). States are lexicographically ordered based on the values of resource variables. Given resource variables $r_1, \ldots, r_{n'}$, a state S is lexicographically smaller than S' if there exists $1 \leq i \leq n'$ such that $S[r_j] = S'[r_j]$ for $1 \leq j < i$, $S[r_i] \neq S'[r_i]$, and $S[r_i]$ is preferred to $S'[r_i]$. In our implementation, we compare element resource variables, numeric resource variables, and set resource variables in order. Resource variables of the same type are compared in order of definition. When all resource variables have the same values, we break ties by the g-value, and then the dual bound value, where smaller is preferred.

If S is a base state, then $\sigma(S)$ is a solution, and the best solution cost $\overline{\gamma}$ is updated if $\sigma(S)$ is better (lines 8–9). In addition, all states $S' \in O$ with $g(S') + \eta(S') \geq \overline{\gamma}$ are removed from the open list since they cannot lead to a better solution (line 10). Since $\eta(S)$ is a lower bound on the solution cost starting from S, $g(S) + \eta(S)$ is a lower bound on the solution cost extending the sequence of transitions $\sigma(S)$. If this value is equal to or worse than the current solution cost, the current sequence does not lead to a better solution, so we ignore it.

If S is not a base state, its successor state $S[[\tau]]$ is generated for every applicable transition in $\tau \in \mathcal{T}(S)$ if it satisfies the state constraints (line 13). If $S[[\tau]]$ is dominated by another state S' in G with a better or equal g-value, it cannot lead to a solution better than S', so $S[[\tau]]$ is ignored (line 15). Otherwise, states dominated by $S[[\tau]]$ with a better or equal g-value are removed from G (line 16). For this dominance detection procedure, G is implemented as a hash table, where keys are the values of the non-resource variables, and entries are arrays of pointers to states. When a successor state is generated, an array of states with the same non-resource variable values is retrieved from the hash table. The successor state is compared against each state in the array to detect dominance and appended to the array if not dominated.

After dominance detection, the dual bound value $\eta(S[[\tau]])$ is computed. If $g(S) + w_\tau(S) + \eta(S[[\tau]])$ is worse than the best solution cost, the successor state $S[[\tau]]$ is ignored (line 17). Otherwise, $\sigma(S[[\tau]])$ and $g(S[[\tau]])$ are initialized or updated, and $S[[\tau]]$ is inserted into the open list and G (line 19). Here, by $\langle \sigma(S); \tau \rangle$, we represent a sequence of transitions, which is an extension of $\sigma(S)$ with τ.

4.5. Summary of the New Features

In summary, we add the following new features:

- The filtering operation to efficiently construct a subset of elements satisfying a given condition.
- Set resource variables for dominance pruning.
- The fractional knapsack expression to efficiently compute an informative dual bound.
- A generic labeling solver considering resource variables in search order.

The filtering operation potentially reduces the size of the state space by removing unnecessary elements from set state variables. Together with the filtering operation, a set resource variable enables the solving algorithm to detect more state dominance. The fractional knapsack expression can provide an informative dual bound. These two features are useful for the generic labeling solver (and other solvers) to prune unnecessary states, as shown in Algorithm 1. Furthermore, the generic labeling solver tries to avoid expanding states dominated by other states generated later. By combining the new modeling features and the new solver, we generalize labeling algorithms used in problem-specific settings to DIDP.

5. Experimental Results

This section describes the computational experiments. These experiments compare the performance of other solvers against branch-and-price where pricing is performed using DIDP.

5.1. Problems and Instances

The solvers are evaluated on the following NP-hard problems. The models are provided in the appendix.

- **Bin packing problem:** The bin packing problem (BPP) considers a number of identical bins with a common capacity and a set of items, each associated with a weight. The aim is to place every item into a bin such that the capacity of the bin is not exceeded and the number of bins used is minimized. The pricing problem takes the form of the 0-1 knapsack problem that decides whether an item is included or excluded in a bin. Instances for the bin packing problem are retrieved from BPPLIB (Delorme et al. 2018). BPPLIB is a collection of instance sets gathered from several sources. The experiments are conducted on the Falkenauer (1996) set, the first of many instance sets within BPPLIB. We use the compact formulation described in Delorme et al. (2016).
- **Graph coloring problem:** Given a graph, the graph coloring problem (GCP) attempts to assign a color to every vertex such that no two adjacent vertices share the same color. The objective is to minimize the number of colors used. The pricing problem is the maximum weighted

independent set problem, where the weight of each node is the dual value. The solvers are tested on the instances collected by Michael Trick (`https://mat.tepper.cmu.edu/COLOR/instances.html`) and the Roars Lab (`https://github.com/dynaroars/npbench/tree/master/instances/coloring/graph_color`) at George Mason University. We use the compact formulation described in Malaguti and Toth (2010).

- **Parallel machine scheduling problem:** In parallel machine scheduling, a set of jobs is scheduled on multiple machines in parallel. In particular, we consider minimizing the total weighted completion time with identical machines, commonly denoted as $P||\sum w_i C_i$ (Eastman et al. 1964). In this problem, n jobs are scheduled on m identical machines, where each job j has processing time p_j and weight w_j. With the total weighted completion time objective, once a set of jobs is assigned to a machine, it is known that scheduling job j before job k results in a better or equal objective value if $w_j/p_j \leq w_k/p_k$ (Elmaghraby and Park 1974). Thus, the pricing problem is a variant of the 0-1 knapsack problem that selects jobs to schedule on a machine. Branch-and-price is compared against a compact formulation (presented in the appendix) on instances generated by us following previous work (van den Akker et al. 1999). In particular, we use $n = 20, 30, 40, 50$ and $m = 3, 4, 5$ with three different configurations for p_j and w_j: p_j uniformly sampled from $[1, 10]$ and w_j uniformly sampled from $[10, 100]$, p_j and w_j uniformly sampled from $[1, 100]$, and p_j and w_j uniformly sampled from $[10, 20]$. For each of the 36 configurations, we generate five instances, resulting in 180 instances in total.

- **Multi-runway aircraft scheduling problem:** The multi-runway aircraft scheduling problem (MRASP), proposed by Ghoniem et al. (2015), is a variant of parallel machine scheduling. It aims to schedule the landing and take-off operations of a set of aircraft while minimizing the weighted sum of the landing times of the aircraft. These operations must be separated by a minimum duration, some of which violate the triangle inequality. The pricing problem is an SPPRC with two additional resources for tracking aircraft operations whose time violates the triangle inequality, and the arc cost depends on the current time. The instances are published by Ghoniem et al. (2015). These instances are randomly generated but some data are derived from regulations specified by the Federal Aviation Administration of the United States of America. The compact formulation is also from Ghoniem et al. (2015).

- **Vehicle routing problem with time windows:** The vehicle routing problem with time windows (VRPTW) (e.g., Vigo and Toth 2014) considers an infinite number of identical vehicles initially stationed at a depot and a set of customers. Every customer is associated with a location distinct

from the depot, a load, and a time window within which the customer must be visited by a vehicle. The problem seeks to determine a sequence of customer visits for each vehicle while respecting the capacity of each vehicle. The objective is to minimize the total travel distance of all vehicles visiting their assigned customers and returning to the depot. The pricing problem is the SPPRC used in our running example. Both the elementary and non-elementary versions are tested. The elementary version restricts every customer to be visited at most once along a path. The non-elementary version is a relaxed problem, where visiting the same node multiple times is allowed, and can be used without loss of optimality (Desrochers et al. 1992). Branch-and-price is compared against two-index and three-index compact models (Vigo and Toth 2014). The experiments are conducted on the well-known Solomon (1987) instances with 50 and 100 customers.

- **Cumulative vehicle routing problem with time windows:** The cumulative vehicle routing problem with time windows (CumVRPTW) modifies the objective of the VRPTW such that the travel cost is multiplied by the cumulative load of the vehicle (Fernández Gil et al. 2020, Corona-Gutiérrez et al. 2022). We also introduce a limit on the number of vehicles used. The pricing problem is the same as the VRPTW but the objective function is modified with the cumulative cost. As far as we know, column generation has not been applied to CumVRPTW. Branch-and-price is compared against two-index and three-index models. The experiments are run on the Solomon instances for the VRPTW.
- **Pickup and delivery problem with time windows:** The pickup and delivery problem with time windows (PDPTW) makes two modifications to the VRPTW. Firstly, the objective function is hierarchical: first minimize the number of vehicles in use and then minimize the total travel distance. Secondly, every customer is associated with a pickup task and a delivery task, specifying a precedence relation. The pickup task and delivery task individually have time windows. The pricing problem is the same as the VRPTW but includes resources to track whether a pickup is on-board and hence the corresponding delivery must be completed. Branch-and-price is compared against two-index (Furtado et al. 2017) and three-index models (Ropke and Cordeau 2009). The 100-case instances from the Li and Lim (2001) benchmarks are used.

5.2. Solvers

We add the new features for DIDP to didp-rs v0.9.0, a software implementation of DIDP. Since didp-rs is written in Rust, we implement the new features in Rust (`https://github.com/domain-independent-dp/didp-rs/releases/tag/labeling`). However, we implement branch-and-price algorithms in Python (`https://github.com/Kurorororo/`

`didp-column-generation`), using PySCIPOpt, the Python interface for SCIP, and DIDPPy, the Python interface for didp-rs. This choice of programming language conveys our goal of quick prototyping and ease of modeling, rather than high performance.

Three search algorithms within DIDP are compared: the new labeling algorithm, CABS and CAASDy. All share a common base for the master problem, the branching rules and the pricing model. They differ only in the choice of solving algorithm in pricing. The branching rules are unique for each problem class, and we describe them in the appendix.

The branch-and-price approaches are compared against solving a compact formulation using SCIP 9.2.1 with PySCIPOpt, GCG 3.5.5 with its Python interface PyGCGOpt, and Gurobi 12.0.2 with its Python interface gurobipy. VRPSolverEasy (Errami et al. 2024), an open-source Python interface for VRPSolver, is also run on the VRPTW. To our knowledge, VRPSolverEasy is not compatible with CumVRPTW and PDPTW.

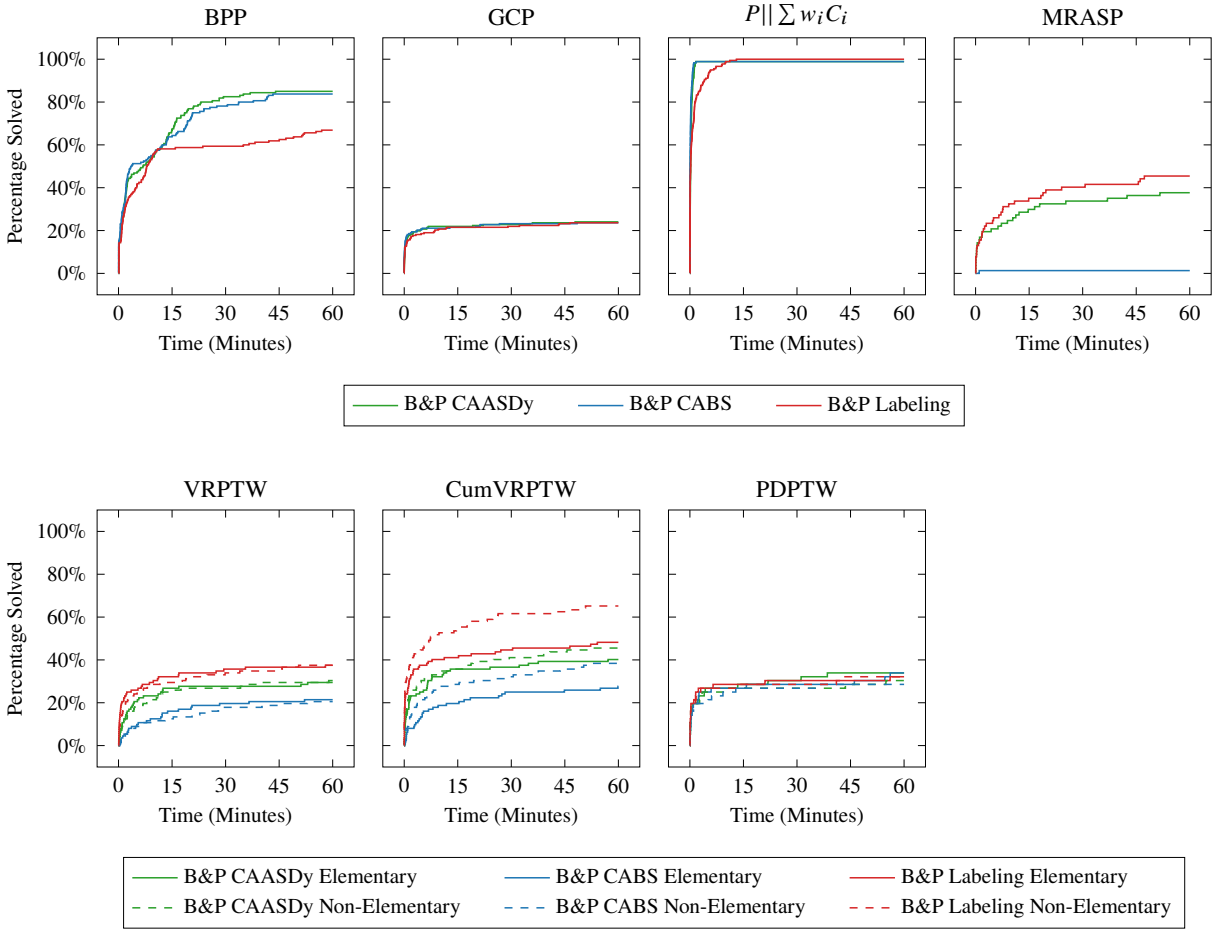
All solvers are single-threaded. All instances are run in parallel for 1 hour on an Intel Xeon Gold 6338 CPU with 64 cores.

5.3. Comparison of DIDP Algorithms for Pricing

Figure 1 compares the performance of branch-and-price backed by the three DIDP pricers. For the BPP, branch-and-price using CAASDy pricing is superior to the other two approaches, challenging the widespread adoption of the labeling algorithm for pricing. For the GCP, the three pricing methods perform almost identically, with CAASDy marginally ahead. For $P||\sum w_i C_i$, CAASDy and CABS are better at the beginning, but only the labeling algorithm is able to close all instances. We observe this performance difference despite the fact that the pricing problem does not have any resource variables. In such a case, the labeling algorithm is still different from CAASDy in that it orders states by their g-values first and then h-values for breaking ties. This ordering possibly results in the observed difference. For the MRASP, the labeling algorithm performs best and CABS almost entirely fails. The traditional labeling algorithm performs significantly better than CABS and CAASDy on the VRPTW and CumVRPTW. The elementary variant ramps up faster than the non-elementary variant but they both close the same number of instances at time-out for the VRPTW. However, the non-elementary version performs significantly better for the CumVRPTW. For the PDPTW, the elementary version of CAASDy has a small lead on the others.

These findings demonstrate that the labeling algorithm, originally developed for vehicle routing, is unchallenged in its intended application domain. Nonetheless, CAASDy is slightly better on three of the seven problem classes, including the PDPTW, which is traditionally priced using labeling.

Figure 1 **Percentage of instances solved over time using branch-and-price with various DIDP pricers.**



These results indicate that search methods developed by the AI community are capable of solving the pricing problem in column generation and could have a meaningful role given further development.

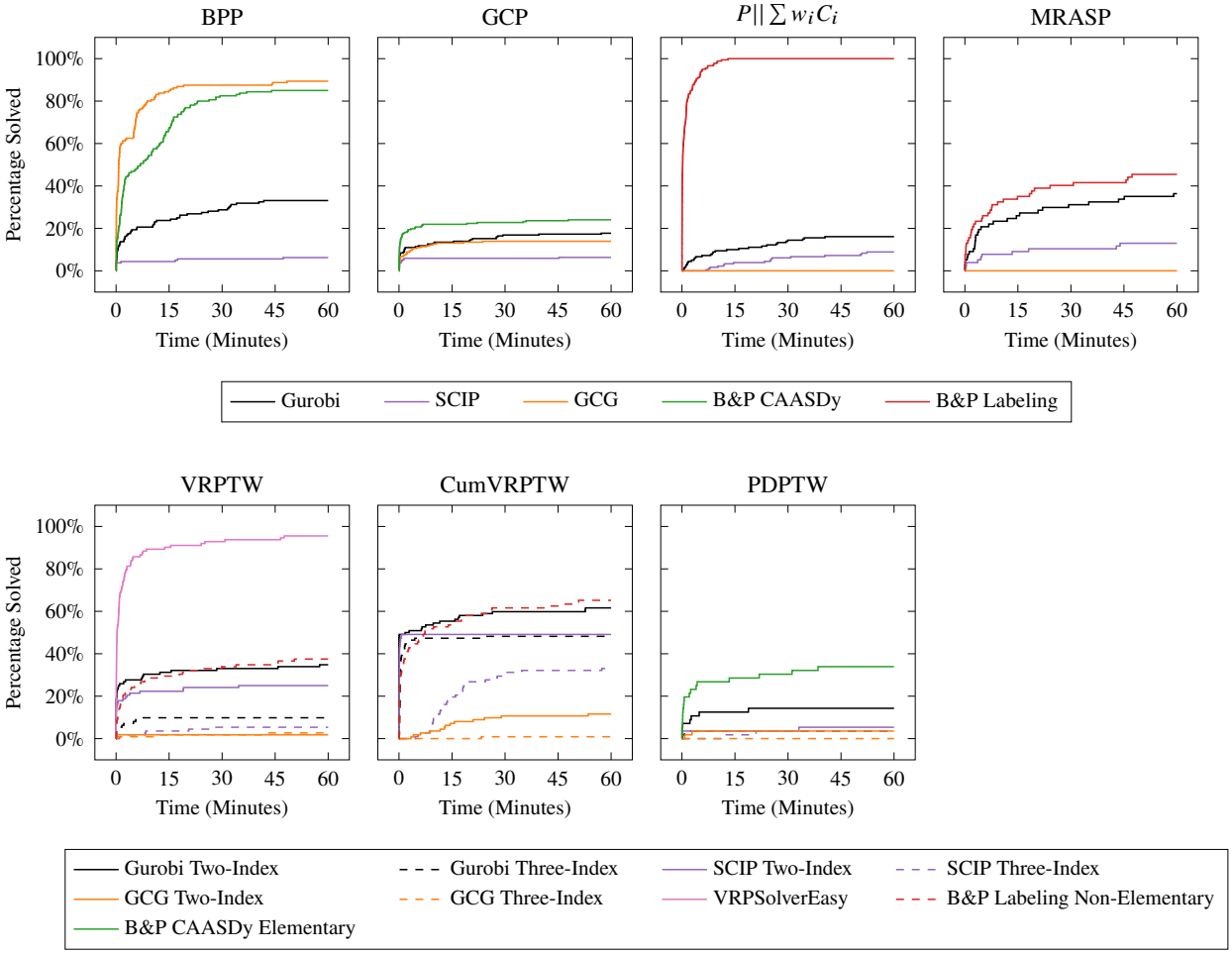
5.4. Comparison Against Other Solvers

Figure 2 compares the best branch-and-price method against the other solvers.

Bin Packing Problem The BPP has simple structure and is one of the standard benchmarks for column generation. The pricing problem takes the form of a knapsack problem, which is known to be easily solved by MIP. The performance of GCG and its generic MIP pricer clearly reflect this observation. Branch-and-price using CAASDy is almost as effective as GCG at time-out but initially ramps up slower. Gurobi performs substantially worse, demonstrating that exploiting problem structure is essential to achieving high performance.

Graph Coloring Problem Branch-and-price with CAASDy dominates the other solvers. GCG is slightly behind Gurobi, less so at the start and more later on. In the compact formulations, the GCP has massive symmetry in the index of the color. The small lead by Gurobi suggests that it is

Figure 2 **Percentage of instances solved over time using other solvers compared to branch-and-price.**



hindered by the symmetry, whereas GCG takes advantage of a column generation model in which this symmetry is entirely absent.

Parallel Machine Scheduling Problem In $P||\sum w_i C_i$, branch-and-price solves all instances in nearly 15 minutes, while other approaches solve less than 20% of instances in 60 minutes. This result shows a large advantage of branch-and-price over the compact formulation. GCG completely fails, demonstrating that unsuitable pricing schemes are catastrophic whereas appropriate pricing algorithms make branch-and-price superior to even commercial solvers.

Multi-Runway Aircraft Scheduling Problem The MRASP, with an SPPRC, again demonstrates that this type of problem structure is well-suited to branch-and-price. Branch-and-price using a labeling algorithm performs best, with Gurobi in second place. GCG fails to solve any instance.

Vehicle Routing Problem with Time Windows The VRPTW is the quintessential example of successful branch-and-price when paired with an appropriate pricer. GCG fails at this problem because its MIP pricer is unsuitable. The naive implementation of branch-and-price solves several

more instances than Gurobi, again indicating that properly exploiting the problem structure is highly beneficial. VRPSolverEasy solves nearly all the instances, demonstrating how advantageous a problem-specific solver can be.

Cumulative Vehicle Routing Problem with Time Windows Branch-and-price with the labeling algorithm is superior at the CumVRPTW, beating Gurobi by a few instances. It is interesting that GCG solves more instances using the two-index model than the three-index model. The two-index model does not replicate each vehicle with a different index, i.e., there is no symmetry in the index of the vehicle. However, the third index gives rise to the block diagonal structure of the matrix that GCG uses for its automatic Dantzig-Wolfe reformulation. This highly unusual result suggests that GCG chose a poor reformulation in the three-index model.

Pickup and Delivery Problem with Time Windows For the PDPTW, branch-and-price using CAASDy on the elementary variant is significantly better than Gurobi. Both SCIP and GCG perform poorly on this problem. GCG again exhibits unexpected behavior regarding its choice of Dantzig-Wolfe reformulation.

5.5. Main Findings

Across our benchmarks, the best pricing strategy is problem-dependent: branch-and-price with either CAASDy or labeling performs best. In line with conventional practice in column generation, labeling is effective when the pricing problem is an SPPRC.

Notably, CAASDy is a search method originating in the AI community. These observations motivate deeper cross-fertilization between mathematical programming and AI planning, and point to several promising directions for future work.

The experiments also prove that DIDP is valuable for rapid prototyping of column generation solvers. Despite the simplicity and naivety of the current branch-and-price solvers, they already outperform a state-of-the-art commercial solver on static formulations in a few cases. This demonstrates that a flexible modeling layer, paired with a reusable library of search algorithms, can provide quick proof-of-concept evidence for (or against) a column generation approach before investing in a bespoke high-performance implementation.

6. Conclusion

This paper introduces four new features in DIDP relevant to pricing in column generation, including a dual bound function based on the fractional knapsack problem and a generic labeling algorithm applicable to any model.

Using these tools, we build straightforward branch-and-price solvers for seven problem classes by modeling the pricing problem as a dynamic program and selecting a solver from the DIDP library. Despite their simplicity, these solvers perform better than a state-of-the-art commercial solver on compact MIP formulations.

While the generic algorithms in DIDP are not intended to compete against bespoke codes, such as VRPSolverEasy, they are useful for obtaining early-stage indications that a bespoke column generation approach could be worthwhile.

Finally, the success of CAASDy, originally developed in the AI planning community, highlights the benefits of cross-fertilization with researchers outside mathematical programming. Closer collaboration with researchers in AI planning is a promising direction for future work.

References

Achterberg T, Berthold T, Koch T, Wolter K (2008) Constraint integer programming: A new approach to integrate CP and MIP. Perron L, Trick MA, eds., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 6–20 (Springer Berlin Heidelberg).

Bellman R (1957) *Dynamic Programming* (Princeton University Press).

Corona-Gutiérrez K, Nucamendi-Guillén S, Lalla-Ruiz E (2022) Vehicle routing with cumulative objectives: A state of the art and analysis. *Computers & Industrial Engineering* 169:108054.

Dantzig GB (1957) Discrete-variable extremum problems. *Operations Research* 5(2):266–277.

de Aragao MP, Uchoa E (2003) Integer program reformulation for robust branch-and-cut-and-price algorithms. *Mathematical Program in Rio: a conference in honour of Nelson Maculan*, 56–61.

Delorme M, Iori M, Martello S (2016) Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research* 255(1):1–20, ISSN 0377-2217.

Delorme M, Iori M, Martello S (2018) BPPLIB: a library for bin packing and cutting stock problems. *Optimization Letters* 12(2):235–250.

Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. *INFORMS Journal on Computing* 40:342–354.

Dijkstra EW (1959) A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Eastman WL, Even S, Isaacs IM (1964) Bounds for the optimal scheduling of *n* jobs on *m* processors. *Management Science* 11(2):268–279.

Edelkamp S, Jabbar S, Lafuente AL (2005) Cost-algebraic heuristic search. *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, 1362–1367 (AAAI Press).

Elmaghraby SE, Park SH (1974) Scheduling jobs on a number of identical machines. *AIIE Transactions* 6(1):1–13.

Errami N, Queiroga E, Sadykov R, Uchoa E (2024) VRPSolverEasy: A Python library for the exact solution of a rich vehicle routing problem. *INFORMS Journal on Computing* 36(4):956–965.

Falkenauer E (1996) A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics* 2(1):5–30.

Fernández Gil A, Gómez Sánchez M, Lalla-Ruiz E, Castro C (2020) Cumulative VRP with time windows: A trade-off analysis. Lalla-Ruiz E, Mes M, Voß S, eds., *Computational Logistics*, 277–291 (Cham: Springer International Publishing).

Fukasawa R, Longo H, Lysgaard J, De Aragão MP, Reis M, Uchoa E, Werneck RF (2006) Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming* 106(3):491–511.

Furtado MGS, Munari P, Morabito R (2017) Pickup and delivery problem with time windows: A new compact two-index formulation. *Operations Research Letters* 45(4):334–341.

Gamrath G, Lübbecke ME (2010) Experiments with a generic Dantzig-Wolfe decomposition for integer programs. Festa P, ed., *Experimental Algorithms. Proceedings*, 239–252 (Berlin, Heidelberg: Springer Berlin Heidelberg).

Ghoniem A, Farhadi F, Reihaneh M (2015) An accelerated branch-and-price algorithm for multiple-runway aircraft sequencing problems. *European Journal of Operational Research* 246(1):34–43.

Gurobi Optimization, LLC (2024) Gurobi Optimizer Reference Manual. URL `https://www.gurobi.com`.

Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* 4(2):100–107.

IBM (2024) Ibm ilog cplex optimization studio. URL `https://www.ibm.com/products/ilog-cplex-optimization-studio`.

Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. Desaulniers G, Desrosiers J, Solomon MM, eds., *Column Generation*, 33–65 (Boston, MA: Springer US).

Karp RM (1972) Reducibility among combinatorial problems. Miller RE, Thatcher JW, Bohlinger JD, eds., *Complexity of Computer Computations*, 85–103, The IBM Research Symposia Series (Boston, MA: Springer US).

Kuroiwa R, Beck JC (2023a) Domain-independent dynamic programming:generic state space search for combinatorial optimization. Koenig S, Stern R, Vallati M, eds., *International Conference on Automated Planning and Scheduling (ICAPS)*, 236–244 (AAAI Press).

Kuroiwa R, Beck JC (2023b) Solving domain-independent dynamic programming problems with anytime heuristic search. Koenig S, Stern R, Vallati M, eds., *International Conference on Automated Planning and Scheduling (ICAPS)*, 245–253 (AAAI Press).

Kuroiwa R, Beck JC (2025a) Domain-independent dynamic programming. URL `https://arxiv.org/abs/2401.13883`.

Kuroiwa R, Beck JC (2025b) RPID: Rust Programmable Interface for Domain-Independent Dynamic Programming. *Principles and Practice of Constraint Programming (CP)*, volume 340, 23:1–23:21 (Dagstuhl, Germany: Schloss Dagstuhl).

Letchford AN, Salazar-González JJ (2006) Projection results for vehicle routing. *Mathematical Programming* 105(2):251–274.

Li H, Lim A (2001) A metaheuristic for the pickup and delivery problem with time windows. *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 160–167.

Lübbecke ME, Desrosiers J (2005) Selected topics in column generation. *Operations Research* 53(6):1007–1023.

Malaguti E, Toth P (2010) A survey on vertex coloring problems. *International Transactions in Operational Research* 17(1):1–34.

Pessoa A, Sadykov R, Uchoa E, Vanderbeck F (2020) A generic exact solver for vehicle routing and related problems. *Mathematical Programming* 183(1):483–523.

Pugliese LDP, Guerriero F (2013) A survey of resource constrained shortest path problems: Exact solution approaches. *Networks* 62(3):183–200.

Ropke S, Cordeau JF (2009) Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science* 43(3):267–286.

Russell S, Norvig P (2020) Solving problems by searching. *Artificial Intelligence: A Modern Approach*, chapter 3, 63–109 (Pearson), fourth edition.

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

van den Akker JM, Hoogeveen JA, van de Velde SL (1999) Parallel machine scheduling by column generation. *Operations Research* 47(6):862–872.

Vigo D, Toth P, eds. (2014) *Vehicle Routing: Problems, Methods, and Applications*. MOS-SIAM Series on Optimization (Society for Industrial and Applied Mathematics), second edition.

Zhang W (1998) Complete anytime beam search. *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, 425–430 (AAAI Press).

https://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING

Vol. 00, No. 0, Xxxxx 0000, pp. 000–000

Submitted to *INFORMS Journal on Computing*

ISSN 0899-1499, EISSN 1526-5528

Appendix to: Column Generation Using Domain-Independent Dynamic Programming

Ryo Kuroiwa
National Institute of Informatics / The Graduate University of Advanced Studies, SOKENDAI, Japan, kuroiwa@nii.ac.jp

Edward Lam
Monash University, Australia, edward.lam@monash.edu

1. Bin Packing Problem

Given a set $\mathcal{B}$ of identical bins with capacity $Q \geq 0$ and a set $\mathcal{I}$ of items, where each item $i \in \mathcal{I}$ has weight $w_i \geq 0$, BPP assigns every item to one bin such that the total weight of all items assigned to each bin does not exceed its capacity.

Problem (1) is the master problem, where $\mathcal{P} \subseteq 2^{\mathcal{I}}$ is the set of patterns, each representing a set of items in the same bin, and λ_p indicates a bin is activated to store the items in the pattern p.

$$\min \sum_{p \in \mathcal{P}} \lambda_p \tag{1a}$$

$$\sum_{p \in \mathcal{P}} a_{i,p} \lambda_p \geq 1 \qquad \forall i \in \mathcal{I} \tag{1b}$$

$$\lambda_p \in \mathbb{Z}_+ \qquad \forall p \in \mathcal{P}. \tag{1c}$$

Ryan-Foster branching is executed on the items (Foster and Ryan 1976), which selects two items and decides if they are paired (must appear in the same bin) or conflicting (must appear in different bins). In each child node, patterns incompatible with the decision are removed from $\mathcal{P}$.

The items are partitioned into groups such that all items within a group are paired. Unpaired items are placed into a singleton group containing only itself. Let $\mathcal{G} = \{0, \dots, |\mathcal{G}| - 1\}$ denote the set of item groups. For each item group $g \in \mathcal{G}$, define $w_g = \sum_{i \in g} w_i$ as the total weight and $\pi_g = \sum_{i \in g} \pi_i$ as the total dual values of all items in g, where $\pi_i \geq 0$ is the dual variable of Constraint (1b) in the linear relaxation. Furthermore, for each group $g \in \mathcal{G}$, define its conflicting groups $\mathcal{H}_g \subseteq \mathcal{G}$ as the groups containing at least one item in conflict with any item in g.

The pricing problem is a variant of the 0-1 knapsack problem to find a set of items compatible with the branching decisions. Problem (2) shows the DP model of the pricing problem. Define $V(g, q, \mathcal{R})$

as the value function of a state $(g, q, \mathcal{R})$ given by an element variable $g \in \mathcal{G} \cup \{|\mathcal{G}|\}$ representing the current item group in consideration, where the value $|\mathcal{G}|$ indicates a dummy terminal value, a numeric variable q representing the remaining capacity, and a set variable $\mathcal{R}$ representing the set of reachable item groups (i.e., groups not yet committed and are compatible with the groups already committed). Objective (2a) defines the sought value, where the -1 constant arises from the cost of all patterns in Objective (1a) according to the column generation framework.

Equation (2b) defines the base case, which terminates the computation when reaching a dummy state with $g = |\mathcal{G}|$. Equation (2c) minimizes the value function over two choices: include group g or not. Inequality (2d) expresses that a state $(g, q_1, \mathcal{R}_1)$ at group g dominates another state $(g, q_2, \mathcal{R}_2)$, also at group g, if its remaining capacity q_1 is larger and its reachable set $\mathcal{R}_1$ is a superset. Inequality (2e) defines a lower bound on as the fractional knapsack problem.

$$\text{compute } V(0, Q, \mathcal{G}) - 1 \tag{2a}$$

$$V(|\mathcal{G}|, q, \mathcal{R}) = 0 \tag{2b}$$

$$V(g, q, \mathcal{R}) = \min \begin{cases} V(g+1, q - w_g, \mathcal{R} \setminus (\mathcal{H}_g \cup \{g' : w_{g'} > q - w_g\})) - \pi_g & \text{if } g \in \mathcal{R} \\ V(g+1, q, \mathcal{R} \setminus \{g\}) \end{cases} \tag{2c}$$

$$V(g, q_1, \mathcal{R}_1) \leq V(g, q_2, \mathcal{R}_2) \text{ if } q_2 \leq q_1 \wedge \mathcal{R}_2 \subseteq \mathcal{R}_1 \tag{2d}$$

$$V(g, q, \mathcal{R}) \geq -\mathsf{fractional\_knapsack}(\mathcal{R}, q, (\pi_i)_{i \in \mathcal{G}}, (w_i)_{i \in \mathcal{G}}) \tag{2e}$$

2. Graph Coloring Problem

The GCP finds the minimum number of colors required to assign a color to every vertex of a given graph $G = (\mathcal{V}, \mathcal{E})$ such that the neighbors of every vertex are assigned a different color.

Problem (3) is the set partitioning problem, a master problem of our column generation model. We use the set of patterns $\mathcal{P} \subseteq 2^{\mathcal{V}}$, each of which represents the set of vertices assigned to the same color. A variable $\lambda_p \in \mathbb{Z}_+$ indicates if the pattern is activated.

$$\min \sum_{p \in \mathcal{P}} \lambda_p \tag{3a}$$

$$\sum_{p \in \mathcal{P}} a_{i,p} \lambda_p = 1 \qquad \forall i \in \mathcal{V} \tag{3b}$$

$$\lambda_p \in \mathbb{Z}_+ \qquad \forall p \in \mathcal{P}. \tag{3c}$$

We use Ryan-Foster branching that selects two vertices and decides if they are assigned the same color (paired) or different colors (conflicting). To correctly define a pricing problem compatible

with these branching decisions, define the set $\mathcal{G} = \{0, \ldots, |\mathcal{G}| - 1\}$ of vertex groups as a partition of the vertices. All paired vertices are collected into one vertex group, and unpaired vertices are placed in singletons. Let $\pi_g = \sum_{i \in g} \pi_i$ be the total dual value of all vertices in group $g \in \mathcal{G}$. Let $\mathcal{H}_g \subseteq \mathcal{G}$ be the groups that contain at least one conflicting vertex with the vertices in $g \in \mathcal{G}$.

The pricing problem decides whether the vertices in each vertex group are included or excluded in a pattern, similarly to that of bin packing, but without the capacity constraint. Let $V(g, \mathcal{R})$ be the value function of a state $(g, \mathcal{R})$ defined by the current group $g \in \mathcal{G} \cup \{|\mathcal{G}|\}$, where $|\mathcal{G}|$ is a dummy terminating value, and the reachable groups $\mathcal{R}$ (groups not yet examined and not in conflict with groups already committed). Problem (4) presents the DP formulation. Inequality (4e) defines a lower bound, excluding the reduced cost of item groups if it is positive.

$$\text{compute } V(0, \mathcal{G}) - 1 \tag{4a}$$

$$V(|\mathcal{G}|, \mathcal{R}) = 0 \tag{4b}$$

$$V(g, \mathcal{R}) = \min \left\{ V(g+1, \mathcal{R} \setminus \mathcal{H}_g) - \pi_g, V(g+1, \mathcal{R} \setminus \{g\}) \text{ if } g \in \mathcal{R} \right\} \tag{4c}$$

$$V(g, \mathcal{R}_1) \leq V(g, \mathcal{R}_2) \text{ if } \mathcal{R}_2 \subseteq \mathcal{R}_1 \tag{4d}$$

$$V(g, \mathcal{R}) \geq \sum_{g' \in \mathcal{R}} \min(-\pi_{g'}, 0) \tag{4e}$$

3. Parallel Machine Scheduling

In $P||\sum w_i C_i$, a set of n jobs $\mathcal{J} = \{1, ..., n\}$ is scheduled on a set of m identical machines $\mathcal{M} = \{1, ..., m\}$, where each job $j \in \mathcal{J}$ has processing time p_j and weight w_j. The objective is to minimize the total weighted completion time. Elmaghraby and Park (1974) show that, given a set of jobs assigned to the same machine, scheduling j before k results in a better or equal objective value if $w_j/p_j \leq w_k/p_k$. Without loss of generality, we assume that jobs are ordered so that $w_j/p_j \leq w_{j+1}/p_{j+1}$ for $j = 1, ..., n-1$. We also use the minimum start time r_j and the maximum completion time d_j of job j derived from theoretical analysis by van den Akker et al. (1999).

In the compact formulation in Problem (5), $x_{i,j}$ represents that job j is scheduled on machine i if $x_{i,j} = 1$, and C_j represents the completion time of job j. Constraint (5b) ensures that if j is scheduled

on i, then C_j is not smaller than the total processing time of j and its predecessors.

$$\min \sum_{j \in \mathcal{J}} w_j C_j \tag{5a}$$

$$C_j \geq \sum_{k=1}^{j} p_k x_{i,k} - \sum_{k=1}^{j-1} p_k (1 - x_{i,j}) \qquad \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \tag{5b}$$

$$x_{i,j} \in \{0,1\} \qquad \forall i \in \mathcal{M}, \forall j \in \mathcal{J} \tag{5c}$$

$$C_j \in [r_j, d_j] \qquad \forall j \in \mathcal{J}. \tag{5d}$$

Our column generation model is based on van den Akker et al. (1999). The master problem is the set partitioning problem, similar to Problem (3), but now $\mathcal{P}$ is a set of schedules for a single machine. In addition, the objective is $\sum_{s \in \mathcal{P}} c_s \lambda_s$ where c_s is the cost of schedule s, and m machines are used, so we have $\sum_{s \in \mathcal{P}} \lambda_s = m$. Our branching strategy is the same as van den Akker et al. (1999), which changes the release date r_j and the deadline d_j for a selected job j.

The pricing problem finds a schedule minimizing the reduced cost, computed from the dual value π_j for each job j defined by Constraint (3b). Since an optimal schedule executes jobs with no idling time (Elmaghraby and Park 1974), the pricing problem is a variant of the 0-1 knapsack problem of deciding whether job j is included in the schedule, with the time window constraints. In our DP formulation in Problem (6), an element variable j represents the job currently considered, and a numeric variable t represents the current time. We can use $H = \sum_{k \in \mathcal{J}} p_k/m + (m-1) \max_{k \in \mathcal{J}} p_k/m$ as an upper bound on the completion time of a schedule. Therefore, the sum of processing time scheduled from state (j,t) must be less than or equal to $H - t$. In Inequality (6c), we use a dual bound function based on the 0-1 knapsack problem with the set of items $\{j, ..., n\}$ and the capacity $H - t$, where each job k has the profit $\pi_k - w_k(r_k + p_k)$ and the weight p_k.

$$\text{compute } V(1,0) \tag{6a}$$

$$V(j,t) = \begin{cases} 0 & \text{if } j = n+1 \\ \min\left\{-\pi_j + w_j t + V(j+1, t+p_j), V(j+1,t)\right\} & \text{if } r_j \leq t \wedge t + p_j \leq d_j \\ V(j+1,t) & \text{otherwise} \end{cases} \tag{6b}$$

$$V(j,t) \geq -\mathsf{fractional\_knapsack}\left(\{j, ...n\}, H - t, (\pi_k - w_k(r_k + p_k))_{k=j,...,n}, (p_k)_{k=j,...,n}\right). \tag{6c}$$

4. Multi-Runway Aircraft Scheduling Problem

The MRASP schedules a set of heterogeneous aircraft on a set of identical runways while respecting minimum separation times between aircraft and minimizing a weighted sum of scheduled times.

Let $\mathcal{R} = \{1, \ldots, R\}$ be the set of R identical runways, $\mathcal{O} = \{\text{takeoff}, \text{landing}\}$ be the set of operations, and $\mathcal{G}$ be the set of aircraft classes and $\mathcal{A}$ be the set of aircraft. Every aircraft $a \in \mathcal{A}$ is associated with a class $g_a \in \mathcal{G}$, an operation $o_a \in \mathcal{O}$, a release time $u_a \geq 0$, a due time $v_a \geq u_a$ and a cost $c_a \geq 0$.

Every tuple $(g_1, o_1, g_2, o_2) \in \mathcal{G} \times \mathcal{O} \times \mathcal{G} \times \mathcal{O}$ is associated with a minimum separation time $d_{g_1,o_1,g_2,o_2} \geq 0$. An aircraft $a_2 \in \mathcal{A}$ scheduled sometime after $a_1 \in \mathcal{A}$, $a_1 \neq a_2$, on the same runway must occur at least $d_{g_{a_1},o_{a_1},g_{a_2},o_{a_2}}$ later. Since the triangle inequality does not necessarily hold in d, it is insufficient to check the minimum separation time of every aircraft and its immediate successor. Instead, the minimum separation time of every later aircraft must be checked for every aircraft.

Problem (7) is the master problem to select a subset of plans from the set of plans $\mathcal{P}$. Every plan $p \in \mathcal{P}$ is associated with an integer variable λ_p and a cost $c_p \geq 0$.

$$\min \sum_{p \in \mathcal{P}} c_p \lambda_p \tag{7a}$$

$$\sum_{p \in \mathcal{P}} \lambda_p \leq R \tag{7b}$$

$$\sum_{p \in \mathcal{P}} w_{a,p} \lambda_p \geq 1 \qquad \forall a \in \mathcal{A} \tag{7c}$$

$$\lambda_p \in \mathbb{Z}_+ \qquad \forall p \in \mathcal{P}. \tag{7d}$$

The branching rule removes the immediate successor of an aircraft, collected in a matrix $S \in \{0,1\}^{\mathcal{A} \times \mathcal{A}}$ whose elements (i, j) signify whether j can immediate succeed i.

Problem (8) is the DP formulation of the pricing problem. Let $\mathcal{Q} = \{(g, o) \in \mathcal{G} \times \mathcal{O} : \exists g_1 \in \mathcal{G}, o_1 \in \mathcal{O}, g_2 \in \mathcal{G}, o_2 \in \mathcal{O}, d_{g_1,o_1,g_2,o_2} + d_{g_2,o_2,g,o} < d_{g_1,o_1,g,o}\}$ denote the set of class and operation pairs that do not respect the triangle inequality. Every state $S = \left(\mathcal{M}, i, t, \left(e_{g,o}\right)_{(g,o) \in \mathcal{Q}}\right)$ is defined by a set variable $\mathcal{M}$ representing the set of reachable aircraft, an element variable i representing the current, a numeric variable t representing the current time, and a numeric variable $e_{g,o}$ for all $(g, o) \in \mathcal{Q}$ representing the earliest time that class g can perform operation o. In addition, we use a flag f indicating if no more aircraft will be scheduled, i.e., the state is base case iff $f = 1$.

Objective (8a) defines the computation required. It initially begins at a dummy task -1 and the data are appropriately extended with zeros to accommodate this dummy task. Equation (8b) is the recursive equation. The first case defines a base case. In the second case, the outer minimization occurs over two expressions. The first one transitions to a base state. The second expression, the inner minimization, transforms $S = \left(\mathcal{M}, i, t, \left(e_{g,o}\right)_{(g,o) \in \mathcal{Q}}\right)$ into $S'(j) = \left(\mathcal{M}'(j), j, t'(j), \left(e'_{g,o})j\right)\right)$, where $\mathcal{M}'(j) = \mathcal{M} \setminus \{j\} \setminus \{k : t'(j) + d_{g_j,o_j,g_k,o_k} > v_k\}$, $t'(j) = \max\left\{t + d_{g_i,o_i,g_j,o_j}, u_j\right\}$ if $(g_j, o_j) \notin \mathcal{Q}$ and

$t'(j) = \max\left\{t + d_{g_i,o_i,g_j,o_j}, u_j, e_{g_j,o_j}\right\}$ otherwise, and $e'_{g,o}(j) = \max\{e_{g,o}, t'(j) + d_{g_j,o_j,g,o}\}$. The cost is computed from the dual value π_j for aircraft j defined by Constraint (7c). Inequality (8c) states that one state dominates another if its reachable set is larger and all its time variables are earlier. Inequality (8d) defines a lower bound as the sum of reduced costs of all reachable scheduling tasks.

$$\text{compute } V((\mathcal{A}, -1, 0, (0, \ldots, 0)), 0) \tag{8a}$$

$$V(S, f) = \begin{cases} 0 & \text{if } f = 1 \\ \min\left\{V(S, 1), \min_{j \in \mathcal{M}: t'(j) \le v_j} -\pi_j + t'(j)c_j + V(S'(j), f) \right. & \text{if } f = 0 \end{cases} \tag{8b}$$

$$V\left(\left(\mathcal{M}_1, i, t_1, \left(e^1_{g,o}\right)_{(g,o)\in\mathcal{Q}}\right), f\right) \le V\left(\left(\mathcal{M}_2, i, t_2, \left(e^2_{g,o}\right)_{(g,o)\in\mathcal{Q}}\right), f\right) \quad \text{if } \mathcal{M}_2 \subseteq \mathcal{M}_1 \wedge t_1 \le t_2 \wedge e^1_{g,o} \le e^2_{g,o} \forall (g,o) \in \mathcal{Q} \tag{8c}$$

$$V\left(\left(\mathcal{M}, i, t, \left(e_{g,o}\right)_{(g,o)\in\mathcal{Q}}\right), f\right) \ge \sum_{j \in \mathcal{M}} u_j c_j - \pi_j. \tag{8d}$$

5. Vehicle Routing Problem with Time Windows

In the VRPTW, an unlimited number of identical vehicles is initially stationed at a depot, tasked with delivering items to a set of customers and then returning to the depot. Every customer is associated with a load and the total load allocated to a vehicle must respect the vehicle's capacity.

Let n be the number of customers and $\mathcal{N} = \{0, \ldots, n+1\}$ be the set of nodes, where nodes 0 and $n+1$ represent the start and end depot locations respectively. Every node $i \in N$ has a load $l_i \ge 0$, release time $a_i \ge 0$, due time $b_i \ge 0$ and service duration $s_i \ge 0$. Let $\mathcal{A} = \{(i, j) \in \mathcal{N} \times \mathcal{N} : i \ne j, i < n+1, j > 0\}$ be the set of arcs. Every arc $(i, j) \in A$ has a travel distance $d_{i,j} \ge 0$.

The master problem is a set partitioning problem similar to Problem (3), but Constraint (3b) is defined for each customer $i = 1, \ldots, n$. We perform edge branching on the most fractional arc, which disables the arc in one node and enforces it in another. The pricing problem is the SPPRC used as the running example in the main text. In this problem, the travel cost $c_{i,j}$ of arc (i, j) is defined as $-\pi_j + d_{i,j}$, where π_j is the dual value for node j defined by Constraint (3b). Problem (9) shows a DP model for the pricing problem. Following the main text, we use $t'(j) = \max\left\{t + s_i + d_{i,j}, a_j\right\}$ and $\mathcal{R}'(j) = \left\{k \in \mathcal{R} \setminus \{j\} : t'(j) + s_j + d^*_{j,k} \le b_k \wedge q + l_j + l_k \le Q\right\}$ where $d^*_{i,j}$ is the precomputed shortest travel time from i to j. In our implementation, d^*_{ij} is computed once at the beginning using the Floyd-Warshall algorithm and not updated after deleting edges by branching. We define $\mathcal{R}'(n+1) = \mathcal{R}$ since we do not care which customers are reachable once the vehicle has arrived at the end depot.

Inequality (9d) defines a dual bound function as explained in the main text. In addition, we consider the 0-1 knapsack problem, where the remaining time $b_{n+1} - t - d^{\text{in}}_{n+1}$ is the capacity, and the minimum time to visit node j, $w^{\text{in}}_j = d^{\text{in}}_j + s_j$, is the weight. We use similar bounds using $d^{\text{out}}_j = \min_{(j,k)\in\mathcal{A}} d_{j,k}$, $v^{\text{out}}_j = \min\left\{\pi_j - d^{\text{out}}_j\right\}$, and $w^{\text{out}}_j = s_j + d^{\text{out}}_j$, though omitted in Inequality (9d).

$$\text{compute } V(\{1,\dots,n\},0,0,0) \tag{9a}$$

$$V(\mathcal{R},i,q,t) = \begin{cases} 0 & \text{if } i = n+1 \\ \min\limits_{j\in\mathcal{R}\cup\{n+1\}:(i,j)\in\mathcal{A}\wedge t+s_i+d_{i,j}\le b_j} c_{i,j} + V\left(\mathcal{R}'(j), j, q+l_j, t'(j)\right) & \text{otherwise} \end{cases} \tag{9b}$$

$$V(\mathcal{R}_1,i,q_1,t_1) \le V(\mathcal{R}_2,i,q_2,t_2) \text{ if } \mathcal{R}_2 \subseteq \mathcal{R}_1 \wedge q_1 \le q_2 \wedge t_1 \le t_2 \tag{9c}$$

$$V(\mathcal{R},i,q,t) \ge \max\begin{cases} 0 \text{ if } i = n+1 \\ -\mathsf{fractional\_knapsack}\left(\mathcal{R}, Q-q, \left(v^{\text{in}}_j\right)_{j=1,\dots,n}, \left(l_j\right)_{j=1,\dots,n}\right) \\ -\mathsf{fractional\_knapsack}\left(\mathcal{R}, b_{n+1}-t-d^{\text{in}}_{n+1}, \left(v^{\text{in}}_j\right)_{j=1,\dots,n}, \left(w^{\text{in}}_j\right)_{j=1,\dots,n}\right) \end{cases} \tag{9d}$$

We also consider a non-elementary version of the pricing problem, where the same node can be visited multiple times, following previous work (Desrochers et al. 1992). We eliminate 2-cycles, which visit an immediate predecessor, e.g., a subpath (i,j,i). Problem (10) shows a DP model for the relaxed pricing problem. Now, instead of the set of reachable customers $\mathcal{R}$, we maintain an element variable p representing the immediate predecessor. In the target state, we use $p = 0$.

For the dual bound function, we compute the set of reachable customers as $\hat{\mathcal{R}} = \{j \in \mathcal{N}\setminus\{p,i\} : t + s_i + d^*_{i,j} \le b_j \wedge q + l_j \le Q\}$ and an upper bound m_j on the number of visits to each customer. Visiting a customer j requires time at least $\min_{k\in\hat{\mathcal{R}}\cup\{n+1\}}(s_c + d_{cj})$, and leaving from j requires at least $s_j + d^{\text{out}}_j$. The customer j must be visited by the deadline b_j. Therefore, given the current time t, we use the maximum integer m_j satisfying $t + m_j\left(\min_{k\in\hat{\mathcal{R}}\cup\{n+1\}:(k,j)\in\mathcal{A}}(s_k + d_{k,j}) + s_j + d^{\text{out}}_j\right) \le b_j + s_j + d^{\text{out}}_j$. In other words, $m_j = \left\lfloor \frac{b_j + s_j + d^{\text{out}}_j - t}{\min_{k\in\hat{\mathcal{R}}\cup\{n+1\}:(k,j)\in\mathcal{A}}(s_k+d_{k,j})+s_j+d^{\text{out}}_j} \right\rfloor$. Then, we multiply v^{in}_j, l_j, w^{in}_j, v^{out}_j, and w^{out}_j when computing the fractional knapsack bound. Again, we omit the bound based on d^{out}_j.

$$\text{compute } V(0,0,0,0) \tag{10a}$$

$$V(p,i,q,t) = \begin{cases} 0 & \text{if } i = n+1 \\ \min\limits_{j \in \mathcal{N}\setminus\{0,p\}:(i,j)\in\mathcal{A}\setminus\{(0,n+1)\}\wedge q+l_j\le Q\wedge t+s_i+d_{i,j}\le b_j} c_{i,j} + V(i,j,q+l_j,t'(j)) & \text{otherwise} \end{cases} \tag{10b}$$

$$V(p,i,q,t) \le V(p,i,q',t') \text{ if } q \le q' \wedge t \le t' \tag{10c}$$

$$V(p,i,q,t) \ge \max \begin{cases} 0 \text{ if } i = n+1 \\ -\text{fractional\_knapsack}\left(\hat{\mathcal{R}}, Q-q, \left(m_j v_j^{\text{in}}\right)_{j=1,\dots,n}, \left(m_j l_j\right)_{j=1,\dots,n}\right) \\ -\text{fractional\_knapsack}\left(\hat{\mathcal{R}}, b_{n+1}-t-d_{n+1}^{\text{in}}, \left(m_j v_j^{\text{in}}\right)_{j=1,\dots,n}, \left(m_j w_j^{\text{in}}\right)_{j=1,\dots,n}\right) \end{cases} \tag{10d}$$

CumVRPTW modifies VRPTW, so that the objective minimizes the travel distance of each arc multiplied by the load at the origin of the arc. We limit the number to be at most K ($K = 25$ is specified by the Solomon instances). The master problem of the CumVRPTW is the same as Problem (3), with one additional constraint $\sum_{p\in\mathcal{P}} \lambda_p \le K$. The pricing problem is also similar, but its objective function is specific to CumVRPTW. We use $qd_{i,j}$ for the travel cost from node i to j and $v_j^{\text{in}} = \min\{-qd_j^{\text{in}} + \pi_j, 0\}$ and $v_j^{\text{out}} = \min\{-qd_j^{\text{out}} + \pi_j, 0\}$ in the dual bound function.

6. Pickup and Delivery Problem with Time Windows

In the PDPTW, a vehicle picks up a commodity at one customer and delivers it to another customer, while respecting time windows (Dumas et al. 1991). Let m be the number of tasks, $\mathcal{N} = \{1,\dots,n\}$ be the set of tasks, and $\mathcal{L} = \{0,\dots,2n+1\}$ be the number of locations. Each task $i \in \mathcal{N}$ is associated with a pickup location $i \in \mathcal{L}$ and a delivery location $n+i \in \mathcal{L}$. Nodes 0 and $n+1$ represent the start and end depot locations, respectively. Every task $i \in N$ has a load $l_i \ge 0$, and every location $i \in \mathcal{L}$ has release time $a_i \ge 0$, due time $b_i \ge 0$ and service duration $s_i \ge 0$. Let $\mathcal{A} = \{(i,j) \in \mathcal{L}\times\mathcal{L} : i \ne j, i < n+1, j > 0\}$ be the set of arcs. Every arc $(i,j) \in A$ has a travel distance $d_{i,j} \ge 0$. Our objective function is the sum of the number of used vehicles multiplied by a constant penalty u and the total travel distance, where $u = 10000$. For all evaluated methods, we tighten time windows and reduce edges by preprocessing, following Dumas et al. (1991).

Our column generation model is based on Ropke and Cordeau (2009). The master problem is the same as Problem (3) while Constraint (3b) is defined for each pickup location $i = 1,\dots,n$. We use edge branching on the most fractional arc $(i,j) \in \mathcal{A}$. Preprocessing for tightening time windows, reducing edges, and computing the shortest travel time from i to j, $d_{i,j}^*$ using the Floyd-Warshall algorithm is done once at the beginning and not updated after deleting edges by branching.

The pricing problem is a variant of SPPRC considering pickup and delivery and is formulated as DP in Problem (11). The set of available tasks is represented by $\mathcal{R}$, and the set of open tasks, whose pickup locations are visited and delivery locations have not been visited, is represented by O. The current location is represented by i, the current load by q, and the current time by t. We use $t'(j) = \max\{t + s_i + d_{i,j}, a_j\}$. We also represent the set of available tasks after visiting j by $\mathcal{R}'(j) = \{k \in \mathcal{R} \setminus \{j\} : t'(j) + s_j + d^*_{j,k} \leq b_k\}$. The first line corresponds to returning to the depot and is available only if no task is open ($q = 0$). The second and third lines correspond to pickup and delivery, respectively. In addition, Equation (11d) defines redundant information implied by Equation (11c): a state does not lead to a solution if there exists a task j that cannot be completed by the deadline, implemented by state constraints in DIDP.

For the dual bound function, we use the fractional knapsack bound considering the deadline for the end depot. Using $d^{\text{in}}_j = \min_{k \in \mathcal{L}:(k,j) \in \mathcal{A}} d_{k,j}$, to complete the set of open tasks O and return to the end depot, we need at least $d^{\text{in}}(O) = \sum_{j \in O} \left(d^{\text{in}}_{n+j} + s_{n+j} \right) + d^{\text{in}}_{2n+1}$. Completing task j increases the cost by at least $v^{\text{in}}_j = d^{\text{in}}_j - \pi_j + d^{\text{in}}_{n+j}$ and the time by at least $w^{\text{in}}_j = d^{\text{in}}_j + s_j + d^{\text{in}}_{n+j} + s_{n+j}$. We consider the 0-1 knapsack problem with the capacity $b_{2n1} - t - d^{\text{in}}(O)$, where each item $j \in \mathcal{R}$ has profit v^{in}_j and weight w^{in}_j. We also use a similar bound using $d^{\text{out}}_j = \min_{k \in \mathcal{L}:(j,k) \in \mathcal{A}} d_{j,k}$, omitted in Problem (11).

$$\text{compute } V(\mathcal{N}, \emptyset, 0, 0, 0) \tag{11a}$$

$$V(\mathcal{R}, O, 2n+1, q, t) = 0 \tag{11b}$$

$$V(\mathcal{R}, O, i, q, t) = \min \begin{cases} d_{i,2n+1} + V(\mathcal{R}, O, 2n+1, q, t'(2n+1)) \text{ if } \begin{array}{l} (i, 2n+1) \in \mathcal{A} \wedge q = 0 \\ \wedge\, t'(2n+1) \leq b_{2n+1} \end{array} \\ \min\limits_{j \in \mathcal{R}:(i,j) \in \mathcal{A} \wedge q + l_j \leq Q \wedge t'(j) \leq b_j} d_{i,j} - \pi_i + V(\mathcal{R}'(j), O \cup \{j\}, j, q + l_j, t'(j)) \\ \min\limits_{j \in O:(i,j) \in \mathcal{A} \wedge t'(n+j) \leq b_{n+j}} d_{i,n+j} + V(\mathcal{R}'(n+j), O \setminus \{j\}, n+j, q - l_j, t'(n+j)) \end{cases} \tag{11c}$$

$$V(\mathcal{R}, O, i, q, t) = \infty \text{ if } \exists j \in O, t + s_i + d^*_{i,n+j} > b_{n+j} \tag{11d}$$

$$V(\mathcal{R}_1, O, i, q_1, t_1) \leq V(\mathcal{R}_2, O, i, q_2, t_2) \text{ if } \mathcal{R}_2 \subseteq \mathcal{R}_1 \wedge q_1 \leq q_2 \wedge t_1 \leq t_2 \tag{11e}$$

$$V(\mathcal{R}, O, 2n_1, q, t) \geq 0 \tag{11f}$$

$$V(\mathcal{R}, O, i, q, t) \geq -\mathsf{fractional\_knapsack}\left(\mathcal{R}, b_{2n+1} - t - d^{\text{in}}(O), \left(v^{\text{in}}_j\right)_{j=1,\ldots,n}, \left(w^{\text{in}}_j\right)_{j=1,\ldots,n}\right). \tag{11g}$$

In the non-elementary version in Problem (12), we allow completing the same task multiple times, and thus $\mathcal{R}$ is removed. We still maintain the set of open tasks O to ensure that a delivery location is visited after a pickup location.

For the dual bound, we compute the set of available tasks as $\hat{\mathcal{R}} = \{j \in \mathcal{N} \setminus O : t + s_i + d^*_{i,j} \leq b_j\}$. We also compute an upper bound m_j on the number of times task j is completed. Assuming $(n+j, j) \notin \mathcal{A}$, completing task j requires at least time $\min_{k \in \mathcal{L}:(k,j)\in\mathcal{A}} \left(s_k + d_{k,j}\right) + s_j + d^*_{j,n+j} + s_{n+j} + d^{\text{out}}_{n+j}$. Therefore, $m_j = \left\lfloor \frac{b_{n+j} + s_{n+j} + d^{\text{out}}_{n+j}}{\min_{k \in \mathcal{L}:(k,j)\in\mathcal{A}} \left(s_k + d_{k,j}\right) + s_j + d^*_{j,n+j} + s_{n+j} + d^{\text{out}}_{n+j}} \right\rfloor$.

$$\text{compute } V(\emptyset, 0, 0, 0) \tag{12a}$$

$$V(O, 2n+1, q, t) = 0 \tag{12b}$$

$$V(O, i, q, t) = \min \begin{cases} d_{i,2n+1} + V(O, 2n+1, q, t'(2n+1)) \text{ if } \begin{array}{l}(i, 2n+1) \in \mathcal{A} \wedge q = 0 \wedge \\ t'(2n+1) \leq b_{2n+1}\end{array} \\ \min\limits_{j \in \mathcal{N} \setminus O:(i,j)\in\mathcal{A} \wedge q + l_j \leq Q \wedge t'(j) \leq b_j} d_{i,j} - \pi_i + V(O, j, q + l_j, t'(j)) \\ \min\limits_{j \in O:(i,j)\in\mathcal{A} \wedge t'(n+j) \leq b_{n+j}} d_{i,n+j} + V(O \setminus \{j\}, n+j, q - l_j, t'(n+j)) \end{cases} \tag{12c}$$

$$V(O, i, q, t) = \infty \text{ if } \exists j \in O, t + s_i + d^*_{i,n+j} > b_{n+j} \tag{12d}$$

$$V(O, i, q_1, t_1) \leq V(O, i, q_2, t_2) \text{ if } q_1 \leq q_2 \wedge t_1 \leq t_2 \tag{12e}$$

$$V(O, 2n+1, q, t) \geq 0 \tag{12f}$$

$$V(O, i, q, t) \geq -\text{fractional\_knapsack}\left(\hat{\mathcal{R}}, b_{2n+1} - t - d^{\text{in}}(O), \left(m_j v^{\text{in}}_j\right)_{j=1,\ldots,n}, \left(m_j w^{\text{in}}_j\right)_{j=1,\ldots,n}\right). \tag{12g}$$

References

Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. *INFORMS Journal on Computing* 40:342–354.

Dumas Y, Desrosiers J, Soumis F (1991) The pickup and delivery problem with time windows. *European Journal of Operational Research* 54(1):7–22.

Elmaghraby SE, Park SH (1974) Scheduling jobs on a number of identical machines. *AIIE Transactions* 6(1):1–13.

Foster BA, Ryan DM (1976) An integer programming approach to the vehicle scheduling problem. *Operational Research Quarterly* 27(2):367–384.

Ropke S, Cordeau JF (2009) Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science* 43(3):267–286.

van den Akker JM, Hoogeveen JA, van de Velde SL (1999) Parallel machine scheduling by column generation. *Operations Research* 47(6):862–872.